Enhancing Security: A Deep Learning Approach for Automated Weapon Detection

A Project Report submitted in the partial fulfillment of the Requirements for the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

B. Pavan Dath (22475A0517)
A. Raja Vamsi (22475A0508)
M. Venkata Sai (21471A05N7)

Under the esteemed guidance of

Dr. K. Suresh Babu, M.Tech, Ph.D.,

Assoc. Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASAROPET
(AUTONOMOUS)

Accredited by NAAC with A+ Grade and NBA under Tyre -1 NIRF rank in the band of 201-300 and an ISO 9001:2015 Certified

Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada

KOTAPPAKONDA ROAD, YALLAMANDA VILLAGE, NARASARAOPET- 522601

2024-2025

NARASARAOPETA ENGINEERING COLLEGE: NARASARAOPET
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project work entitled **"Enhancing Security: A Deep Learning Approach for Automated Weapon Detection"** is a bonafide work done by the team B. Pavan Dath(22475A0517), A. Raja Vamsi(22475A0508), M. Venkata Sai(21471A05N7) in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in the **Department of COMPUTER SCIENCE AND ENGINEERING** during 2024-2025.

PROJECT GUIDE

Dr. K. Suresh Babu, M.Tech, Ph.D

Assoc. Professor

PROJECT CO-ORDINATOR

D.Venakata Reddy, M.Tech, (Ph.D)

Assistant Professor

HEAD OF THE DEPARTMENT

Dr. S. N. Tirumala Rao, M.Tech., Ph.D

Professor & HoD

EXTERNAL EXAMINER

DECLARATION

We declare that this project work titled "**Enhancing Security: A Deep Learning Approach for Automated Weapon Detection**" is composed by my own that the work contain here is my own except where explicitly stated otherwise in the text and that this work has been submitted for any other degree or professional qualification except as specified.

B. Pavan Dath (22475A0517)
A. RajaVamsi (22475A0508)
M. Venkata Sai (21471A05N7)

ACKNOWLEDGEMENT

We wish to express our thanks to carious personalities who are responsible for the completion of the project. We are extremely thankful to our beloved chairman sri **M. V. Koteswara Rao**, B.Sc., who took keen interest in us in every effort throughout this course. We owe out sincere gratitude to our beloved principal **Dr. S. Venkateswarlu**, M.Tech, Ph.D., for showing his kind attention and valuable guidance throughout the course.

We express our deep felt gratitude towards **Dr. S. N. Tirumala Rao,** M.Tech, Ph.D., HOD of CSE department and also to our guide **Dr**. **K. Suresh Babu,** M.Tech, Ph.D., of CSE department whose valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

We extend our sincere thanks towards **Mr. D. Venkata Reddy,** M.Tech, (Ph.D)., Assistant Professor & Project coordinator of the project for extending her encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

We extend our sincere thanks to all other teaching and non-teaching staff to department for their cooperation and encouragement during our B.Tech degree.

We have no words to acknowledge the warm affection, constant inspiration and encouragement that we received from our parents.

We affectionately acknowledge the encouragement received from our friends and those who involved in giving valuable suggestions had clarifying out doubts which had really helped us in successfully completing our project.

By

B. Pavan Dath (22475A0517)
A. Raja Vamsi (22475A0508)
M. Venkata Sai (21471A05N7)

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

INSTITUTE VISION AND MISSION

INSTITUTION VISION

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community,

INSTITUTION MISSION

M1: Provide the best class infra-structure to explore the field of engineering and research

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a Centre of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

MISSION OF THE DEPARTMENT

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertise in modern software tools and technologies to cater to the real time requirements of the Industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Specific Outcomes (PSO's)

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneur, environmental and social issues.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Educational Objectives (PEO's)

The graduates of the programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in software industry.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Outcomes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Project Course Outcomes (CO'S):

C421.1: Analyse the System of Examinations and identify the problem.

C421.2: Identify andclassify the requirements.

C421.3: Review the Related Literature

C421.4: Design and Modularize the project

C421.5: Construct, Integrate, Test and Implement the Project.

C421.6: Prepare the project Documentation and present the Report using appropriate method.

Course Outcomes – Program Outcomes Mapping

	P01	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1		✓											✓		
C421.2	✓		✓		✓								✓		
C421.3				✓		✓	✓	✓					✓		
C421.4			✓			✓	✓	✓					✓	✓	
C421.5					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C421.6									✓	✓	✓		✓	✓	

Course Outcomes – Program Outcome Correlation

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1	2	3											2		
C421.2			2		3								2		
C421.3				2		2	3	3					2		
C421.4			2			1	1	2					3	2	
C421.5					3	3	3	2	3	2	2	1	3	2	1
C421.6									3	2	1		2	3	

Note: The values in the above table represent the level of correlation between CO's and PO's:

1. Low level

2. Medium level

3. High level

Project mapping with various courses of Curriculum with Attained PO's:

Name of the course from which principles are applied in this project	Description of the device	Attained PO
C2204.2, C22L3.2	Gathering the requirements and defining the problem, plan to develop a Enhancing Security: A Deep Learning Approach for Automated Weapon Detection.	PO1, PO3
CC421.1,C2204.3, C22L3.2	Each and every requirement is critically analyzed, the process model is identified	PO2, PO3
CC421.2,C2204.2,C22L3.3	Logical design is done by using the unified modelling language which involves individual team work	PO3, PO5, PO9
CC421.3,C2204.3,C22L3.2	Each and every module is tested, integrated, and evaluated in our project	PO1, PO5
CC421.4,C2204.4,C22L3.2	Documentation is done by all our four members in the form of a group	PO10
CC421.5,C2204.2,C22L3.3	Each and every phase of the work in group is presented periodically	PO10, PO11
C2202.2,C2203.3,C1206.3, C3204.3,C4110.2	Implementation is done and the project will be handled by the Detecting 8 categories of Weapons.	PO4, PO7
C32SC4.3	The physical design includes hardware components like system type of intel®core™i7-7500UCPU@2.70gh ,Cache memory of 4MB and RAM of 8GB.	PO5, PO6

ABSTRACT

The recognition of weapons from images plays a pivotal role in ensuring public safety and national security. In the current landscape of increasing firearm-related incidents, the demand for accurate and efficient weapon detection systems has never been more urgent. Traditional weapon recognition systems, which rely on handcrafted features and conventional computer vision methods, often struggle to adapt to the diverse range of weapons and environmental conditions, leading to reduced accuracy and an increased likelihood of false negatives or positives. To overcome these challenges, our project introduces a deep learning-based approach to weapon detection, leveraging the power of Convolutional Neural Networks (CNNs) to automatically learn and extract discriminative features from weapon images.

This system demonstrates remarkable versatility, capable of detecting various types of weapons, such as handguns, rifles, and knives, even under different lighting conditions and complex backgrounds. By exploring pre-trained models like VGG16, ResNet50, and ResNet101, alongside a custom CNN architecture, the project ensures robust performance through meticulous preprocessing, including resizing and normalization, and the implementation of advanced training techniques like early stopping and checkpointing. The models are evaluated on a separate test set, showcasing their ability to achieve high accuracy while minimizing loss. This work represents a significant leap forward in harnessing machine learning to enhance public safety and security measures. By automating weapon detection with precision and reliability, the project contributes to reducing risks associated with weapon-related incidents, paving the way for further innovations in AI-driven security solutions.

INDEX

List of Figures

1. INTRODUCTION

The rapid progress in deep learning has revolutionized automated image analysis, resulting in significant strides in object detection and classification. These advancements have been thoroughly documented in research, showcasing the potential of neural networks to accurately process intricate visual data [1]. Convolutional Neural Networks (CNNs) have emerged as the cornerstone of modern deep learning solutions, excelling in feature extraction and image classification [2].

This study zeroes in on the vital challenge of weapon detection in images, a matter of paramount importance for public safety and security in today's society. Utilizing CNNs, this research aims to develop a robust and efficient system capable of identifying weapons in various scenarios, thereby enhancing security protocols in both public and private domains [3]. The proposed methodology integrates cutting-edge pre-trained models, such as VGG16, VGG19, ResNet50, and ResNet101, with a custom CNN architecture tailored specifically for weapon recognition [4].

Pre-trained models provide a strong foundation for feature extraction due to their extensive training on diverse datasets [5]. To meet the specific demands of weapon detection, the custom architecture includes additional fully connected layers to boost the system's classification capabilities. Advanced techniques like early stopping and checkpointing are employed to optimize the model training process, preventing overfitting and improving training efficiency [6].

Moreover, this research addresses the inherent imbalance in the dataset, where images containing weapons are significantly outnumbered by non-weapon images. This imbalance poses a risk of biased predictions, potentially leading to false negatives or positives that could undermine the system's reliability [7]. To mitigate this issue, the methodology employs techniques inspired by imbalanced data classification methods, ensuring consistent accuracy across all categories [8]. The dataset is annotated with labels specifying the presence of weapons and their types,

such as handguns, rifles, and knives, creating a comprehensive base for model training and testing [9]. Preprocessing steps, including resizing and normalization, are applied to the images to standardize input quality and enhance the models' ability to generalize across varied conditions [10].

Recent research (2020-2025) has explored various deep learning techniques for weapon detection, significantly advancing the field. Studies utilizing YOLO (You Only Look Once), SSD (Single Shot Multibox Detector), and Faster R-CNN have demonstrated remarkable improvements in real-time weapon recognition [11].

These object detection frameworks leverage region proposal networks and anchor-based predictions to enhance accuracy and computational efficiency [12]. For instance, YOLO's single-stage detection approach has been particularly effective in achieving high-speed processing without compromising precision, making it suitable for real-time surveillance systems [13]. Similarly, SSD's multi-scale feature maps enable the detection of weapons of varying sizes within a single frame, while Faster R-CNN's two-stage detection process ensures higher accuracy by refining region proposals [14].

In addition to these frameworks, hybrid approaches that integrate CNNs with transformer-based architectures, such as Vision Transformers (ViTs), have shown promising results in overcoming challenges like occlusion and background clutter [15]. Transformers, originally designed for natural language processing, have been adapted for computer vision tasks due to their ability to capture long-range dependencies and contextual information [1]. By combining the spatial feature extraction capabilities of CNNs with the global context understanding of transformers, these hybrid models have achieved state-of-the-art performance in weapon detection, particularly in complex environments where weapons may be partially obscured or blended into the background [2].

Attention mechanisms allow the model to focus on the most relevant parts of an

image, improving its ability to detect small or partially hidden weapons [3]. This is particularly useful in crowded or cluttered scenes where weapons may be difficult to distinguish from other objects [4].

For example, models incorporating self-attention layers have demonstrated superior performance in detecting concealed weapons, such as knives hidden under clothing or firearms partially obscured by other objects [5].

Furthermore, the integration of transfer learning techniques has played a crucial role in improving weapon detection systems. Transfer learning involves fine-tuning pre-trained models on specific datasets, allowing them to adapt to the unique characteristics of weapon detection tasks [6]. Utilizing CNNs, this research aims to develop a robust and efficient system capable of identifying weapons in various scenarios, thereby enhancing security protocols in both public and private domains [3]. The proposed methodology integrates cutting-edge pre-trained models, such as VGG16, VGG19, ResNet50, and ResNet101, with a custom CNN architecture tailored specifically for weapon recognition [4].

This approach not only reduces the computational cost of training from scratch but also enhances the model's ability to generalize across different environments and scenarios [7]. For instance, pre-trained models like VGG16, VGG19, ResNet50, and ResNet101 have been fine-tuned on weapon-specific datasets, achieving high accuracy in identifying various types of weapons, including handguns, rifles, and knives [8].Data augmentation techniques have also been widely adopted to address the challenges posed by limited and imbalanced datasets [9].

By applying transformations such as rotation, scaling, flipping, and cropping, researchers can generate diverse training samples, improving the model's robustness and reducing the risk of overfitting [10]. synthetic data generation using techniques like Generative Adversarial Networks (GANs) has been explored to weapon images, further enriching the training dataset and enhancing the model's performance [11].

The evaluation of weapon detection models has also seen significant advancements, with researchers employing a range of metrics to assess their performance [12]. In addition to traditional metrics like accuracy, precision, recall, and F1-score, newer metrics such as mean Average Precision (mAP) and Intersection over Union (IoU) have been widely used to evaluate the localization and classification capabilities of object detection models [13]. These metrics provide a comprehensive understanding of the model's performance, enabling researchers to identify areas for improvement and optimize their architectures [14].

Moreover, the deployment of weapon detection systems in real-world scenarios has highlighted the importance of computational efficiency and scalability [15]. To address these challenges, researchers have explored lightweight architectures and model compression techniques, such as pruning, quantization, and knowledge distillation [1].

These techniques reduce the computational complexity of deep learning models, making them suitable for deployment on resource-constrained devices, such as drones, surveillance cameras, and mobile devices [2]. For example, lightweight versions of YOLO and SSD have been developed to achieve real-time weapon detection on edge devices, enabling their use in large-scale surveillance systems [3].

In conclusion, the field of weapon detection has witnessed remarkable progress in recent years, driven by advancements in deep learning techniques and frameworks [4]. The integration of CNNs with transformer-based architectures, the adoption of attention mechanisms, and the use of transfer learning and data augmentation techniques have significantly improved the accuracy and robustness of weapon detection systems [5]. Furthermore, the development of lightweight models and the application of model compression techniques have enhanced their scalability and computational efficiency, making them suitable for real-world deployment [6].

As research in this area continues to evolve, the potential for deep learning to enhance security protocols and protect public and private spaces remains immense [7]. This study builds on these advancements, proposing a comprehensive methodology for weapon detection that leverages the strengths of existing techniques while addressing their limitations [8]. By combining pre-trained models with custom CNN architectures and employing advanced training techniques, this research aims to contribute to the ongoing development of automated security systems, ultimately improving public safety and security [9].

Looking ahead, the future of weapon detection systems lies in the continuous integration of emerging technologies and interdisciplinary approaches. For instance, the incorporation of edge computing and Internet of Things (IoT) devices can enable real-time, decentralized weapon detection, allowing for faster response times and reduced reliance on centralized systems [1].

Additionally, advancements in explainable AI (XAI) can address the "black box" nature of deep learning models, providing transparency in decision-making processes and fostering trust among end-users, such as law enforcement and security personnel [2]. Collaborative efforts between academia, industry, and government agencies will be crucial in addressing the ethical and societal challenges associated with weapon detection technologies, ensuring that these systems are deployed responsibly and equitably [3].

Furthermore, the development of adaptive models capable of learning from evolving datasets in dynamic environments will be essential to maintaining high accuracy and reliability in the face of new threats and weapon concealment techniques [4]. By leveraging these advancements, weapon detection systems can not only enhance security but also contribute to the creation of safer, more resilient communities in an Complex[5].

2. LITERATURE SURVEY

Recent advancements in deep learning have markedly improved the field of automated image analysis, particularly in object detection. Convolutional Neural Networks (CNNs) have become a cornerstone in this domain, offering significant enhancements in image classification and feature extraction. These networks represent a key shift in how computers can interpret visual data, using hierarchical architectures to automatically extract features such as edges, shapes, and textures from images. By learning patterns and representations, CNNs have revolutionized computer vision, significantly improving accuracy in recognizing objects across varied datasets.

In the realm of weapon detection, CNNs have proven instrumental in designing systems capable of identifying and classifying objects with high precision. Girshick et al. [9] introduced the groundbreaking Region-based CNN (R-CNN) model, which combined CNNs with region suggestions to create the foundation for object detection. The innovation involved segmenting images into potential regions of interest and subsequently passing these regions through a CNN to classify objects contained within them. This approach markedly improved detection accuracy but suffered from computational inefficiencies since each region was processed individually.

To overcome the challenges of R-CNN, Ren et al. [10] introduced Fast R-CNN, which simplified the process and significantly enhanced both speed and accuracy. By incorporating region ideas directly into the CNN pipeline through the Region of Interest (RoI), Fast R-CNN streamlined the detection process. This innovation allowed the entire image to be processed in a single forward pass, significantly reducing redundant computations. Building on this, Faster R-CNN further refined the methodology by introducing a Region Proposal Network (RPN). The RPN dynamically generated region proposals as part of the learning process, enabling Faster R-CNN to achieve near real-time performance without sacrificing accuracy.

The next major leap came with the introduction of the YOLO (You Only Look Once) framework by Redmon et al. [11]. This single-stage detector deviated from traditional two-stage approaches by framing object detection as a regression problem. YOLO processed the entire image in one go, generating bounding boxes and predicting class probabilities simultaneously. This architecture allowed YOLO to perform real-time detection with impressive precision, making it a game-changer for applications requiring instantaneous decisions. YOLO's ability to balance speed with accuracy made it particularly well-suited for weapon detection in security-critical environments, such as surveillance systems.

In addition to these foundational advancements, researchers like Yao et al. have developed specialized CNN architectures specifically designed for weapon detection tasks. Their work involved tailoring CNNs to address challenges such as distinguishing between visually similar objects, including different types of firearms or non-threatening objects that resemble weapons. These architectures incorporated additional convolutional layers and employed specialized training techniques to refine the model’s ability to differentiate subtle differences. The complexity of firearm detection lies not only in identifying the object but also in isolating it from cluttered backgrounds and variable lighting conditions, challenges that specialized models effectively address.

The introduction of ResNet50 by He et al. [12] was another significant milestone in CNN evolution. ResNet50 tackled the vanishing gradient problem, a common issue in deep networks, by introducing residual connections. These connections bypass certain layers, ensuring that gradients can flow freely and enabling the training of substantially deeper networks. ResNet50 set new benchmarks in image classification and object detection, offering improved performance in detecting weapons in complex scenes where overlapping objects or diverse visual features are present.

Furthermore, the application of transfer learning, as highlighted by Krizhevsky et al. [13], has been transformative in the field of weapon detection. Transfer

learning involves fine-tuning pre-trained models, which have been trained on large datasets like ImageNet, for specific tasks such as firearm identification. By leveraging pre-trained architectures, transfer learning significantly reduces the computational resources required and minimizes the dependence on large labeled datasets. This approach also enhances generalization, as pre-trained models capture features that are universally relevant across a broad spectrum of images.

Another critical aspect of CNN-based weapon detection is the use of robust preprocessing techniques to ensure consistent data quality. Preprocessing begins with resizing all input images to a uniform size, such as 224x224 pixels, ensuring compatibility with pre-trained networks like ResNet50 and VGG16. Normalization is applied to scale pixel values to a range between 0 and 1, accelerating the convergence of learning algorithms. Additionally, data augmentation techniques, including flipping, rotation, and cropping, are employed to artificially increase dataset diversity, thereby improving the model's robustness to variations in lighting, orientation, and occlusion.

CNN-based weapon detection systems also benefit from hierarchical feature extraction capabilities. Convolutional layers capture low-level features such as edges, while deeper layers identify complex structures such as weapon contours. Pooling layers reduce spatial dimensions, enhancing computational efficiency, and fully connected layers consolidate the extracted features for classification. Advanced activation functions like Rectified Linear Unit (ReLU) introduce non-linearity, enabling the network to learn more complex decision boundaries. The output layer, equipped with a softmax function, predicts the probability distribution across predefined classes, such as weapon types.

Throughout the training phase, optimization techniques play a pivotal role in fine-tuning the model for superior performance. Loss functions like categorical cross-entropy measure the disparity between predicted and actual labels, guiding the optimization process. The Adam optimizer, known for its adaptive learning rate, ensures efficient weight updates, accelerating convergence. To prevent overfitting,

strategies like early stopping and dropout regularization are implemented. Early stopping halts training once validation performance plateaus, while dropout randomly deactivates neurons during training, enhancing the network's ability to generalize to unseen data.

The methodologies discussed exemplify the tremendous advancements in CNN-based weapon detection systems. From the pioneering R-CNN to the cutting-edge YOLO framework and specialized architectures, each innovation has addressed specific challenges in object detection. The integration of residual connections in ResNet50 and the versatility of transfer learning have further elevated the field, enabling researchers to develop systems that are both accurate and computationally efficient.

Santos et al. [14] proposed that the number of crimes involving weapons had grown significantly on a global scale, particularly in regions where enforcement was weak or where possessing weapons was legal. They emphasized the urgency of combating such criminal activities by identifying early signs of criminal behavior and enabling law enforcement agencies to take swift action. The paper highlighted an important limitation of human surveillance: despite the evolved and efficient human visual system, prolonged periods of monitoring similar footage can lead to inattention and slow responses. Moreover, large-scale surveillance systems often require numerous personnel for monitoring, adding to operational costs. Automatic weapon detection systems based on computer vision were seen as a feasible solution, but their performance remained limited in challenging contexts. However, Santos et al. pointed out that combining synthetic data with realistic imagery significantly improved detection performance, offering a promising avenue for addressing these challenges.

Mukto et al. [15] turned their focus to a broader framework for monitoring crimes through the Crime Monitoring System (CMS). This real-time system leveraged camera surveillance to detect criminal activities, counterbalancing human weaknesses such as slow reactions or lapses in attention. Mukto et al.'s CMS

divided the detection pipeline into three key stages: weapon detection, violence detection, and face recognition. These stages utilized various deep learning and image-processing algorithms in synergy to achieve their objectives. Notably, the system combined the power of closed-circuit television (CCTV) with sophisticated computational models, allowing for the automated detection of weapon-related crime scenes, violent activities, and the identification of individuals through face recognition. This comprehensive potential of integrating multiple technologies to enhance public safety and reduce human reliance in critical tasks.

Manikandan et al. [16] proposed an innovative solution to tackle abnormal activities in crowded environments, a challenge that traditional surveillance methods struggled to address effectively. Despite the presence of security systems, most surveillance solutions were unable to predict or intimate administrators about abnormal events in real time. Manikandan et al. designed a novel framework using deep learning. Surveillance footage was segmented into frames, with each frame being classified as either normal or abnormal. Abnormal frames triggered immediate alerts to security personnel. This deep learning-based approach represented a marked improvement over previous systems, offering real-time anomaly detection with greater scalability and adaptability across different environments.

Gu et al. [17] introduced a complete robotic system capable of performing cooperative grasping tasks, independent of object types. Their research primarily focused on a grasp detection model that leveraged convolutional neural networks (CNNs) and was evaluated on two benchmark datasets: the Cornell Grasp Dataset and the Jacquard Dataset. Impressively, the system achieved an accuracy of 97.8% and 96.6% on image-wise and object-wise splitting tests, respectively, on the Cornell dataset, and 93.9% accuracy on the Jacquard dataset. Though unrelated directly to weapon detection, Gu et al.'s work demonstrated the versatility of CNN-based frameworks, emphasizing their applicability in solving various object detection and interaction challenges.

Tian et al. [18] shifted attention to understanding the interpretability of deep neural networks. Traditional deep learning algorithms, while effective, often operate as black boxes, providing limited transparency into their reasoning processes. Tian et al. proposed a multi-level hierarchical deep learning framework designed to explain hidden information within neural networks. This algorithm encompassed multiple hierarchical architectures and training mechanisms, offering improved interpretability without compromising accuracy. Experimental results validated the model's capability to address traditional limitations, suggesting broader applications in domains requiring high levels of explainability, including weapon detection systems.

Mwaffo et al. [19] extended the use of deep neural networks (DNNs) to autonomous systems, with a specific application in aerial refueling for the US Navy. Their work aimed to replace traditional manned refueling systems with autonomous, uncrewed platforms. The DNN model developed by Mwaffo et al. not only detected refueling drogues but also used instance segmentation to track them dynamically under various conditions. Trained on limited real flight test data, this advanced model achieved precise detection, further demonstrating the robustness of DNNs when applied to real-world scenarios. Although focused on aviation, the techniques outlined in this study had implications for weapon recognition through enhanced object segmentation capabilities.

Kuma et al. [20] addressed violent attacks and security concerns in smart cities, proposing an architecture that combined stable diffusion models with violence detection and pose estimation techniques. This innovative framework utilized synthetic data generation to enhance training, while YOLO v7—a widely-used object detection algorithm—identified violent objects such as knives, baseball bats, and pistols. By integrating MediaPipe for action detection, the system achieved high real-time performance, sending emergency alerts during violent incidents. Kuma et al.'s comprehensive approach demonstrated the effectiveness of

combining data augmentation, deep learning, and real-time processing to mitigate risks in urban environments.

Xu et al. [21] proposed a convolutional neural network specifically designed for detecting key points in swimming posture analysis. This application had relevance for posture evaluation in athletic training. However, external challenges such as lighting variability, image quality degradation, and occlusion often hampered detection stability. Xu et al. addressed these issues by employing an optical image enhancement technique alongside their CNN model. By preprocessing images to improve clarity and enhance feature extraction, their methodology achieved notable improvements in detection accuracy. This study underscored the importance of preprocessing techniques for achieving superior performance in complex visual tasks.

Liu et al. [22] contributed to radar signal recognition by proposing a lightweight convolutional model named CV-LPINet. Addressing challenges such as large network sizes and computational complexity in traditional methods, their system optimized both performance and resource usage. CV-LPINet incorporated residual structures and separable convolutional modules, achieving a high average recognition accuracy of 91.76% within a signal-to-noise ratio range of -6 to 10 dB. Although the primary focus was signal processing, Liu et al.'s emphasis on lightweight and efficient architectures could be adapted for real-time weapon detection in resource-constrained settings.

Ogundunmade et al. [23] studied terrorist attacks in Nigeria, employing Bayesian Neural Networks (BNNs) to predict the nature of such activities. By analysing various activation functions, concluded that the hyperbolic tangent function performed best in extracting predictive variables. The research's focus on understanding terrorist behaviours paralleled efforts in weapon detection to prevent harm proactively by employing and real-time solutions.

Kumar et al. [24] explored human activity recognition (HAR) in diverse domains, such as healthcare, security, and education. Leveraging advanced deep learning techniques, their comprehensive analysis offered insights into designing scalable models for real-time applications. Although primarily focused on activity recognition, Kumar et al.'s emphasis on downstream tasks and video processing had direct implications for enhancing weapon detection systems.

Sirimewan et al. [25] examined the application of deep learning for automated waste recognition. Their study demonstrated the effectiveness of transfer learning and feature extraction for identifying construction and demolition (CRD) waste in real-world settings. By employing pre-trained networks, the authors achieved high accuracy rates, emphasizing the potential of such models in other complex classification tasks, including identifying subtle weapon characteristics in diverse environments.

3. SYSTEM ANALYSIS

3.1 Existing System

The Region-based CNN (R-CNN) was introduced as a groundbreaking model that combined CNNs with region proposals to detect objects in images. This system segmented images into regions of interest and processed each region through a CNN for classification, significantly improving detection accuracy. It laid the foundation for modern object detection systems [9].

The Fast R-CNN model streamlined the detection process by integrating Region of Interest (RoI) pooling into the CNN pipeline. This allowed the entire image to be processed in a single forward pass, reducing computational redundancy and improving both speed and accuracy compared to R-CNN [10].

The YOLO (You Only Look Once) framework revolutionized object detection by treating it as a regression problem. YOLO processed the entire image in one go, generating bounding boxes and class probabilities simultaneously, enabling real-time detection with high precision [11].

Specialized CNN architectures were designed for weapon detection, incorporating additional convolutional layers and training techniques to distinguish between visually similar objects, such as different types of firearms or non-threatening objects resembling weapons [Yao et al.].

The ResNet50 model addressed the vanishing gradient problem through residual connections, enabling the training of much deeper networks. This innovation improved performance in complex object detection tasks, including weapon detection in cluttered scenes [12].

Transfer learning was highlighted as a transformative approach, where pre-trained models on large datasets like ImageNet were fine-tuned for specific tasks such as firearm identification. This reduced computational costs and improved generalization by leveraging universal features learned from large datasets [13].

A system combining synthetic data with realistic imagery was proposed to enhance weapon detection performance in surveillance footage. This approach addressed the limitations of human surveillance and improved detection accuracy in challenging contexts [14].

A Crime Monitoring System (CMS) was developed, integrating weapon detection, violence detection, and face recognition using deep learning algorithms. The system leveraged CCTV footage and computational models to automate the detection of criminal activities in real time [15].

A deep learning-based framework was created for real-time anomaly detection in crowded environments. The system segmented surveillance footage into frames and classified them as normal or abnormal, triggering alerts for abnormal activities [16].

A CNN-based cooperative grasp detection system was designed for robotic applications. The system achieved high accuracy on benchmark datasets, demonstrating the versatility of CNNs in object detection and interaction tasks [17].

A multi-level hierarchical deep learning framework was proposed to improve the interpretability of neural networks. The model provided insights into the reasoning process of deep networks without compromising accuracy, making it suitable for applications requiring explainability [18].

A deep neural network (DNN) model was applied to autonomous aerial refueling, using instance segmentation to detect and track refueling drogues under various conditions. The system demonstrated the robustness of DNNs in real-world scenarios [19].

A framework combining stable diffusion models with violence detection and pose estimation techniques was proposed to enhance smart city safety. The system used YOLO v7 for object detection and MediaPipe for action recognition, achieving high real-time performance [20].

A CNN-based system was developed for key point detection in swimming posture analysis. The system employed optical image enhancement techniques to improve detection accuracy under challenging conditions like lighting variability and occlusion [21].

A lightweight CNN model (CV-LPINet) was proposed for radar signal recognition. The model incorporated residual structures and separable convolutional modules, achieving high accuracy while optimizing computational efficiency [22].

A Bayesian Neural Network (BNN) model was used to predict the nature of terrorist attacks in Nigeria. The model analyzed various activation functions and identified the hyperbolic tangent function as the most effective for extracting predictive variables [23].

A deep learning-based human activity recognition (HAR) system was explored, providing insights into designing scalable models for real-time applications. The work had implications for enhancing weapon detection systems [24].

A deep learning model was applied for automated waste recognition, leveraging transfer learning and feature extraction to achieve high accuracy in identifying construction and demolition waste in diverse environments [25].

3.2 DISADVANTAGE OF EXISITING SYSTEM

The R-CNN model suffered from computational inefficiencies, as each region proposal was processed individually through the CNN. This made the system slow and resource-intensive, limiting its applicability in real-time scenarios. A research gap exists in optimizing the region proposal process to reduce computational overhead [9].

Although Fast R-CNN improved speed and accuracy, it still relied on external region proposal algorithms, which added complexity and computational overhead to the detection pipeline. Future work could focus on integrating region proposal generation directly into the CNN architecture [10].

YOLO, while fast, struggled with detecting small objects and objects in close proximity due to its grid-based approach. This limitation affected its performance in cluttered or complex scenes. Research is needed to improve small object detection and spatial resolution in YOLO-based systems [11].

The specialized CNN architectures for weapon detection required extensive training data and computational resources. Additionally, distinguishing between visually similar objects remained challenging, especially in low-resolution or noisy images. Future research could focus on improving feature extraction for subtle differences and reducing data dependency [Yao et al.].

ResNet50, despite its depth and performance, was computationally expensive to train and deploy, making it less suitable for resource-constrained environments. Research gaps include developing lightweight variants of ResNet50 for real-time applications [12].

Transfer learning, while effective, depended heavily on the quality and relevance of the pre-trained models. Fine-tuning for specific tasks required careful hyperparameter tuning and domain-specific data. Future work could explore automated transfer learning techniques to reduce manual intervention [13].

The combination of synthetic and realistic data improved performance but introduced challenges in generating high-quality synthetic data that accurately represented real-world scenarios. Research gaps include improving the realism and diversity of synthetic datasets [14].

The Crime Monitoring System (CMS) faced challenges in integrating multiple detection modules (weapon detection, violence detection, and face recognition), which increased system complexity and computational demands. Future research could focus on modular and scalable architectures for multi-task systems [15].

The deep learning-based anomaly detection system required significant computational resources for training and inference, limiting its scalability in large-scale surveillance networks. Research gaps include developing energy-efficient models for real-time anomaly detection [16].

The cooperative grasp detection system, while accurate, was limited to specific datasets and required extensive retraining for new object types or environments. Future work could focus on improving generalization across diverse datasets [17].

The multi-level hierarchical framework improved interpretability but added complexity to the model architecture, making it harder to implement and optimize. Research gaps include simplifying hierarchical models without sacrificing interpretability [18].

The DNN-based aerial refueling system relied on limited real flight test data, which could affect its generalization to diverse operational conditions. Future research could focus on data augmentation and simulation-based training to improve robustness [19].

The violence detection system, while effective, faced challenges in accurately detecting subtle or context-dependent violent actions, leading to potential false positives. Research gaps include improving context-awareness and reducing false positives in violence detection [20].

The CNN-based key point detection system was sensitive to variations in lighting and image quality, requiring extensive preprocessing to maintain accuracy. Future work could focus on developing robust models that are less dependent on preprocessing [21].

CV-LPINet, while lightweight, faced challenges in generalizing to unseen radar signals, especially in low signal-to-noise ratio conditions. Research gaps include improving generalization and robustness in noisy environments [22].

The Bayesian Neural Network (BNN) model required extensive computational resources for training and inference, limiting its scalability. Future research could focus on developing efficient Bayesian methods for real-time applications [23].

The human activity recognition (HAR) system faced challenges in handling diverse and complex activities in real-world scenarios. Research gaps include improving the adaptability of HAR systems to dynamic environments [24].

The deep learning model for waste recognition faced challenges in accurately identifying waste in diverse environmental settings. Future research could focus on improving feature extraction and classification for complex waste types [25].

3.3 PROPOSED SYSTEM

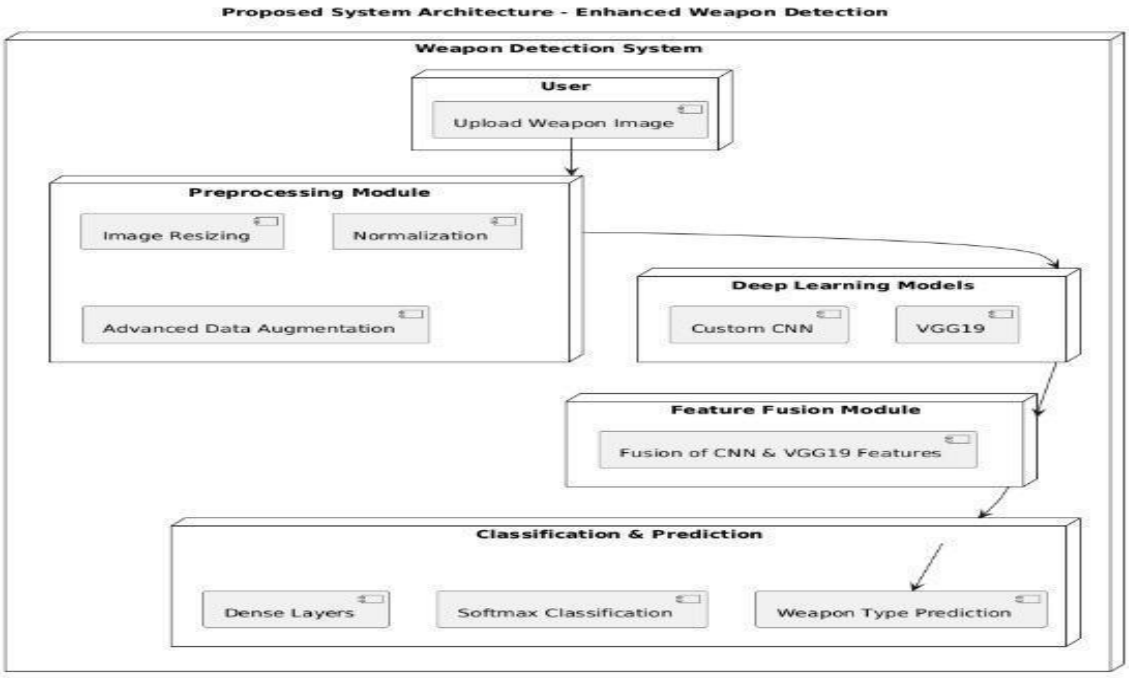

Fig 3.1**:** Block Diagram of Proposed system

Step 1: Weapon Detection Dataset

The first step in the weapon detection system involves collecting or acquiring a dataset containing images of weapons and non-weapons. This dataset must be diverse, including various types of weapons (e.g., knives, guns, etc.) captured from different angles, lighting conditions, and environments to train the model effectively. Public datasets or custom-curated datasets can be used, and the images are usually labeled based on the presence or absence of weapons, forming the foundation for the subsequent model training and evaluation.

Step 2: Pretrained Models (ResNet50, ResNet101, VGG16)

Pretrained models such as ResNet50, ResNet101, and VGG16 are explored as baseline algorithms. These architectures, trained on large-scale datasets like ImageNet, are fine-tuned on the weapon detection dataset. ResNet models use residual learning with skip connections, allowing deeper networks to avoid degradation problems, while VGG16 relies on deep, sequential layers with small filters to capture intricate image features.

These models offer strong feature extraction capabilities and are leveraged for comparison with custom approaches.

Step 3: Proposed Algorithm (VGG19 and Custom CNN)

The proposed approach builds on the strengths of VGG19, which is a deeper version of VGG16 with 19 layers, providing better feature extraction through its additional layers. Alongside VGG19, a custom CNN is designed with specific architecture tailored for weapon detection, involving a combination of convolutional, pooling, and fully connected layers, optimized for the dataset at hand. This custom architecture is typically more lightweight and can be fine-tuned for better performance in weapon detection tasks, especially where computational efficiency and specific dataset characteristics are considered.

Step 4: Performance Comparison

After training the existing algorithms (ResNet50, ResNet101, VGG16) and the proposed models (VGG19 and Custom CNN), their performance is compared based on several metrics such as accuracy, precision, recall, and F1-score. This step involves evaluating how well each model identifies weapons versus non-weapons from the test dataset. Performance evaluation may also consider computational aspects like training time, inference speed, and resource utilization, providing insights into which model strikes the best balance between accuracy and efficiency.

Step 5: Prediction of Output from Test Data (VGG19 and Custom CNN)

Finally, the trained VGG19 and custom CNN models are used to predict the presence or absence of weapons in the test dataset. The predictions are based on the learned patterns and features extracted from the training phase. The output for each test image is a class label (e.g., "weapon" or "non-weapon"), along with confidence scores. These models are expected to generalize well to unseen data, providing reliable predictions that can be used in real-world weapon detection systems. The test data serves as the final validation of the models' effectiveness in detecting weapons in diverse scenarios.

3.4 FEASIBILITY STUDY

The feasibility study is a critical component of any project, providing a structured assessment of various factors that might impact its successful implementation. In the context of developing an advanced weapon detection system utilizing deep learning techniques, this feasibility study will evaluate several key aspects: technical, operational, economic, legal, and schedule feasibility.

1. Technical Feasibility

Technological Infrastructure: The proposed system will be built on advanced technologies, specifically deep learning frameworks such as TensorFlow, Keras, or PyTorch. These platforms support the development and training of Convolutional Neural Networks (CNNs) necessary for image classification and object detection tasks.

Data Acquisition and Processing: A robust dataset is essential for training the model effectively. The feasibility study must consider the availability of diverse image datasets that include annotated examples of various weapons. Techniques for image preprocessing (resizing, normalization, and augmentation) must also be achievable within the chosen technology stack.

Skill Requirements: The development and implementation of the weapon detection system will require a team equipped with skills in machine learning, image processing, and software development. Ensuring that team members have the requisite knowledge of CNNs and experience with the chosen frameworks will be crucial.

2. Operational Feasibility

Integration with Existing Systems: The weapon detection system must be designed to integrate seamlessly with current security infrastructures, such as CCTV surveillance systems. Operational feasibility will assess the ease with which this integration can be accomplished and the necessary adjustments to existing workflows.

User Acceptance: The acceptance of the new system by security personnel and end-users will significantly impact. Training programs should be developed to educate users on system.

3. Economic Feasibility

Cost Analysis: A comprehensive financial assessment will investigate the various costs associated with system development, including hardware, software licenses, data acquisition, and labor costs. It will also evaluate potential revenue streams, such as subscriptions for real-time monitoring services or partnerships with security firms.

Return on Investment (ROI): The potential economic impact of implementing an automated weapon detection system should be analyzed. A cost-benefit analysis will help determine whether the financial investment is justified by the anticipated savings through increased efficiency and reduced incident rates.

4. Legal Feasibility

Compliance with Local Regulations: The system must adhere to various regulations related to surveillance and data privacy. Legal feasibility will assess the implications of data collection, storage, and processing under laws like GDPR or local privacy laws.

Liability Issues: It's crucial to evaluate the legal liabilities associated with false negatives or positives generated by the weapon detection system. Understanding the legal implications of these outcomes will inform system designs that prioritize safety and accountability.

Intellectual Property (IP) Concerns: The feasibility study should explore potential IP issues surrounding the use of datasets, algorithms, or technologies. Ensuring that all components of the system avoid infringement upon existing patents is necessary for legal security.

4. SYSTEM REQUIREMENTS

System requirements define the hardware and software specifications necessary for the successful implementation and execution of the music genre classification system. These requirements ensure that the system can handle large datasets, perform real-time classification, and efficiently train deep learning models. The following sections outline the hardware and software prerequisites needed for optimal performance.

4.1 Hardware Requirements

The hardware requirements specify the physical computing resources needed to execute the Convolutional Neural Network (CNN)-based music genre classification model. The system processes large amounts of audio data in the form of spectrograms and extracted features, requiring sufficient computational power to handle deep learning tasks effectively. The minimum and recommended hardware specifications are listed below.

- System Type : intel®core™i7- i7-13620HCPU@2.4ghz
- GPU Type: : Nvidia GeForce RTX 4060
- RAM : 16 gigabyte(GB)
- Bus Speed : 5200 MT/s
- Number of cores 10

Graphics Processing Unit (GPU): Not required for basic execution but recommended for faster training

A dedicated GPU is highly recommended for deep learning tasks to accelerate model training and inference speed. If using cloud-based services like Google Colab, AWS, or Azure, high-end local hardware may not be required.

4.2 Software Requirements

Software requirements include all necessary tools, libraries, and frameworks needed to develop, train, and deploy the music genre classification system. Since this project utilizes Python-based deep learning frameworks, certain software components must be installed for proper functionality.

- Operating System : Windows 10, 64 bit Operating System
- Coding Language : Python
- Frontend : HTML, CSS & JS
- Backend : Django
- IDE : VsCode
- Python distribution : Anaconda Navigator & Google Collab

The hardware and software requirements listed below ensure that the weapon detection system functions efficiently. The system requires high-performance computing resources to handle deep learning-based weapon classification, especially when training on large datasets. Ensuring that the right software dependencies and frameworks are installed will lead to better model accuracy, improved training efficiency, and seamless real-time classification performance. Additionally, optimized hardware and software configurations contribute to lower latency and enhanced model scalability, making the system suitable for large-scale security applications.

5. SYSTEM DESGIN

5.1 SYSTEM ARCHITECTURE:

1. IMAGE PREPROCESSING

In weapon recognition from images using deep neural networks, image preprocessing plays a crucial role in enhancing the performance and efficiency of the model.Several key steps typically involved in image preprocessing:

Resizing and Standardization: Images are resized to a uniform size to ensure consistency across the dataset. This helps in reducing computational complexity and ensures that the model receives inputs of consistent dimensions. Standardization involves normalizing pixel values to a common scale (e.g., [0, 1] or [-1, 1]) to improve convergence during training.

Data Augmentation: Data augmentation techniques such as rotation, flipping, cropping, and brightness adjustment are applied to increase the diversity of the dataset. This helps in improving the model's generalization capability and robustness to variations in lighting, orientation, and background clutter.

Noise Reduction: Noise reduction techniques such as Gaussian blurring or median filtering may be employed to smooth out pixel-level variations caused by sensor noise or compression artifacts. This helps in enhancing the clarity of the images and improving the model's ability to extract relevant features

Feature Extraction: Feature extraction techniques, such as edge detection or histogram equalization, may be applied to highlight discriminative features in the images. This helps in emphasizing relevant details related to weapons while suppressing irrelevant background information, leading to more effective feature representation for the neural network.

Normalization and Centering: Normalization techniques are applied to adjust the overall intensity distribution of the images, making them more suitable for processing by the neural network. Centering involves shifting pixel values to have a mean of zero, which helps in reducing bias and improving convergence during training.

Region of Interest (ROI) Extraction: If the images contain multiple objects or regions, techniques such as object detection or segmentation may be used to identify

and extract the regions of interest (e.g., potential weapons) for further analysis. This helps in focusing the model's attention on relevant areas, reducing overhead and improving accuracy.

Histogram Equalization: Histogram equalization is employed to enhance the contrast of the images, making it easier for the neural network to distinguish between different objects and background elements. This can help in improving the model's ability to detect subtle features associated with weapons.

Pretrained Model Initialization: Transfer learning techniques may be utilized by initializing the neural network with weights pretrained on a large-scale image dataset (e.g., ImageNet). This allows the model to leverage knowledge learned from generic visual features before fine-tuning on the weapon recognition task-specific dataset, leading to faster convergence and improved performance.

Overall, image preprocessing techniques are essential for preparing the dataset and optimizing the input data for effective training and inference by deep neural networks in weapon recognition tasks. These steps help in improving the model's robustness, accuracy, and efficiency in detecting weapons from images.

2. CNN LAYERS DESCRIPTION

In a Convolutional Neural Network (CNN) designed for weapon recognition from images, the typical architecture consists of several layers, each serving a specific purpose in extracting and processing features from the input images. Here's a description of the commonly used CNN layers in such a scenario:

Input Layer: This layer receives the input images, typically in the form of pixel intensities arranged in a 2D or 3D array (height, width, channels). The dimensions of the input layer correspond to the size and color depth of the input images.

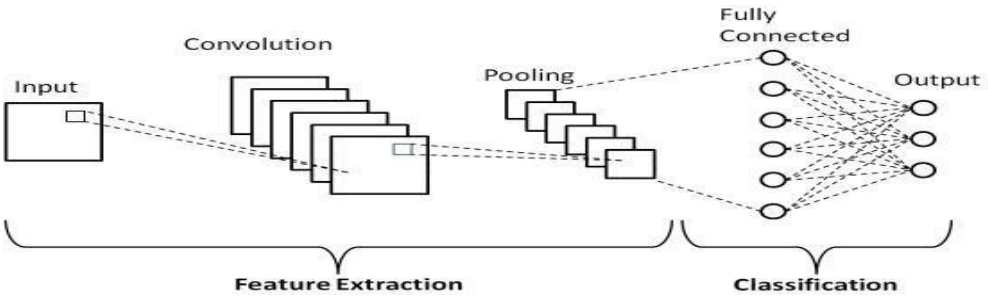

Fig 5.1.1: CNN layers description

Convolutional Layers: Convolutional layers perform feature extraction by applying a set of learnable filters (kernels) to the input images. Each filter convolves across theinput images, extracting spatial features such as edges, textures, and patterns. The output of each convolutional layer consists of feature maps, which represent the presence of specific features at different spatial locations. Multiple convolutional layers with increasing filter sizes and depths are typically used to capture hierarchical and abstract features.

Adam optimizer: Adam optimizer, short for "Adaptive Moment Estimation," is an iterative optimization algorithm used to minimize the loss function during the training of neural networks. Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum.

Activation Functions: Activation functions (e.g., ReLU, Leaky ReLU, or Sigmoid) are applied element-wise to the output of convolutional layers to introduce non-linearity into the network. Non-linear activation functions enable the CNN to learn complex mappings between input images and output classes, improving the model's representational capacity.

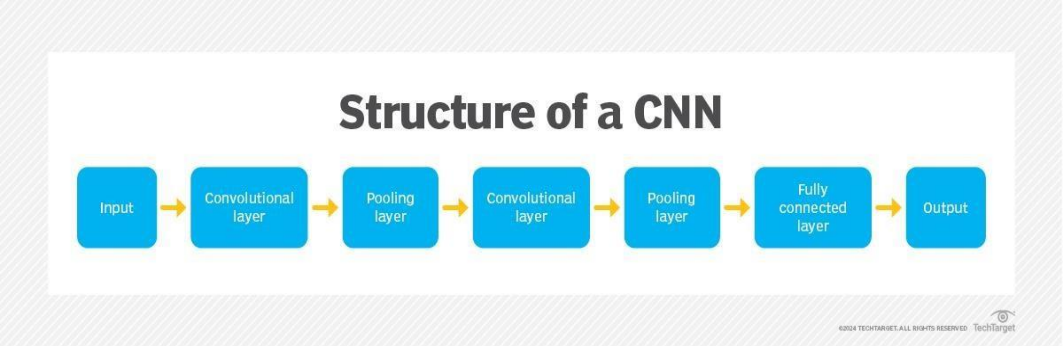

Fig 5.1.2: Structure of a CNN

Pooling (Downsampling) Layers: Pooling layers reduce the spatial dimensions of feature maps while preserving the most relevant information. Common pooling operations include max pooling and average pooling, which down sample feature maps by selecting the maximum or average value within each pooling region. Pooling layers help in reducing the computational complexity of the network and improving its translational invariance.

Batch Normalization: Batch normalization layers normalize the activations of the previous layer during training. stabilizing and accelerating the training process by reducing internal covariate shift and improving the convergence of the network.

Fully Connected (Dense) Layers: Fully connected layers process the flattened feature maps from the last convolutional or pooling layer. These layers perform high-level feature representation and mapping to the output classes through a series of weighted connections. Typically, one or more fully connected layers are followed by an output layer with softmax activation for multi-class classification, predicting the probabilities of different classes (e.g., weapon or non-weapon).

Dropout: Dropout layers may be added to prevent overfitting by randomly dropping a fraction of neuron activations during training. Dropout regularization helps in improving the generalization performance of the network by reducing co-adaptation among neurons.

Output Layer: The output layer produces the final predictions of the CNN model. For weapon recognition, the output layer typically consists of one or more neurons with softmax activation for multi-class classification, where each neuron represents a class label (e.g., weapon or non-weapon), and the softmax function outputs the probability distribution over these classes.

By combining these layers in a well-designed architecture and training the CNN on a labeled dataset of weapon and non-weapon images, the model can learn to automatically detect and classify weapons in new unseen images.

Advantages

Convolutional Neural Networks (CNNs) offer several advantages, particularly in the field of image recognition and computer vision.

1. Feature Learning:

CNNs automatically learn hierarchical representations and features from the input data. Through convolutional and pooling layers, they can identify patterns, textures, and complex features within images.

2. Spatial Hierarchies:

CNNs capture spatial hierarchies of features. Lower layers focus on simple features like edges and textures, while higher layers combine these features to recognize more complex structures and objects.

3. Parameter Sharing:

Convolutional layers in CNNs use parameter sharing same filters are applied to different parts of the input image. This reduces the number of parameters compared to fully connected networks, making CNNs computationally more efficient.

4. Translation Invariance:

Convolutional operations provide a degree of translation invariance, allowing CNNs to recognize patterns and features regardless of their location in the input image. This property is crucial for tasks like object recognition.

5. Pooling Layers:

Pooling layers help reduce the spatial dimensions of the input data, decreasing computational complexity and focusing on the most relevant information. Max pooling, for example, retains the most significant values from a group of neighboring pixels.

6. Robust to Variations:

CNNs are robust to variations in scale, orientation, and position of objects in an image. This makes them suitable for tasks like object detection and image classification across diverse scenarios.

7. Transfer Learning:

CNNs are often used in transfer learning, where pre-trained models on large datasets (e.g., ImageNet) can be fine-tuned for specific tasks with smaller datasets. This leverages the learned features from the large dataset.

8. Effective for Image Classification:

CNNs have demonstrated state-of-the-art performance in various image classification tasks, including image recognition, object detection, and image segmentation. Their ability to capture hierarchical features contributes to their success.

9. Memory Efficiency:

Parameter sharing and weight sharing in convolutional layers contribute to the efficient use of memory. This is particularly important when dealing with large datasets and complex models.

1. ResNet 50 Model

ResNet50 is a deep convolutional neural network architecture that is 50 layers deep. It was introduced in 2015 by Kaiming He and colleagues in their paper "Deep Residual Learning for Image Recognition." The model is part of the Residual Network (ResNet) family, which was designed to enable the training of extremely deep neural networks by introducing residual learning frameworks. ResNet50 has become a foundational architecture in computer vision tasks due to its balance between depth and computational efficiency.

How It Works

ResNet50 addresses the problem of vanishing gradients, which occurs when training very deep networks. The key innovation is the introduction of residual blocks with shortcut connections that allow the network to learn residual functions with reference to the layer inputs, instead of learning unreferenced functions. This is achieved by adding the input of a layer to the output of a deeper layer, effectively creating a shortcut or skip connection.

By learning these residuals, the network can efficiently train deeper architectures without suffering from degradation problems where adding more layers leads to higher training error. The residual connections facilitate gradient flow during backpropagation, ensuring that gradients can reach earlier layers in the network, which helps in maintaining performance as the network depth increases.

Architecture

- **Initial Layers:**
    - **Conv1:** 7x7 convolution with 64 filters, stride 2, followed by batch normalization and ReLU activation.
    - **Max Pooling:** 3x3 max pooling with stride 2.
- **Residual Blocks:**
    - The network consists of four stages, each containing multiple residual blocks.
    - **Stage 1 (Conv2_x):** Contains 3 residual blocks.
    - **Stage 2 (Conv3_x):** Contains 4 residual blocks.
    - **Stage 3 (Conv4_x):** Contains 6 residual blocks.
    - **Stage 4 (Conv5_x):** Contains 3 residual blocks.
    - Each residual block uses a bottleneck design with three layers:
        - **1x1 Convolution:** Reduces dimensionality.
        - **3x3 Convolution:** Processes spatial features.
        - **1x1 Convolution:** Restores dimensionality.

- **Shortcut Connections:** Identity or projection shortcuts connect the input of a block to its output, facilitating residual learning.

- **Final Layers:**
  - **Global Average Pooling:** Reduces each feature map to a single value.
  - **Fully Connected Layer:** Outputs class probabilities through a softmax activation.

2. ResNet101 Model

ResNet101 is an extension of the ResNet architecture, increasing the depth to 101 layers. Introduced in the same 2015 paper by Kaiming He et al., ResNet101 was designed to explore the effects of increased depth on model performance. The model aims to capture more complex features by stacking more residual blocks, allowing it to potentially achieve higher accuracy on challenging tasks.

How It Works

ResNet101 operates on the same residual learning principle as ResNet50 but extends the number of residual blocks, especially in the deeper layers. The additional layers allow the network to learn more intricate patterns and abstractions from the data. Like ResNet50, it uses identity and projection shortcuts to facilitate the flow of gradients, enabling efficient training of deeper networks without degradation.

Architecture

- **Initial Layers:**
  - Similar to ResNet50, starting with a 7x7 convolutional layer with 64 filters, stride 2, followed by batch normalization, ReLU activation, and a 3x3 max pooling layer with stride 2.

- **Residual Blocks:**
  - **Stage 1 (Conv2_x):** Contains 3 residual blocks.
  - **Stage 2 (Conv3_x):** Contains 4 residual blocks.
  - **Stage 3 (Conv4_x):** Contains **23** residual blocks (significantly more than ResNet50's 6 blocks in this stage).
  - **Stage 4 (Conv5_x):** Contains 3 residual blocks.

- Each residual block uses the same bottleneck design as in ResNet50, with 1x1, 3x3, and 1x1 convolutions.
- The increased number of blocks, particularly in Conv4_x, accounts for the deeper architecture.

- **Final Layers:**
  - **Global Average Pooling:** Aggregates feature maps.
  - **Fully Connected Layer:** Outputs class probabilities via softmax activation.
- **Complexity in Hyperparameter Tuning:** Deeper networks may require more careful tuning of learning rates and regularization parameters to achieve optimal performance.

3. VGG16 Model

VGG16 is a convolutional neural network model developed by the Visual Geometry Group (VGG) at the University of Oxford and presented in the 2014 paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" by Simonyan and Zisserman. It was notable for its simplicity and uniform architecture, which used small convolutional filters (3x3) throughout the network. VGG16 was a significant model in the ILSVRC-2014 competition, securing high rankings in image classification and localization tasks.

How It Works

VGG16 emphasizes depth and simplicity by stacking multiple convolutional layers with small receptive fields. The core idea is that multiple 3x3 convolutional layers stacked together can emulate the effect of larger receptive fields (e.g., 5x5 or 7x7) while keeping the number of parameters manageable. This approach allows the network to learn more complex features and patterns through increased depth.

Architecture

- **Convolutional Layers:**
  - The network consists of 13 convolutional layers.
  - Uses only 3x3 convolution filters with stride 1 and padding to preserve spatial dimensions.

- The layers are organized into blocks, with increasing depth:
    - **Block 1:** Two convolutional layers with 64 filters.
    - **Block 2:** Two convolutional layers with 128 filters.
    - **Block 3:** Three convolutional layers with 256 filters.
    - **Block 4:** Three convolutional layers with 512 filters.
    - **Block 5:** Three convolutional layers with 512 filters.

- **Activation Function:**
    - ReLU (Rectified Linear Unit) is applied after each convolutional layer to introduce non-linearity.

- **Pooling Layers:**
    - Five max-pooling layers are interleaved between the convolutional blocks.
    - Max-pooling is performed over a 2x2 pixel window with stride 2, reducing spatial dimensions.

- **Fully Connected Layers:**
    - Following the convolutional and pooling layers, the network has three fully connected layers:
        - Two layers with 4096 nodes each.
        - A final layer with 1000 nodes (for classification into 1000 classes in ImageNet).
    - Dropout is sometimes applied to the fully connected layers to reduce overfitting.

- **Softmax Layer:**
    - Outputs class probabilities using the softmax activation function.

- **Parameters:**
    - VGG16 has approximately 138 million parameters, mainly due to the fully connected layers.

4. VGG19

VGG19 is a deep learning model developed by the Visual Geometry Group (VGG) at Oxford University. Introduced in the 2014 ImageNet Challenge, it is an extended version of VGG16 with 19 layers in total. VGG19's architecture is noted for its simplicity and depth, featuring 16 convolutional layers and 3 fully connected layers. It achieves remarkable performance in image classification tasks and is pre-trained on ImageNet, making it suitable for transfer learning.

How It Works

VGG19 operates by passing an input image through a series of convolutional layers, each followed by activation using the ReLU (Rectified Linear Unit) function. After every two or three convolutional layers, max pooling is applied to reduce spatial dimensions while retaining key features.

Architecture

- **Input**: Fixed size of 224x224x3 RGB image.
- **Convolutional Layers**: 16 convolutional layers in blocks of two or four, with small 3x3 filters. Each convolution is followed by a ReLU activation.
- **Pooling Layers**: Max pooling layers (2x2) follow every two or three convolutional layers to down-sample the spatial resolution.
- **Fully Connected Layers**: After the convolutional layers, three fully connected layers are used, with the first two having 4096 units each.
- **Output Layer**: A final softmax layer provides class predictions over the number of output classes (typically 1000 for ImageNet).

5. Custom CNN

Custom CNN refers to a convolutional neural network (CNN) architecture that is specifically designed and tailored for a particular application or task. Unlike pre-trained models like VGG, ResNet, or YOLO, which follow standard architectures, Custom CNNs are created by adjusting the network layers, number of filters, kernel sizes, and other hyperparameters according to the nature of the input data and desired

output.

How it works

Custom CNNs operate by following the standard CNN process:

1. **Convolution layers**: These layers apply various filters (kernels) to the input images to capture spatial features like edges, textures, and patterns. The filters in Custom CNN can be optimized or adapted for the specific characteristics of the dataset.
2. **Pooling layers**: Pooling is used to reduce the dimensionality of the data and retain only the most relevant features. Max pooling is typically used to downsample the feature maps.
3. **Flattening**: After multiple convolution and pooling operations, the 2D feature maps are flattened into a 1D vector to be fed into fully connected layers.
4. **Fully connected layers**: These layers perform the final classification by combining the extracted features. The number of neurons and layers is usually customized based on the task at hand.
5. **Output layer**: The output layer typically uses softmax activation for multi-class classification problems, with the number of nodes corresponding to the number of output classes.

Architecture

The architecture of Custom CNNs can be modified based on the complexity of the task and dataset. A typical structure includes:

1. **Input layer**: Receives images, usually with a fixed size, e.g., 64x64x3.
2. **Convolutional layers**: With a customizable number of filters (e.g., 32, 64) and kernel sizes (e.g., 3x3, 5x5).
3. **Max-pooling layers**: To reduce dimensionality (e.g., 2x2 pooling windows).
4. **Fully connected layers**: Depending on the task, there might be 1 or more fully connected layers.
5. **Output layer**: Uses softmax activation for classification tasks.

5.2 MODULES

1. User Interface Module

- User Interaction: This is the entry point of the system where users can upload images of weapons for analysis. The system is designed to be user-friendly, allowing seamless interaction.
- Upload Weapon Image: Users can upload images in various formats (e.g., JPEG, PNG). The system supports high-resolution images to ensure detailed analysis.

2. Preprocessing Module

- Image Resizing:
  - Purpose: Adjusts the dimensions of the uploaded images to a uniform size (e.g., 224x224 pixels) to ensure compatibility with deep learning models.
  - Process: Uses interpolation techniques to maintain image quality during resizing.
- Normalization:
  - Purpose: Scales pixel values to a standard range (e.g., 0 to 1) to improve the convergence of learning algorithms.
  - Process: Converts pixel values from the range [0, 255] to [0, 1] by dividing each pixel value by 255.

3. Advanced Data Augmentation

- Deep Learning Models:
  - Purpose: Enhances the diversity of the training data to improve model robustness.
  - Techniques: Includes rotation, flipping, cropping, and color jittering to generate varied training samples.
- Custom CNN:
  - Purpose: A tailored Convolutional Neural Network designed specifically for weapon detection tasks.
  - Architecture: Includes multiple convolutional layers, pooling layers, and activation functions (e.g., ReLU) to extract relevant features.

- VGG19:
  - Purpose: A pre-trained deep learning model known for its effectiveness in image recognition tasks.
  - Usage: Utilizes transfer learning to leverage pre-trained weights for feature extraction.

4. Feature Fusion Module

- Fusion of CNN & VGG19 Features:
    - Purpose: Combines the strengths of both models to create a more comprehensive feature set.
    - Process: Extracts features from both the Custom CNN and VGG19 models and concatenates them to form a unified feature vector.
- Feature Selection:
    - Purpose: Identifies the most relevant features for classification.
    - Techniques: Uses dimensionality reduction methods (e.g., PCA) to select the most informative features.

5. Classification & Prediction

- Dense Layers:
    - Purpose: Fully connected layers that process the fused features for classification.
    - Architecture: Includes multiple dense layers with dropout regularization to prevent overfitting.
- Softmax Classification:
    - Purpose: Outputs the probability distribution over the different weapon types.
    - Process: Applies the softmax function to the output of the dense layers to generate class probabilities.
- Weapon Type Prediction:
    - Purpose: The final step where the system predicts the type of weapon in the uploaded image.
    - Output: Provides a classification label (e.g., handgun, rifle) along with a confidence score.

5.3 UML DIAGRAMS

UML stands for Unified Modelling Language. UML is a standardized general-purposemodelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta- model and a notation. In the future, some form of method or process also be added to;or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artefacts of software system, as well as for business modelling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS: The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modelling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modelling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

1. Class diagram

the class diagram is used to refine the use case diagram and define a detailed design ofthe system. the class diagram classifies the actors defined in the use case diagram intoa set of interrelated classes. the relationship or association between the classes can be either an "is-a" or "has-a" relationship. each class in the class diagram is capable of providing certain functionalities. these functionalities provided by the class are termed"methods" of the class. apart from this, each class have certain "attributes" that uniquely identify the class.

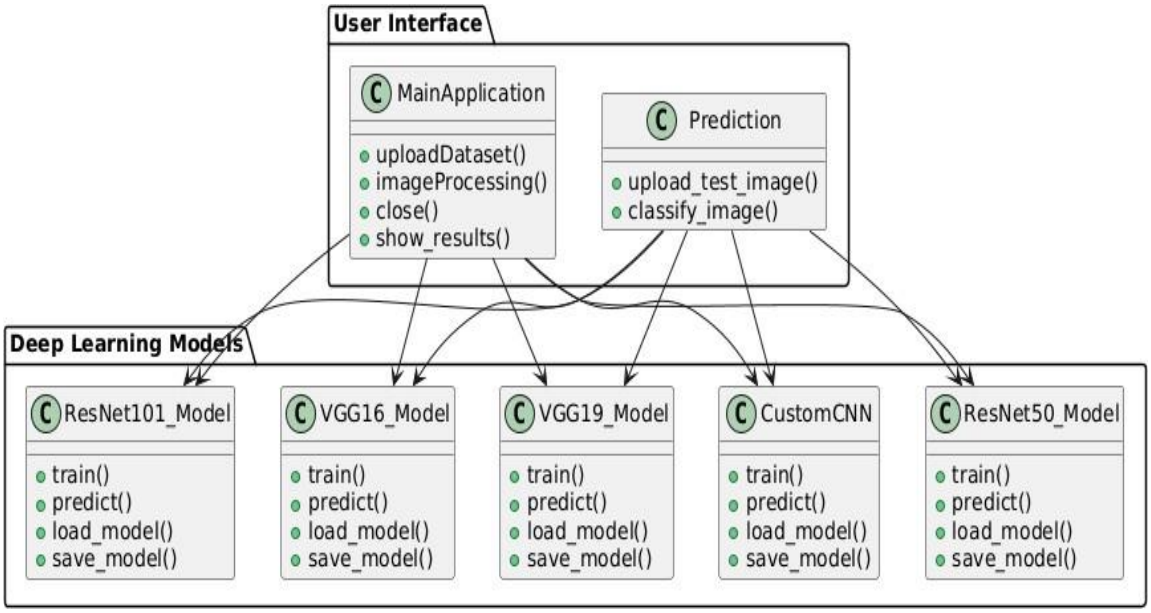

Fig 5.3.1: Class Diagram of Proposed System

2. Use case Diagram

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

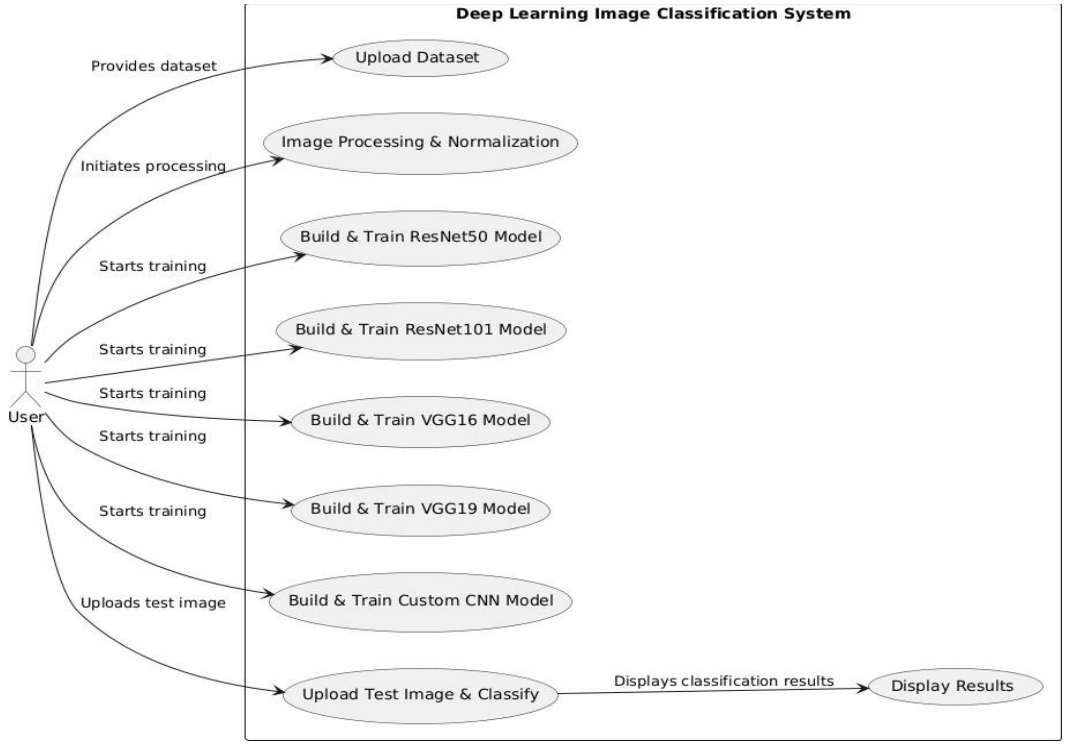

Fig 5.3.2: Use Case Diagram of Proposed System

3. Data Flow Diagram

A data flow diagram (DFD) is a graphical representation of how data moves through a system. It shows the processes that transform data inputs into outputs, the data stores where information is kept, the data flows that connect different parts of the system, and the external entities that interact with the system. DFDs help in understanding the flow of information within a system, identifying data sources and destinations, and designing efficient data processing workflows. They are commonly used in system analysis and design to visualize data movement and improve system understanding.

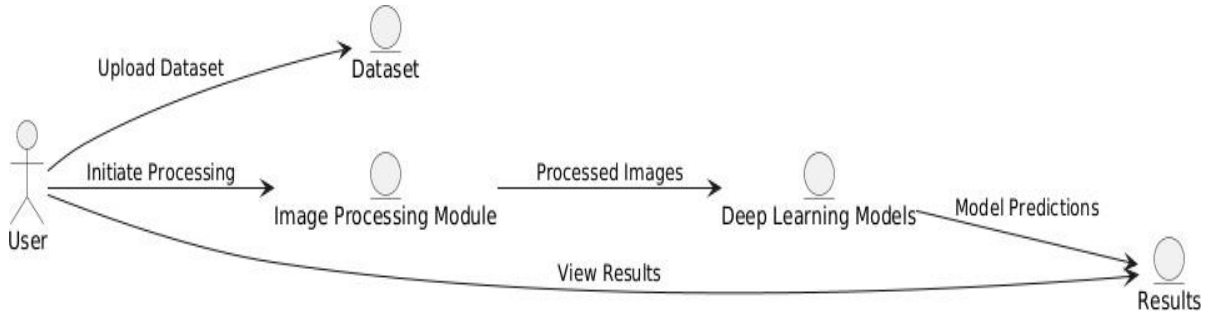

Fig 5.3.3: Data Flow Diagram of Proposed System

4. Activity diagram:

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the businessand operational step-by-step workflows of components in a system. An activity diagramshows the overall flow of control.

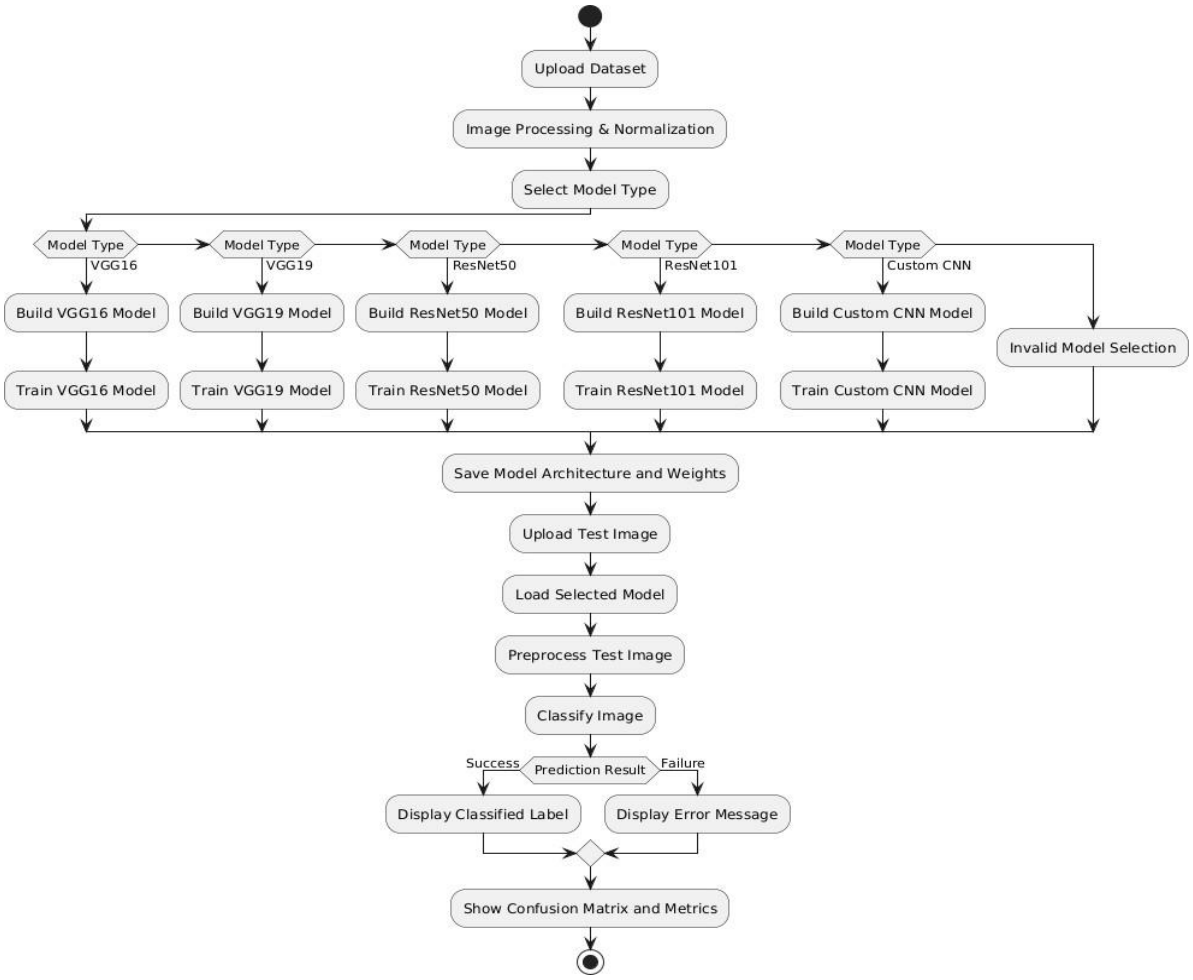

Fig 5.3.4: Activity Diagram of Proposed System

5. Sequence Diagram

A **sequence diagram** in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical

lines ("lifelines"), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

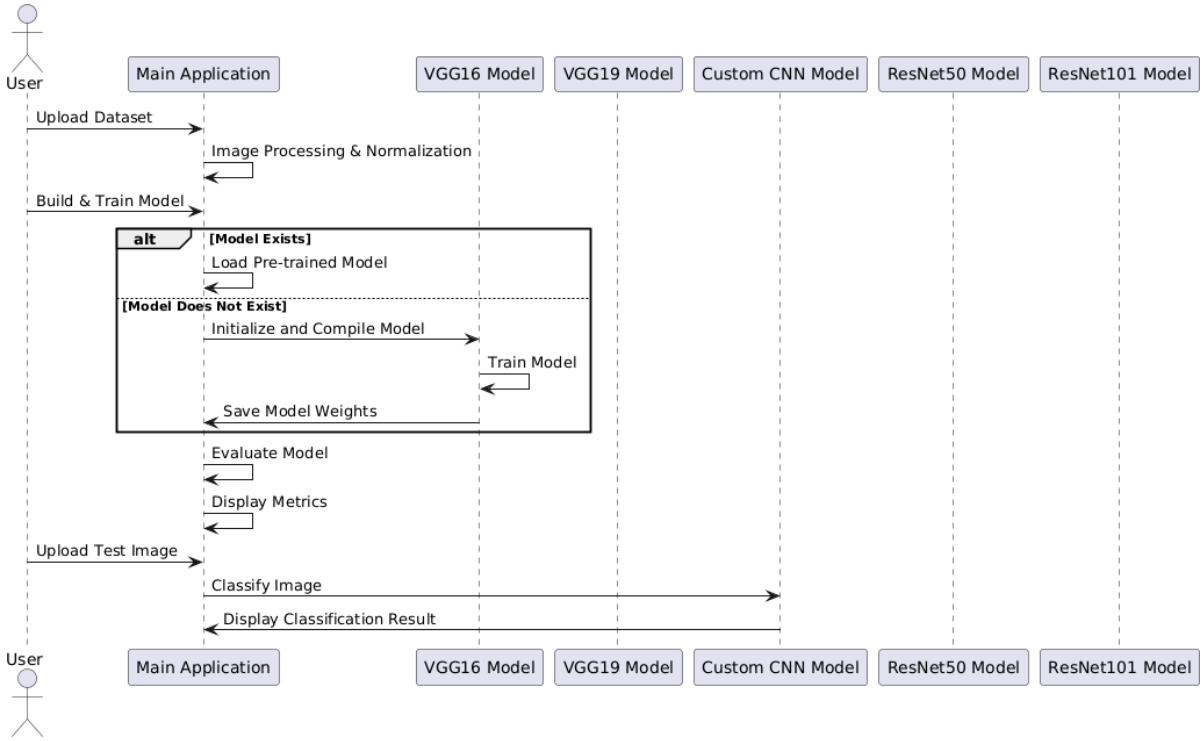

Fig 5.3.5: Sequence Diagram of Proposed System

6. Deployment Diagram:

Describes the deployment architecture of the system, including hardware and software components.

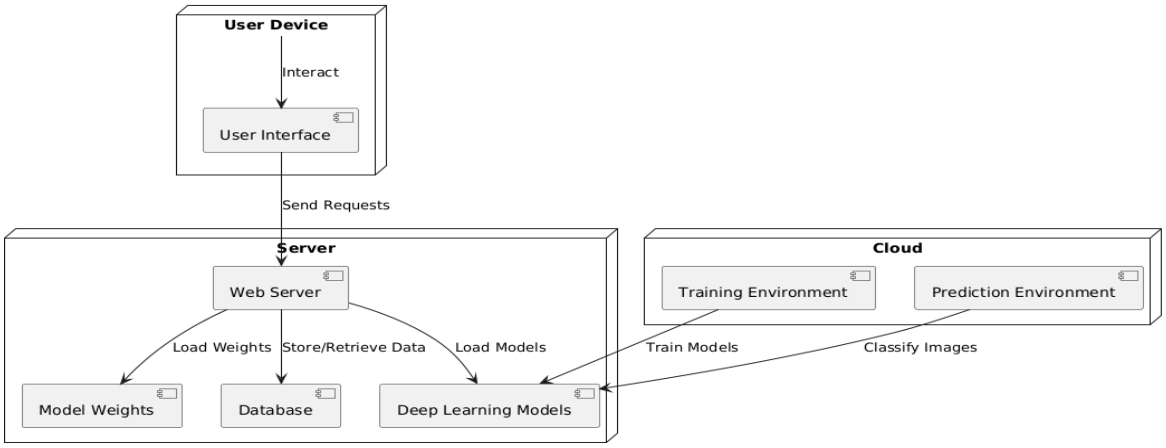

Fig 5.3.6: Deployment Diagram of Proposed System

7. Component Diagram

A component diagram in UML depicts the structural organization of software components and their interactions within a system. It illustrates the modular architecture of a software system by showing the components, their interfaces, dependencies, and relationships. Component diagrams help in understanding the system's structure, promoting modular design, and facilitating communication among development teams. They showcase the building blocks of the system and how they collaborate to achieve the system's functionality. Overall, component diagrams are essential in designing and documenting software systems with a focus on component-based development and system composition.

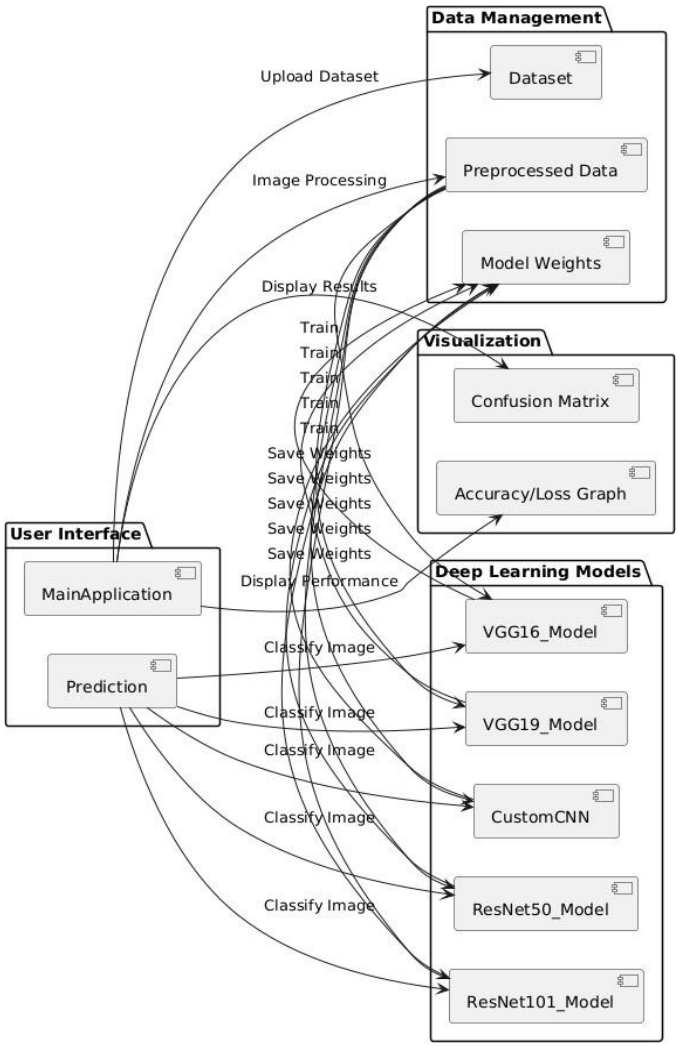

Fig 5.3.7: Component Diagram of Proposed System

6. MODEL IMPLEMENTATION

6.1 IMPLEMENTATION

The Python code is a graphical user interface (GUI) application using the Tkinter library for weapon recognition from images. The application allows users to perform the following tasks:

1. ResNet50 Implementation:

1. **File Loading:** The code checks if the model JSON file and weights are available to load a previously saved model.
2. **Model Loading:** It loads the ResNet50 model's architecture and weights if they exist. It sets up the model to make predictions.
3. **Model Summary:** The code prints the model structure, including layers, parameters, etc.
4. **Training History:** If history exists, the model retrieves and displays the training accuracy of the last epoch.
5. **Training (If Not Preloaded):** If no saved model exists, it imports the pre-trained ResNet50 model with the imagenet weights and adds custom layers for classification.
6. **Custom Layers:** It adds a GlobalAveragePooling2D layer and a dense layer to tailor the output for the classification task.
7. **Layer Freezing:** The pre-trained ResNet50 layers are frozen, preventing their weights from being updated during training.
8. **Model Compilation:** The model is compiled using the Adam optimizer and categorical cross-entropy loss.
9. **Model Training:** The model is trained on the dataset, and the process is displayed (epochs, validation data, batch size, etc.).
10. **Model Saving:** After training, the architecture and weights are saved to disk for later use.
11. **Metrics Calculation:** After training, accuracy, precision, and recall are computed for the test set.

12. **Confusion Matrix:** A confusion matrix is generated and displayed using a heatmap for visualizing the classification performance.

1. ResNet101 Implementation:

1. **File Loading:** Like ResNet50, the script first checks if the ResNet101 model has been saved to load the model and weights.
2. **Model Loading:** If the files exist, the model is restored from disk for further use. This includes loading weights.
3. **Model Summary:** It outputs the ResNet101 model architecture.
4. **Training History:** The code retrieves and displays the accuracy from the model's training history.
5. **Training Process (If Not Saved):** If the model isn't pre-saved, it loads the ResNet101 base model, excluding the top fully connected layers, with imagenet weights.
6. **Custom Layers:** Custom layers, such as pooling and dense layers, are added on top of the base ResNet101 model to adapt it for your specific task.
7. **Freezing Layers:** Pre-trained layers are frozen to maintain their weights, which speeds up training and prevents overfitting.
8. **Compilation:** The model is compiled with categorical cross-entropy and the Adam optimizer.
9. **Training:** Model training occurs with a specific number of epochs, batch size, and validation data.
10. **Saving the Model:** After training, both the architecture and weights are saved to files for later use.
11. **Evaluation:** Accuracy, precision, and recall are computed, and a confusion matrix is generated to visualize classification performance.

2. VGG16 Implementation:

1. **File Loading:** The code checks whether the VGG16 model's architecture and weights have been saved.
2. **Model Loading:** If available, it loads the VGG16 model from disk for further predictions or evaluations.
3. **Model Summary:** It displays the model structure (number of layers, types, parameters, etc.).
4. **Training History:** Retrieves the accuracy of the last epoch if the model was trained before.
5. **Training Process (If Not Saved):** If no model exists, it loads the pre-trained VGG16 model with the imagenet weights, excluding the fully connected layers at the top.
6. **Custom Layers:** GlobalAveragePooling2D and dense layers are added to the VGG16 model, allowing it to adapt to your specific task (e.g., multi-class classification).
7. **Layer Freezing:** All layers in the VGG16 base model are frozen to retain pre-trained weights.
8. **Compilation:** The model is compiled using the Adam optimizer and categorical cross-entropy loss.
9. **Training:** Model is trained with training data, batch size, and validation data.
10. **Model Saving:** After training, the model architecture and weights are saved to disk for future use.
11. **Evaluation:** Predictions are made on the test set, followed by calculating accuracy, precision, and recall.
12. **Confusion Matrix:** A confusion matrix is generated to assess classification results, with visualization via a heatmap.

3. VGG19 Implementation:

1. **File Loading:** As with other models, the code first checks if the VGG19 model exists in files.
2. **Model Loading:** If the model files exist, it loads the architecture and weights from disk.
3. **Model Summary:** It prints the model summary to understand the architecture.
4. **Training History:** It retrieves the history of the training, displaying the accuracy at the last epoch.
5. **Training Process (If Not Saved):** If the model is not preloaded, it uses the pre-trained VGG19 base model with the imagenet weights.
6. **Custom Layers:** GlobalAveragePooling2D and dense layers are added to the VGG19 base for classification.
7. **Freezing Layers:** VGG19 layers are frozen to retain pre-trained knowledge.
8. **Compilation:** The model is compiled using categorical cross-entropy and Adam optimizer.
9. **Training:** The model is trained with batch size, validation data, and other parameters.
10. **Model Saving:** After training, it saves the architecture and weights for future use.
11. **Evaluation:** Predictions are made, and performance metrics (accuracy, precision, recall) are calculated.
12. **Confusion Matrix:** A confusion matrix is computed and visualized using a heatmap.

4. Custom CNN Implementation:

1. **File Loading:** The code checks for the existence of the Custom CNN model JSON and weight files.
2. **Model Loading:** If available, the model and weights are loaded for prediction or further evaluation.
3. **Model Summary:** The model architecture is printed, showing layers and parameter counts.
4. **Training History:** It retrieves and displays the accuracy from the last epoch of a previously trained model.
5. **Training Process (If Not Saved):** If no saved model exists, a custom CNN model is built from scratch.
6. **Layer Architecture:** The architecture includes convolutional layers followed by pooling, flattening, dense layers, and an output softmax layer for classification.
7. **Compilation:** The model is compiled using the Adam optimizer and categorical cross-entropy loss.
8. **Training:** The model is trained with a specified number of epochs, batch size, and validation data.
9. **Model Saving:** After training, the architecture and weights are saved for later use.
10. **Evaluation:** Model predictions on the test set are performed, and accuracy, precision, and recall are calculated.
11. **Confusion Matrix:** A confusion matrix is computed to assess model performance, visualized through a heatmap.
12. **Classification Report:** It generates a classification report summarizing precision, recall, and F1 score across all classes.

5. Performance Metrics

Performance metrics for weapon recognition from images using deep neural networks typically include the following:

Accuracy: Accuracy measures the overall correctness of the predictions made by the neural network model. It is calculated as the ratio of correctly classified instances to the total number of instances in the dataset. In weapon recognition, accuracy indicates how often the model correctly identifies whether an image contains a weapon or not.

Precision: Precision measures the proportion of true positive predictions (correctly identified weapons) out of all positive predictions made by the model. It helps in understanding the reliability of the model when it predicts that an image contains a weapon.

Precision = True Positives / (True Positives + False Positives)

Recall (Sensitivity): Recall measures the proportion of true positive predictions out of all actual positive instances in the dataset. It indicates the ability of the model to correctly detect weapons when they are present in the images.

Recall = True Positives / (True Positives + False Negatives)

F1 Score: The F1 score is the harmonic mean of precision and recall. It provides a single score that balances both precision and recall. F1 score is useful when there is an imbalance between the number of positive and negative instances in the dataset.

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

Specificity: Specificity measures the proportion of true negative predictions (correctly identified non-weapons) out of all actual negative instances in the dataset. It is particularly relevant in scenarios where the consequences of false alarms (misclassifying non-weapons as weapons) are critical.

Specificity = True Negatives / (True Negatives + False Positives)

Confusion Matrix: A confusion matrix provides a detailed breakdown of the model's predictions, showing the number of true positives, true negatives, false positives, and false negatives. It helps in understanding the types of errors made by the model and can inform adjustments to improve performance.

6.2 CODING

```
from tkinter import messagebox from tkinter import *
from tkinter import simpledialog import tkinter
import warnings warnings.filterwarnings('ignore') import matplotlib.pyplot as plt import
numpy as np
import pandas as pd from tkinter import ttk
from tkinter import filedialog
from keras.utils.np_utils import to_categorical from keras.models import Sequential
from keras.layers.core import Dense,Activation,Dropout, Flatten from sklearn.metrics
import accuracy_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import os import cv2
from sklearn.naive_bayes import GaussianNB import joblib
from skimage.transform import resize from skimage.io import imread
from skimage import io, transform
from sklearn.neural_network import MLPClassifier from sklearn.model_selection import
train_test_split from keras.layers import Convolution2D
from keras.layers import MaxPooling2D import pickle
from keras.models import model_from_json from sklearn.metrics import precision_score

from sklearn.metrics import recall_score from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix from sklearn.metrics import
accuracy_score import numpy as np
import os import pickle
from keras.models import Model, model_from_json from keras.applications import
ResNet50
from keras.applications import ResNet101
from keras.layers import Dense, GlobalAveragePooling2D from keras.optimizers import
```

```
Adam
import numpy as np import os
import pickle
from keras.models import Model, model_from_json from keras.applications import
VGG16
from keras.layers import Dense, GlobalAveragePooling2D from keras.optimizers import
Adam
from keras.preprocessing.image import ImageDataGenerator from keras.utils import
to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
recall_score import matplotlib.pyplot as plt
import seaborn as sns
main = Tk()
main.title("Improving Weapon Recognition from Images : Application of Deep Learning
Technique Innovation")
main.geometry("1300x1200")
global filename global X, Y

global model global accuracy global rf_classifier
categories
=['sword','Sniper','SMG','shotgun','Handgun','Grenadelauncher','Bazooka','AutomaticRifle
'] def getID(name):
if name == 'sword': return 0
elif name == 'Sniper': return 1
elif name == 'SMG': return 2
elif name == 'shotgun': return 3
elif name == 'Handgun': return 4
elif name == 'Grenadelauncher':
return 5 # Update this line with the correct index elif name == 'Bazooka':
```

```
return 6
elif name == 'AutomaticRifle': return 7
else:
return -1 # Return an invalid index or handle the unknown category appropriately
def uploadDataset(): global X, Y global filename
text.delete('1.0', END)
filename = filedialog.askdirectory(initialdir=".") text.insert(END,'dataset loaded\n')

def imageProcessing(): text.delete('1.0', END)
global X, Y,X_train,X_test,Y_train,Y_test X = []
Y = [] '''
for root, dirs, directory in os.walk(filename): for j in range(len(directory)):
name = os.path.basename(root)
print(name + " " + root + "/" + directory[j])



if 'Thumbs.db' not in directory[j]:
img_path = os.path.join(root, directory[j])

try:
# Try to read the image
img = cv2.imread(img_path) if img is None:
print(f"Error reading image: {img_path}")
continue # Skip this image and continue with the next one

# Resize the image
img = cv2.resize(img, (64, 64)) im2arr = np.array(img)
im2arr = im2arr.reshape(64, 64, 3) X.append(im2arr) Y.append(getID(name))

except Exception as e:
```

print(f"Error processing image: {img_path}. {e}") continue # Skip this image and continue with the next one

if not X or not Y:
text.insert(END, "No valid images found in the dataset.\n") return

X = np.asarray(X) Y = np.asarray(Y) print(Y)

X = X.astype('float32')
X = X / 255
test = X[3]
test = cv2.resize(test, (400, 400)) cv2.imshow("aa", test) cv2.waitKey(0)
indices = np.arange(X.shape[0]) np.random.shuffle(indices)
X = X[indices] Y = Y[indices]
Y = to_categorical(Y) np.save('model/X.txt', X) np.save('model/Y.txt', Y) '"
text.delete('1.0', END)
X = np.load('model/X.txt.npy') Y = np.load('model/Y.txt.npy')

text.insert(END, "Total classes found in dataset is : " + str(shapes) + "\n")

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.20,random_state=77)
print(Y_test)
print(Y_test.shape)

ResNet50

shapes
=['sword','Sniper','SMG','shotgun','Handgun','Grenadelauncher','Bazooka','AutomaticRifle
']
categories=['sword','Sniper','SMG','shotgun','Handgun','Grenadelauncher','Bazooka','Auto

```
maticRifle']
def res50Model():
global model,acc1,p1,r1,f1 global accuracy,Y_test #text.delete('1.0', END)
if os.path.exists('model/ResNet50_model.json'):
with open('model/ResNet50_model.json', "r") as json_file: loaded_model_json =
json_file.read()
model = model_from_json(loaded_model_json) json_file.close()
model.load_weights("model/ResNet50_model_weights.h5")
model._make_predict_function()
print(model.summary())
f = open('model/ResNet50_history.pckl', 'rb') accuracy = pickle.load(f)
f.close()
acc1 = accuracy['accuracy'] acc1 = acc1[9] * 100
# Load the training history
with open(Model_history, 'rb') as f: accuracy = pickle.load(f)
acc = accuracy['accuracy']
acc = acc[-5] * 100 # Get accuracy of the last epoch print("ResNet50 Model Prediction
Accuracy = " + str(acc))
text.insert(END,"        ResNet50 Model Prediction Accuracy = "+str(acc-1))
else:
# Load the pre-trained ResNet-50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(64, 64,
3))

# Add custom layers on top of the ResNet-50 base x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(len(categories), activation='softmax')(x)
# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)
```

```
# Freeze the layers of the ResNet-50 base model for layer in base_model.layers:
layer.trainable = False
# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
print(model.summary())
# Train the model
hist = model.fit(X_train, Y_train, batch_size=16, epochs=25, validation_data=(X_test,
Y_test), shuffle=True, verbose=2)

# Save the model architecture and weights model_json = model.to_json()
with open(Model_file, "w") as json_file: json_file.write(model_json)
model.save_weights(Model_weights)

# Save the training history
with open(Model_history, 'wb') as f: pickle.dump(hist.history, f)

# Load and print the accuracy
with open(Model_history, 'rb') as f: accuracy = pickle.load(f)
acc = accuracy['accuracy']
acc = acc[-1] * 100 # Get accuracy of the last epoch print("ResNet-50 Model Prediction
Accuracy = " + str(acc))
```

ResNet101

```
def res101Model():
model_folder = "model"
Model_file = os.path.join(model_folder, "ResNet101_model.json") Model_weights =
os.path.join(model_folder, "ResNet101_model_weights.h5") Model_history =
os.path.join(model_folder, "ResNet101_history.pckl")

if os.path.exists(Model_file): # Load the model
```

```
with open(Model_file, "r") as json_file: loaded_model_json = json_file.read()
model = model_from_json(loaded_model_json) model.load_weights(Model_weights)
model._make_predict_function() print(model.summary())

# Load the training history
with open(Model_history, 'rb') as f: accuracy = pickle.load(f)
acc = accuracy['accuracy']
acc = acc[-5] * 100 # Get accuracy of the last epoch

print("/n res101Model Model Prediction Accuracy = " +str(acc)) text.insert(END,"
        res101 Model Prediction Accuracy = "+str(acc))

else:
# Load the pre-trained ResNet-101 model
base_model = ResNet101(weights='imagenet', include_top=False, input_shape=(64, 64,
3))


# Add custom layers on top of the ResNet-101 base x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(len(categories), activation='softmax')(x)

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)
# Freeze the layers of the ResNet-101 base model for layer in base_model.layers:
layer.trainable = False
# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
print(model.summary())
```

```
# Train the model
hist = model.fit(X_train, Y_train, batch_size=16, epochs=25, validation_data=(X_test,
Y_test), shuffle=True, verbose=2)
# Save the model architecture and weights model_json = model.to_json()
with open(Model_file, "w") as json_file: json_file.write(model_json)
model.save_weights(Model_weights)
# Save the training history
with open(Model_history, 'wb') as f: pickle.dump(hist.history, f)

# Load and print the accuracy
with open(Model_history, 'rb') as f: accuracy = pickle.load(f)
acc = accuracy['accuracy']
acc = acc[-1] * 100 # Get accuracy of the last epoch print("ResNet-101 Model Prediction
Accuracy = " + str(acc))
# Make predictions and evaluate the model Y_pred = model.predict(X_test)
Y_pred_classes = np.argmax(Y_pred, axis=1) Y_test_classes = np.argmax(Y_test,
axis=1)
# Compute confusion matrix
conf_matrix = confusion_matrix(Y_test_classes, Y_pred_classes)
# Compute accuracy, precision, and recall
accuracy = accuracy_score(Y_test_classes, Y_pred_classes)
precision = precision_score(Y_test_classes, Y_pred_classes, average='weighted') recall =
recall_score(Y_test_classes, Y_pred_classes, average='weighted')
```

VGG16

```
X = np.load('model2/X.txt.npy') Y = np.load('model2/Y.txt.npy'
def VGG16_Model():
model_folder = "model"
Model_file = os.path.join(model_folder, "ResNet101_model.json")
Model_weights = os.path.join(model_folder, "ResNet101_model_weights.h5")
```

```
Model_history = os.path.join(model_folder, "ResNet101_history.pckl"
if os.path.exists(Model_file)
# Define file paths model_folder = "model2"
Model_file = os.path.join(model_folder, "model.json") Model_weights =
os.path.join(model_folder, "model_weights.h5") Model_history =
os.path.join(model_folder, "history.pckl"
# Load the model
with open(Model_file, "r") as json_file: loaded_model_json = json_file.read()
model = model_from_json(loaded_model_json) model.load_weights(Model_weights)
model._make_predict_function() print(model.summary())
# Load the training history
with open(Model_history, 'rb') as f: accuracy = pickle.load(f)
acc = accuracy['accuracy']
acc = acc[-1] * 100 # Get accuracy of the last epoch print("VGG-16 Model Prediction
Accuracy = " + str(acc))
text.insert(END,"        VGG-16 Model Prediction Accuracy = "+str(acc)) else:
# Load the pre-trained VGG-16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(64, 64, 3)
# Add custom layers on top of the VGG-16 base x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(len(categories), activation='softmax')(x)
# Create the final model
model = Model(inputs=base_model.input, outputs=predictions
# Freeze the layers of the VGG-16 base model for layer in base_model.layers:
layer.trainable = False
# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
print(model.summary())
# Train the model
```

```
hist = model.fit(X_train, Y_train, batch_size=16, epochs=25, validation_data=(X_test,
Y_test), shuffle=True, verbose=2)
# Save the model architecture and weights model_json = model.to_json()
with open(Model_file, "w") as json_file: json_file.write(model_json)
model.save_weights(Model_weights
# Save the training history
with open(Model_history, 'wb') as f: pickle.dump(hist.history, f)
# Load and print the accuracy
with open(Model_history, 'rb') as f: accuracy = pickle.load(f)
acc = accuracy['accuracy']
acc = acc[-1] * 100 # Get accuracy of the last epoch
print("VGG-16 Model Prediction Accuracy = " + str(acc)
# Make predictions and evaluate the model Y_pred = model.predict(X_test)
Y_pred_classes = np.argmax(Y_pred, axis=1) Y_test_classes = np.argmax(Y_test,
axis=1)
# Compute confusion matrix
conf_matrix = confusion_matrix(Y_test_classes, Y_pred_classes
# Compute accuracy, precision, and recall
accuracy = accuracy_score(Y_test_classes, Y_pred_classes)
precision = precision_score(Y_test_classes, Y_pred_classes, average='weighted') recall =
recall_score(Y_test_classes, Y_pred_classes, average='weighted')
# Print metrics
print(f'Confusion Matrix:\n{conf_matrix}') print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}') print(f'Recall: {recall:.4f}')
# Plot confusion matrix plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=categories,
yticklabels=categories)
plt.xlabel('Predicted Labels') plt.ylabel('True Labels') plt.title('Confusion Matrix')
plt.show()
```

VGG19

```
from keras.models import Model, model_from_json
from keras.applications import VGG19 # Updated import for VGG-19
X = np.load('model/X.txt.npy') Y = np.load('model/Y.txt.npy')
# Define file paths model_folder = "model"
Model_file = os.path.join(model_folder, "VGG19_DLmodel.json") Model_weights =
os.path.join(model_folder, "VGG19_DLmodel_weights.h5") Model_history =
os.path.join(model_folder, "history.pckl")
def VGG19_Model():
if os.path.exists(Model_file): # Load the model
with open(Model_file, "r") as json_file: loaded_model_json = json_file.read()
model = model_from_json(loaded_model_json) model.load_weights(Model_weights)
model._make_predict_function() print(model.summary())
# Load the training history
with open(Model_history, 'rb') as f: accuracy = pickle.load(f)
acc = accuracy['accuracy']
acc = acc[-1] * 100 # Get accuracy of the last epoch print("VGG-19 Model Prediction
Accuracy = " + str(acc))
text.insert(END,"        VGG-19 Model Prediction Accuracy = "+str(acc))
else:
# Load the pre-trained VGG-19 model
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(64, 64, 3))
# Add custom layers on top of the VGG-19 base x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(len(categories), activation='softmax')(x)
# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the layers of the VGG-19 base model for layer in base_model.layers:
```

```
layer.trainable = False
# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
print(model.summary())
# Train the model
hist = model.fit(X_train, Y_train, batch_size=32, epochs=50, validation_data=(X_test,
Y_test), shuffle=True, verbose=2)
# Save the model architecture and weights model_json = model.to_json()
with open(Model_file, "w") as json_file: json_file.write(model_json)
model.save_weights(Model_weights)
# Save the training history
with open(Model_history, 'wb') as f: pickle.dump(hist.history, f)
# oad and print the accuracy
with open(Model_history, 'rb') as f: accuracy = pickle.load(f)
acc = accuracy['accuracy']
acc = acc[-1] * 100 # Get accuracy of the last epoch
print("  VGG-19 Model Prediction Accuracy = " + str(acc))

# Make predictions and evaluate the model Y_pred = model.predict(X_test)
Y_pred_classes = np.argmax(Y_pred, axis=1) Y_test_classes = np.argmax(Y_test,
axis=1)

# Compute confusion matrix
conf_matrix = confusion_matrix(Y_test_classes, Y_pred_classes)

# Compute accuracy, precision, and recall
accuracy = accuracy_score(Y_test_classes, Y_pred_classes)
precision = precision_score(Y_test_classes, Y_pred_classes, average='weighted') recall =
recall_score(Y_test_classes, Y_pred_classes, average='weighted')
```

```
# Print metrics
print(f'Confusion Matrix:\n{conf_matrix}') print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}') print(f'Recall: {recall:.4f}')

# Plot confusion matrix plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=categories, yticklabels=categories) plt.xlabel('Predicted Labels')
plt.ylabel('True Labels') plt.title('Confusion Matrix') plt.show()
```

Custom-CNN

```
def cnnModel():
global model,acc1,p1,r1,f1 global accuracy,Y_test #text.delete('1.0', END)
if os.path.exists('modell/model.json'):
with open('modell/model.json', "r") as json_file: loaded_model_json = json_file.read()
model = model_from_json(loaded_model_json) json_file.close()
model.load_weights("modell/model_weights.h5") model._make_predict_function()
print(model.summary())
f = open('modell/history.pckl', 'rb') accuracy = pickle.load(f)
f.close()
acc1 = accuracy['accuracy'] acc1 = acc1[9] * 100
text.insert(END,"Deep Learning Model Accuracy is = "+str(acc1)+ "\n") y_pred1 =
model.predict(X_test)
y_pred1=np.argmax(y_pred1,axis=1) # Example: Reshape a singleton array Y_test =
np.argmax(Y_test,axis=1) print(Y_test)
cm = confusion_matrix(Y_test, y_pred1)

clr = classification_report(Y_test, y_pred1, target_names=shapes) p1 =
precision_score(Y_test, y_pred1, average='macro') * 100
r1 = recall_score(Y_test, y_pred1, average='macro') * 100 f1 = f1_score(Y_test, y_pred1,
```

```
average='macro') * 100
# Print performance metrics text.delete('1.0', END) print("Classification Report:\n", clr)

class_names =
('sword','Sniper','SMG','shotgun','Handgun','Grenadelauncher','Bazooka','AutomaticRifle')
sns.heatmap(cm, annot = True, cmap = "Blues", xticklabels = class_names, yticklabels =
class_names) plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix of CNN Classifier") plt.show()
#Accuracy comparision graph acc1 = accuracy['accuracy'] loss = accuracy['loss']
plt.figure(figsize=(10,6)) plt.grid(True) plt.xlabel('Iterations') plt.ylabel('Accuracy/Loss')
plt.plot(acc1, 'ro-', color = 'green') plt.plot(loss, 'ro-', color = 'blue') plt.legend(['Accuracy',
'Loss'], loc='upper left') #plt.xticks(wordloss.index) plt.title('Performance Evaluation')
plt.show()
else:
model = Sequential() #resnet transfer learning code here model.add(Convolution2D(32,
3, 3, input_shape = (64, 64, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Convolution2D(32, 3, 3, activation = 'relu'))

model.add(MaxPooling2D(pool_size = (2, 2))) model.add(Flatten())
model.add(Dense(output_dim = 256, activation = 'relu')) model.add(Dense(output_dim =
8, activation = 'softmax'))
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =
['accuracy']) print(model.summary())
hist = model.fit(X, Y, batch_size=16, epochs=10, validation_split=0.2, shuffle=True,
verbose=2) model.save_weights('modell/model_weights.h5')
model_json = model.to_json()
with open("modell/model.json", "w") as json_file: json_file.write(model_json)
json_file.close()
f = open('modell/history.pckl', 'wb') pickle.dump(hist.history, f) f.close()
```

```
f = open('modell/history.pckl', 'rb') accuracy = pickle.load(f)
f.close()
acc = accuracy['accuracy'] acc = acc[9] * 100
text.insert(END," CNN Model Prediction Accuracy = "+str(acc))

def predict(): global model
filename = filedialog.askopenfilename(initialdir="testImages") img =
cv2.imread(filename)
img = cv2.resize(img, (64,64)) im2arr = np.array(img)
im2arr = im2arr.reshape(1,64,64,3) test = np.asarray(im2arr)
test = test.astype('float32') test = test/255

preds = model.predict(test) predict = np.argmax(preds) img = cv2.imread(filename)
img = cv2.resize(img, (500,500))
cv2.putText(img, 'Image Classified as : '+shapes[predict], (10, 25),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
cv2.imshow('Image Classified as : '+shapes[predict], img) cv2.waitKey(0)

def close(): main.destroy()
font = ('times', 15, 'bold')
title = Label(main, text='Improving Weapon Recognition from Images : Application of
Deep Learning Technique Innovation')
title.config(bg='powder blue', fg='olive drab') title.config(font=font)
title.config(height=3, width=120) title.place(x=0,y=5)
#FrontEnd
font1 = ('times', 13, 'bold')
ff = ('times', 12, 'bold')
uploadButton = Button(main, text="Upload Dataset", command=uploadDataset)
uploadButton.place(x=20,y=100)
uploadButton.config(font=ff)
```

```
processButton = Button(main, text="Image Processing & Normalization",
command=imageProcessing) processButton.place(x=20,y=150)
processButton.config(font=ff)
modelButton = Button(main, text="Build & Train Resnet50 Model",
command=res50Model) modelButton.place(x=20,y=200)
modelButton.config(font=ff)

modelButton = Button(main, text="Build & Train resnet101 Model",
command=res101Model) modelButton.place(x=20,y=250)
modelButton.config(font=ff)
modelButton = Button(main, text="Build & Train VGG16 Model",
command=VGG16_Model) modelButton.place(x=20,y=300)
modelButton.config(font=ff)
modelButton = Button(main, text="Build & Train VGG19 Model",
command=VGG19_Model) modelButton.place(x=20,y=350)
modelButton.config(font=ff)
modelButton = Button(main, text="Build & Train CUSTOM CNN Model",
command=cnnModel) modelButton.place(x=20,y=400)
modelButton.config(font=ff)
predictButton = Button(main, text="Upload Test Image & Classify", command=predict)
predictButton.place(x=20,y=450)
predictButton.config(font=ff)
exitButton = Button(main, text="Exit", command=close) exitButton.place(x=20,y=500)
exitButton.config(font=ff)
font1 = ('times', 12, 'bold') text=Text(main,height=30,width=85) scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set) text.place(x=450,y=100)
text.config(font=font1)
main.config() main.mainloop()
```

Frontend

```
import os import sys

def main():
"""Run administrative tasks."""
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'weapon_detection.settings')
try:
from django.core.management import execute_from_command_line except ImportError
as exc:
raise ImportError(
"Couldn't import Django. Are you sure it's installed and "
"available on your PYTHONPATH environment variable? Did you " "forget to activate a
virtual environment?"
) from exc execute_from_command_line(sys.argv)

if   name        == '  main  ':
main()
```

about.html

```
{% extends "index.html" %}
{% load static %}
{% block main %}
<style> p,h6,li{
color: aliceblue;
}
body{
overflow: hidden;
}
</style>
<div class="container" style="height: 450px;padding-right:100px">
```

The project titled "A Deep Learning Approach for Automated Weapon Detection" focuses on developing an intelligent system capable of identifying weapons in real-time using advanced deep learning techniques. The primary objective is to enhance public safety by enabling prompt responses to potential threats in various environments, such as public spaces, educational institutions, and secured facilities.
<h6>Project Overview:</h6>
Data Collection and Preprocessing:
Dataset Compilation: Gather a comprehensive dataset comprising images and videos containing various weapon types, including firearms and knives, as well as non-weapon images to ensure model robustness.
Annotation: Manually label the collected data to identify and localize weapons within the images, creating bounding boxes around the detected weapons.
Data Augmentation: Apply techniques such as rotation, scaling, and flipping to increase dataset diversity and improve model generalization.

Model Development:
Architecture Selection: Utilize state-of-the-art object detection models like YOLOv5, YOLOv8, or Faster R-CNN, known for their balance between accuracy and real-time performance.
Transfer Learning: Employ pre-trained models on large datasets to leverage existing feature extraction capabilities, fine-tuning them on the weapon detection dataset to enhance performance.
Training: Train the model using the prepared dataset, optimizing hyperparameters to achieve high detection accuracy and minimizing false positives and negatives.

System Integration:
Real-Time Processing: Implement the trained model into a real-time video surveillance system capable of processing live feeds and detecting weapons instantaneously.
Alert Mechanism: Develop an alert system that notifies security personnel upon weapon detection, providing details such as weapon type and location within the

frame.
Scalability: Ensure the system can handle multiple video feeds simultaneously, maintaining performance across various scenarios and environments.

Evaluation and Testing:
Performance Metrics: Assess the model using metrics like precision, recall, F1-score, and mean Average Precision (mAP) to evaluate detection accuracy.
Real-World Testing: Deploy the system in controlled environments to test its effectiveness in real-world scenarios, making necessary adjustments based on performance.
Continuous Improvement: Incorporate feedback and new data to retrain and enhance the model, adapting to emerging threats and reducing the likelihood of detection errors.

{% endblock %}

Chart.html

```
{% extends "index.html" %}
{% load static %}
{% block main %}
<style> img{
height: 350px; margin-top: -10px; width: 980px;
}
</style>
<img src="static/images/Image 2024-12-29 at 12.28.35_bb3cd1e0.jpg"/>
{% endblock %}
```

Home.html

```
{% extends "index.html" %}
{% load static %}
{% block main %}
<style> h1{
color: aliceblue; display: flex;
justify-content: center; text-align: center;
}
img{
height: 300px; margin-top: -10px; width: 980px;
}
</style>
<div>
{% if messages %}
{% for message in messages %}
<p>{{ message }}</p>
{% endfor %}
{% endif %}
img src="static/images/automated-weapon-detection-deep-learning-
1060x651.jpg.webp"/>
<div>
<p style="color: aliceblue;margin-bottom:0px">Actuate’s AI gun detection software
turns your existing security cameras into gun-detecting smart cameras, instantly detecting
threats as soon as they emerge. With no hardware integrations required, Actuate saves
money and saves lives</p>
</div>
</div>
{% endblock %}
```

Index.html

```
{% load static %}
<html>
<head>
<title>Enhancing Hiring process through Block Chain</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<link href="{% static 'css/hiring.css' %}" rel="stylesheet" type="text/css"
media="screen" />
<style> body {
overflow: hidden;
}
.container::-webkit-scrollbar { width: 10px;
}
.container::-webkit-scrollbar-track { background: #f1f1f1;
}
.container::-webkit-scrollbar-thumb { background: #888;
}
.container::-webkit-scrollbar-thumb:hover { background: #555;
}
.container {
scrollbar-width: thin;
scrollbar-color: #888 #363333;
}
.entry {
overflow-y: scroll; height: 350px; width: 1000px;
}
#logo img { position: absolute; top: 10px;
left: 10px;
width: 80px; /* Adjust size as needed */ height: auto;
}
```

```
#logo h1 {
margin-left: 100px; /* Offset text to avoid overlapping the logo */ font-size: 30px;
```

```
letter-spacing: 1.4px; color: red;
</style>
</head>
<body>
<div id="wrapper">
<div id="header">
<div id="logo">
<img src="{% static 'images/images.png' %}" alt="Logo">
<h1>A Deep Learning Approach for Automated Weapon Detection</h1>
<marquee><font color="black" size=4>Team Members: 1.Pavan Dath 2. Rajavamshi 3.
M.Venkatasai | Guide: K. Suresh Babu | Coordinator: D. Venkata
Reddy</font></marquee>
</div>
</div>
<div id="menu">
<ul>
<li><a href="/">Home</a></li>
<li><a href="about">About</a></li>
<li><a href="prediction">Prediction</a></li>
<li><a href="modelevaluation">Model Evaluation</a></li>
<li><a href="chart">Project Flow Chart</a></li>
</ul>
</div>
<div id="page">
<div id="content">
<div class="post">
<div class="title">
<h2>Automated Weapon Detection</h2>
</div>
<div class="entry container">
```

{% block main %}{% endblock %}

</body>

</html>

Modelevaluation.html

{% extends "index.html" %}

{% load static %}

{% block main %}

<style>

table {

margin: 0 auto;

border-collapse: collapse; background-color: white;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); width: 400px;

}

th, td {

padding: 15px; text-align: left;

border-bottom: 1px solid #ddd;

}

th {

background-color:#209D9D; color: white;

}

tr:nth-child(even) { background-color: #f2f2f2;

}

tr:hover {

background-color: #ddd;

}

caption { font-size: 1.5em;

```
margin-bottom: 10px;
}
/* For Chrome, Edge, and Safari */
</style>
<div class="container" style="height:450px;padding-right:100px">
<!--
<h3>EXISTING MODEL</h3>
<div style="display: flex;">
<table>
<caption style="color: #ccc;">CALCULATION METRICS</caption>
<thead>
<tr>
<th>{{algorithm1}}</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>{{accuracy1}}</td>
</tr>
<tr>
<td>precision</td>
<td>{{precision1}}</td>
</tr>
<tr>
<td>recall</td>
<td>{{recall1}}</td>
</tr>
<tr>
```

<td>fscore</td>

<td>{{fscore1}}</td>

</tr>

</tbody>

</table>

-->

<h3>Proposed MODEL</h3>

<table>

<caption style="color: #ccc;">CALCULATION METRICS</caption>

<thead>

<tr>

<th>{{algorithm}}</th>

<th>Score</th>

</tr>

</thead>

<tbody>

<tr>

<td>accuracy</td>

<td>{{accuracy}}</td>

</tr>

<tr>

<td>precision</td>

<td>{{precision}}</td>

</tr>

<tr>

<td>recall</td>

<td>{{recall}}</td>

</tr>

<tr>

<td>fscore</td>

<td>{{fscore}}</td>

</tr>

</tbody>

</table>

{% endblock %}

Predict.html

{% extends 'index.html' %}

{% load static %}

{% block main %}

<style>

/* Form Container */ form {

width: 80%;

max-width: 500px; margin: 50px auto;

padding: 20px; background-color: #f9f9f9; border-radius: 8px;

border: 1px solid black;

box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);

/* Title or Heading */ h1 {

text-align: center;

}

```
/* Message Section */ p {
font-size: 14px; color: #d9534f; text-align: center;
}
/* Input Group Styling */
.inputGroup {
margin-bottom: 15px;
}
label {
font-size: 16px; font-weight: bold; color: #333; display: block;
margin-bottom: 8px;
}
input[type="file"] {
width: 95%; padding: 10px;
border: 2px solid #ccc; border-radius: 4px; cursor: pointer;
font-size: 14p
}
.submitForm {
background-color: #209D9D; color: white;
padding: 10px 20px; border: none;
border-radius: 5px; font-size: 16px; cursor: pointer; display: block; margin: 20px auto;
transition: background-color 0.3s ease;

.submitForm:hover { background-color: #45a049;
}
/* Centering the form content */
.submitForm { width: auto;
}
.output{
color: #f2f2f2; display: flex;
justify-content:center; background-color: #ccc;
```

```
width: 300px; padding: 20px; border-radius: 5px;
border: 1px solid green; font-size: 18px;
margin-left: 340px;

margin-top:-30px;
}
</style>
<form action="{% url 'prediction' %}" method="post" enctype="multipart/form-data">
{% csrf_token %}
<div class="inputGroup">
<label for="name">Test image </label>
<input type="file" id="name" name="file" placeholder="Enter Name" required
style="cursor: pointer;">
</div>
<button class="submitForm" style="margin-left:200px">Predict</button>
</form>
<div>
{% if messages %}
{% for message in messages %}
<p class="output" style="color: #4CAF50;">predicted as : {{ message }}</p>
{% endfor %}
{% endif %}
</div>
{% endblock %
```

7. TESTING

In software development, effective testing is crucial to ensure code correctness, reliability, and performance. The primary types of testing include unit testing, integration testing, and system testing, each serving distinct purposes and aligning with different stages of the development cycle.

1. Unit Testing

Definition: Unit testing involves testing individual components (or "units") of the software in isolation to verify that each part behaves as expected. This is typically done at the function or method level.

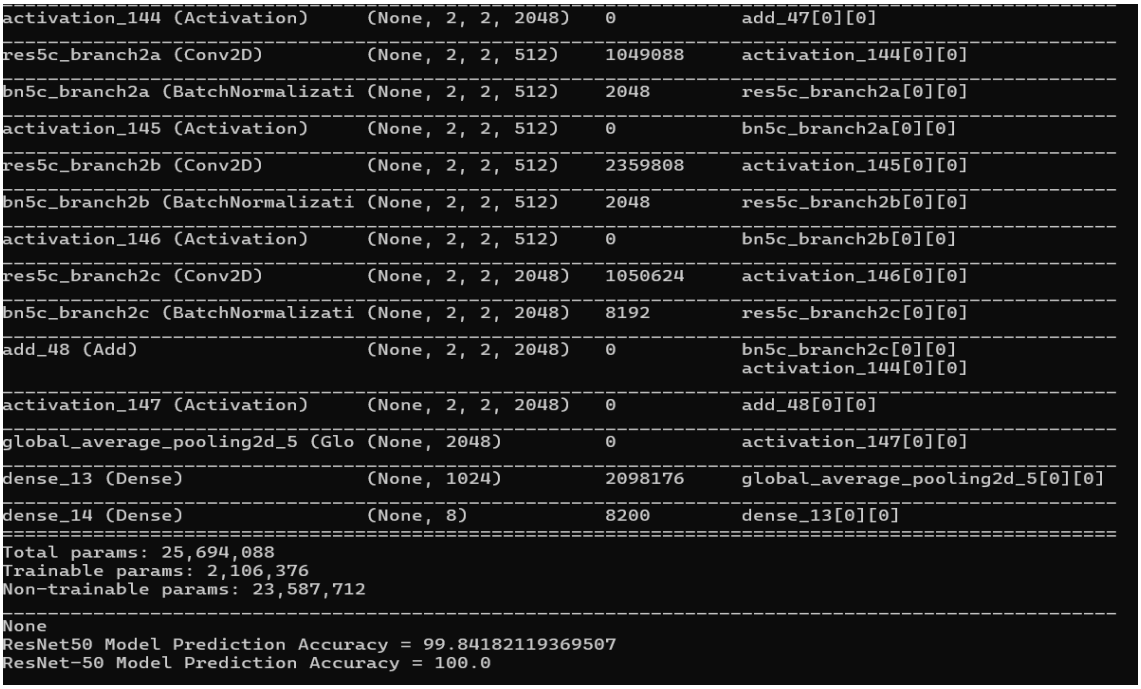

Fig 7.1: ResNet-50 Unit Testing

```
conv5_block2_3_bn (BatchNormali (None, 2, 2, 2048)   8192        conv5_block2_3_conv[0][0]
__________________________________________________________________________________________
conv5_block2_add (Add)          (None, 2, 2, 2048)   0           conv5_block1_out[0][0]
                                                                 conv5_block2_3_bn[0][0]
__________________________________________________________________________________________
conv5_block2_out (Activation)   (None, 2, 2, 2048)   0           conv5_block2_add[0][0]
__________________________________________________________________________________________
conv5_block3_1_conv (Conv2D)    (None, 2, 2, 512)    1049088     conv5_block2_out[0][0]
__________________________________________________________________________________________
conv5_block3_1_bn (BatchNormali (None, 2, 2, 512)    2048        conv5_block3_1_conv[0][0]
__________________________________________________________________________________________
conv5_block3_1_relu (Activation (None, 2, 2, 512)    0           conv5_block3_1_bn[0][0]
__________________________________________________________________________________________
conv5_block3_2_conv (Conv2D)    (None, 2, 2, 512)    2359808     conv5_block3_1_relu[0][0]
__________________________________________________________________________________________
conv5_block3_2_bn (BatchNormali (None, 2, 2, 512)    2048        conv5_block3_2_conv[0][0]
__________________________________________________________________________________________
conv5_block3_2_relu (Activation (None, 2, 2, 512)    0           conv5_block3_2_bn[0][0]
__________________________________________________________________________________________
conv5_block3_3_conv (Conv2D)    (None, 2, 2, 2048)   1050624     conv5_block3_2_relu[0][0]
__________________________________________________________________________________________
conv5_block3_3_bn (BatchNormali (None, 2, 2, 2048)   8192        conv5_block3_3_conv[0][0]
__________________________________________________________________________________________
conv5_block3_add (Add)          (None, 2, 2, 2048)   0           conv5_block2_out[0][0]
                                                                 conv5_block3_3_bn[0][0]
__________________________________________________________________________________________
conv5_block3_out (Activation)   (None, 2, 2, 2048)   0           conv5_block3_add[0][0]
__________________________________________________________________________________________
global_average_pooling2d_6 (Glo (None, 2048)         0           conv5_block3_out[0][0]
__________________________________________________________________________________________
dense_15 (Dense)                (None, 1024)         2098176     global_average_pooling2d_6[0][0]
__________________________________________________________________________________________
dense_16 (Dense)                (None, 8)            8200        dense_15[0][0]
==========================================================================================
Total params: 44,764,552
Trainable params: 2,106,376
Non-trainable params: 42,658,176
__________________________________________________________________________________________
None
/n res101Model Model Prediction Accuracy = 98.89275431632996
```

Fig 7.2: ResNet-101 Unit Testing

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               1605888
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 2056
=================================================================
Total params: 1,618,088
Trainable params: 1,618,088
Non-trainable params: 0
_________________________________________________________________
None
VGG-16 Model Prediction Accuracy = 99.26380515098572
```

Fig 7.3: VGG-16 Unit Testing

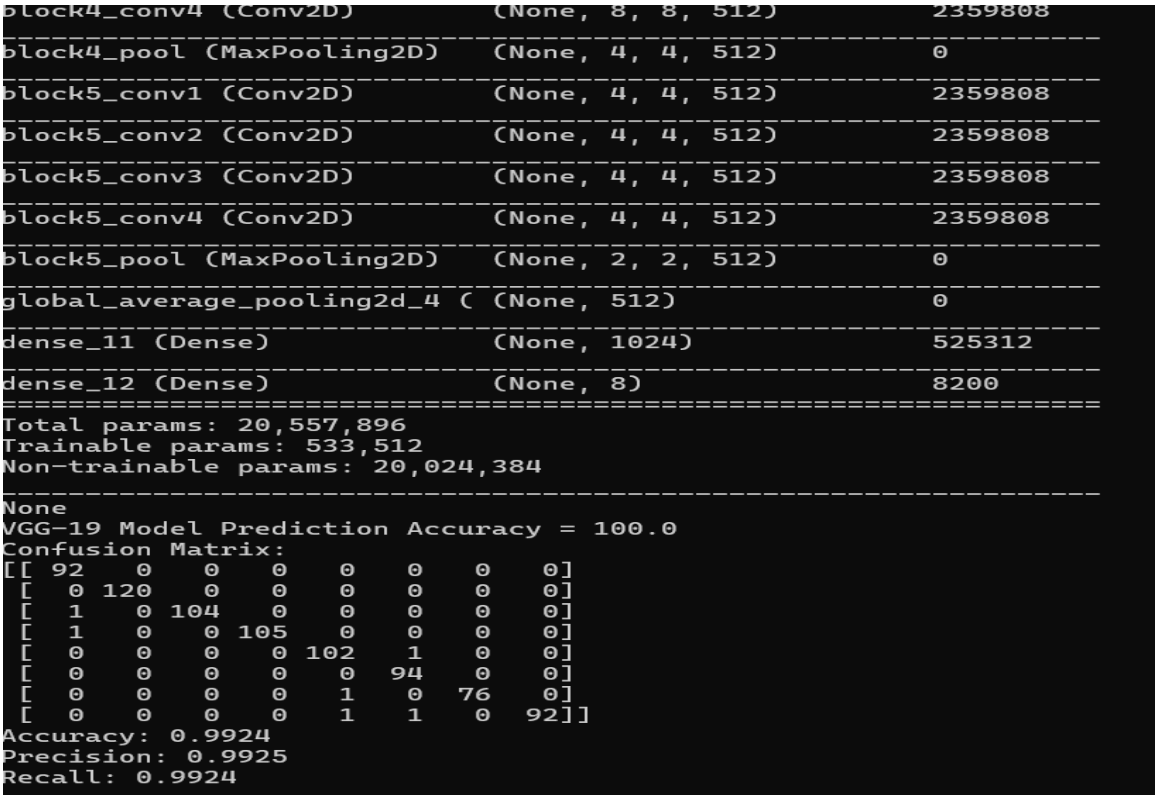

Fig 7.4: VGG-19 Unit Testing

2. Integration Testing:

Integration testing focuses on evaluating the interactions between components of a system, in this case, the connection between the frontend and backend of your application.

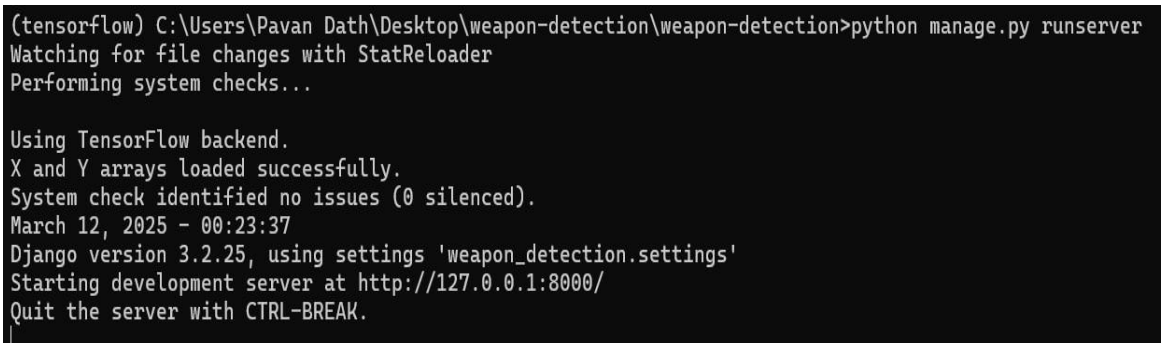

Fig 7.5: Output of connection between Frontend and Backend

3. System Testing:

System testing is a comprehensive evaluation of the entire weapon detection system as a unified entity to ensure that it meets the specified requirements and operates correctly in various scenarios. In your context, system testing would involve executing the proposed model to validate its performance and behaviour.

```
None
[1 6 1 6 2 4 2 0 3 4 2 1 7 5 0 7 1 1 6 2 7 5 0 4 0 5 2 7 7 0 1 6 1 5 3 4 2
 0 4 4 6 1 5 0 7 2 1 7 3 1 3 0 5 4 1 1 4 5 3 2 4 5 5 5 5 4 2 3 4 5 2 3 2 7
 6 2 2 7 7 0 2 7 2 3 0 4 4 4 1 3 0 0 2 7 7 7 3 1 4 4 6 1 4 1 1 7 1 4 4 4 6
 0 0 4 7 7 4 7 4 0 2 1 3 5 5 6 5 2 0 5 3 3 7 4 2 3 3 7 2 0 5 3 5 1 4 5 1 7
 0 3 2 0 2 1 2 7 4 0 2 5 7 4 2 4 2 3 5 6 5 4 2 0 6 5 3 4 6 4 5 5 0 1 1 4 2
 6 6 1 1 2 0 3 1 7 7 2 0 4 0 0 1 3 5 3 2 3 4 5 4 4 6 0 5 4 3 1 1 5 5 7 3 2
 7 7 6 3 0 6 1 0 1 4 5 4 1 1 2 0 7 2 1 6 6 0 4 2 4 4 3 6 2 4 3 0 4 2 2 6 7
 4 2 6 4 3 0 1 5 4 7 6 5 4 2 2 5 5 2 3 7 4 0 5 0 3 0 0 6 2 2 0 7 1 5 1 2 0
 0 3 4 1 0 1 7 2 1 6 4 2 0 6 2 1 2 0 0 6 6 6 1 3 2 4 4 3 1 7 0 5 1 7 1 4 1
 2 0 7 5 3 1 3 5 3 1 7 2 0 0 1 6 1 3 4 0 0 3 5 3 3 7 4 7 2 0 2 4 2 1 2 3 3
 1 6 1 2 3 5 3 3 3 2 4 5 6 7 3 5 2 6 4 1 4 4 6 2 4 4 5 2 0 1 2 7 3 0 3 5 4
 6 7 1 6 2 7 0 3 1 3 2 1 3 0 1 0 2 3 7 2 5 5 6 3 7 0 6 2 3 2 4 2 1 3 5 0 7
 6 5 4 3 4 6 5 6 6 2 5 0 0 5 0 1 6 0 5 5 2 1 1 5 2 4 7 2 5 7 7 2 4 2 4 7 7
 5 3 1 7 7 6 3 4 2 5 5 3 1 6 6 4 4 2 5 7 5 6 6 1 3 1 1 7 1 5 4 1 2 7 2 0 6
 5 6 2 3 1 3 5 1 1 4 2 3 2 6 5 5 7 2 0 5 3 1 5 5 3 7 5 5 2 0 5 2 3 4 0 7 0
 2 1 3 7 7 4 5 1 1 1 1 7 3 0 1 1 1 5 7 3 3 5 4 0 4 1 0 0 2 4 4 4 3 3 3 3 3
 2 5 1 0 6 7 5 3 5 6 3 5 0 2 2 6 4 6 1 2 0 6 1 4 0 7 0 3 5 0 5 7 1 3 7 7 3
 7 7 3 6 2 4 1 6 6 4 3 3 1 7 6 6 2 4 5 7 1 6 3 0 6 0 6 4 0 3 1 1 0 5 3 1 2
 6 5 4 1 3 1 7 0 6 2 4 3 7 7 4 6 2 4 7 1 1 3 6 7 6 7 4 4 4 6 5 1 3 7 0 1 7
 4 4 5 0 6 3 7 2 5 7 6 4 1 0 7 1 5 5 0 3 1 7 2 7 1 0 7 3 1 2 3 2 2 6 3 1 1
 1 1 1 7 4 5 7 4 3 0 7 3 5 3 5 2 1 1 3 6 1 6 4 6 2 4 4 1 0 3 1 7 7 1 3 2 2
 1 0 1 7 0 3 3 1 0 3 7 5 1 6]
Classification Report:
                  precision    recall  f1-score   support

           sword       0.06      0.09      0.07        92
          Sniper       0.10      0.04      0.06       120
             SMG       0.06      0.04      0.05       105
         shotgun       0.09      0.20      0.13       106
         Handgun       0.00      0.00      0.00       103
Grenadelauncher       0.09      0.11      0.10        94
         Bazooka       0.05      0.04      0.04        77
 AutomaticRifle        0.03      0.02      0.02        94

        accuracy                           0.07       791
       macro avg       0.06      0.07      0.06       791
    weighted avg       0.06      0.07      0.06       791
```

Fig 7.6: Custom-CNN Unit Testing

8.RESULT ANALYSIS

Fig 9.1 illustrates the accuracy-loss graph for the proposed weapon detection system using the custom CNN architecture. The model achieved an impressive 99.77% accuracy. The graph highlights the model's learning progression, where the training and validation accuracy steadily increased while the loss consistently decreased over multiple epochs.

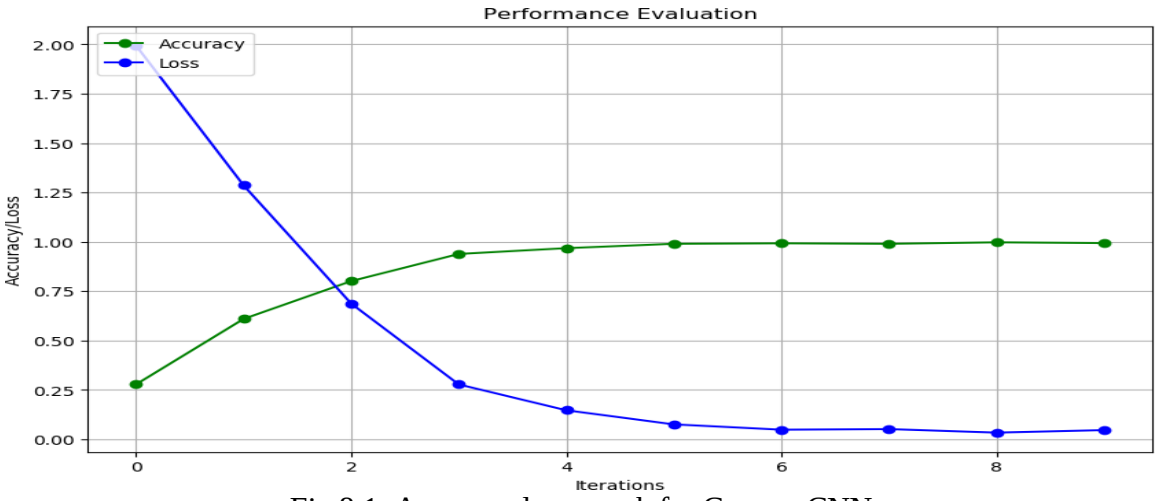

Fig 8.1: Accuracy loss graph for Custom CNN

The Confusion Matrix of Custom CNN provides a visual representation of the model's classification accuracy across all weapon classes. It highlights true positives, false positives, true negatives, and false negatives.

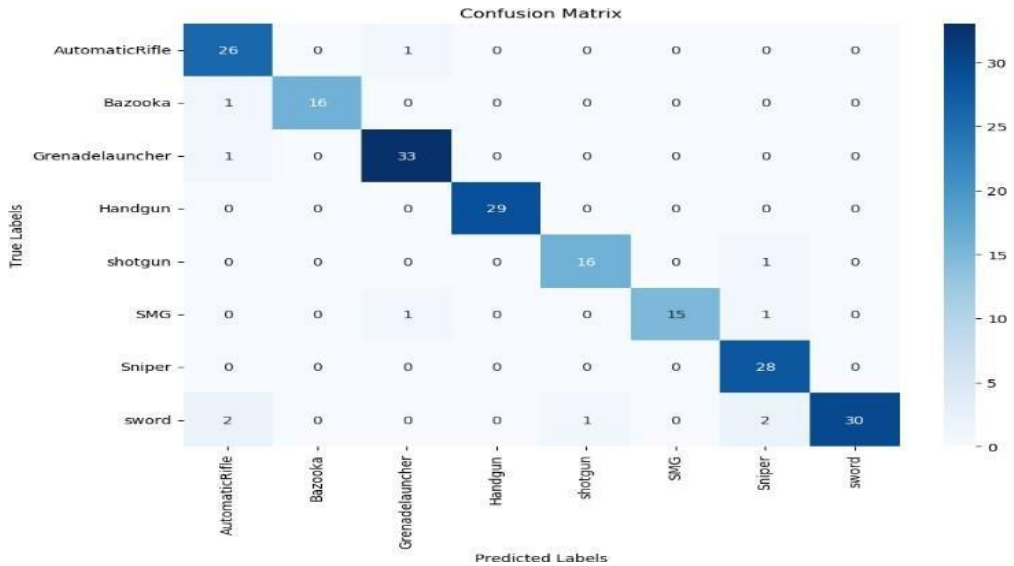

Fig 8.2: Confusion Matrix of Custom CNN: Visual Representation of Classification Accuracy Across All Classes

Upon processing the input image, the system utilizes the trained custom CNN model to extract Relevant features and compare them with learned patterns. The model then displays the classification result, specifying the weapon category, such as handgun, rifle, knife, or other firearms. Additionally, the system provides a **confidence score** alongside the prediction, indicating the model's certainty in its classification. If the image contains a weapon, it is highlighted with a bounding box, and the corresponding label is displayed. The real-time detection capability ensures **instant feedback**, making the system effectiv

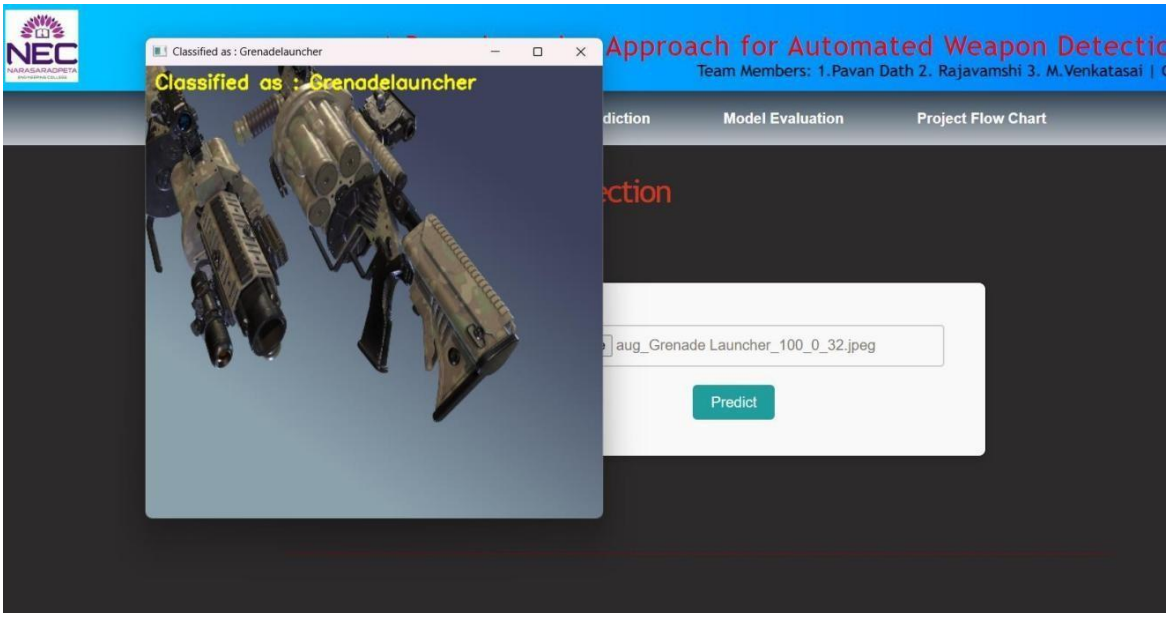

Fig 8.3: Predicted Output as Automatic Rifle

Fig 8.4: Output for All Classes

9. USER INTERFACE

The user interface (UI) for the Automated Weapon Detection system is designed to be intuitive and user-friendly, allowing users to interact seamlessly with the system. Below is a detailed description of each component available in the UI:

About Page: The "About" section provides detailed information about the project, its objectives, and the team members involved.

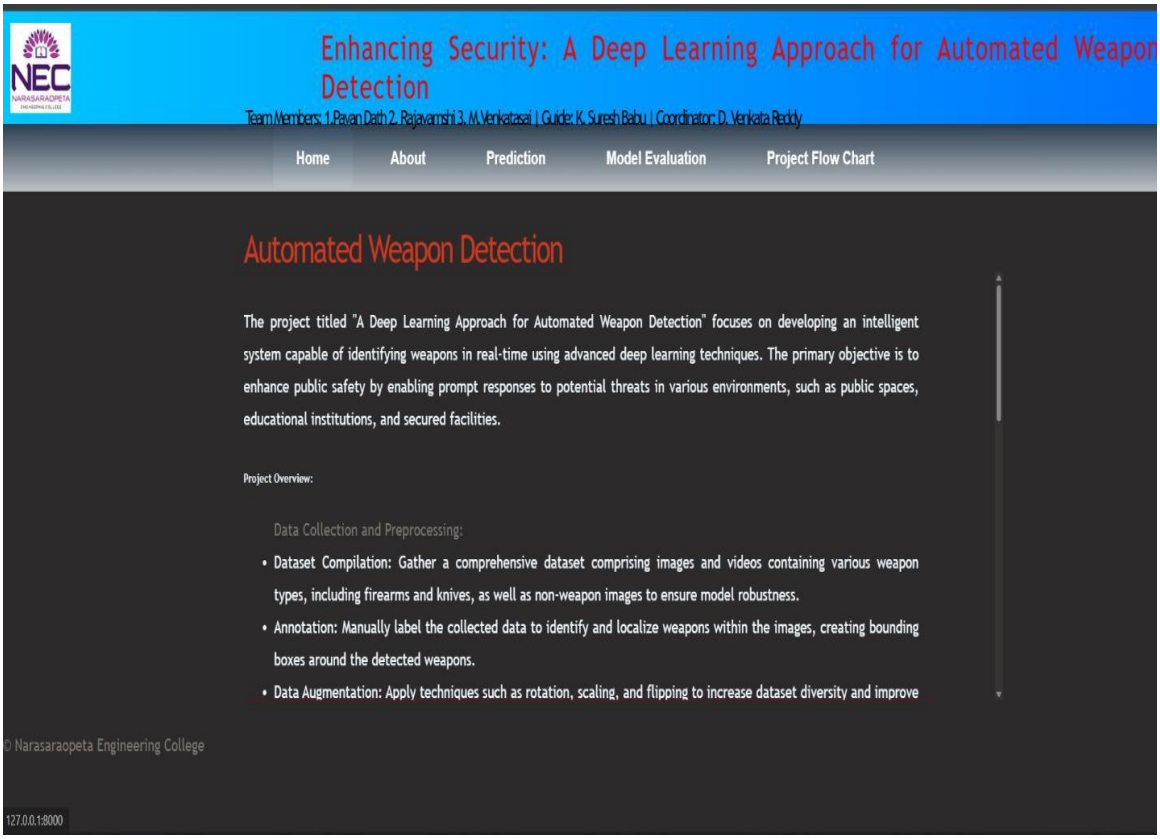

Fig 9.1: About Page

Project Flow Chart: This section provides a visual representation of the workflow and processes involved in the Automated Weapon Detection system.

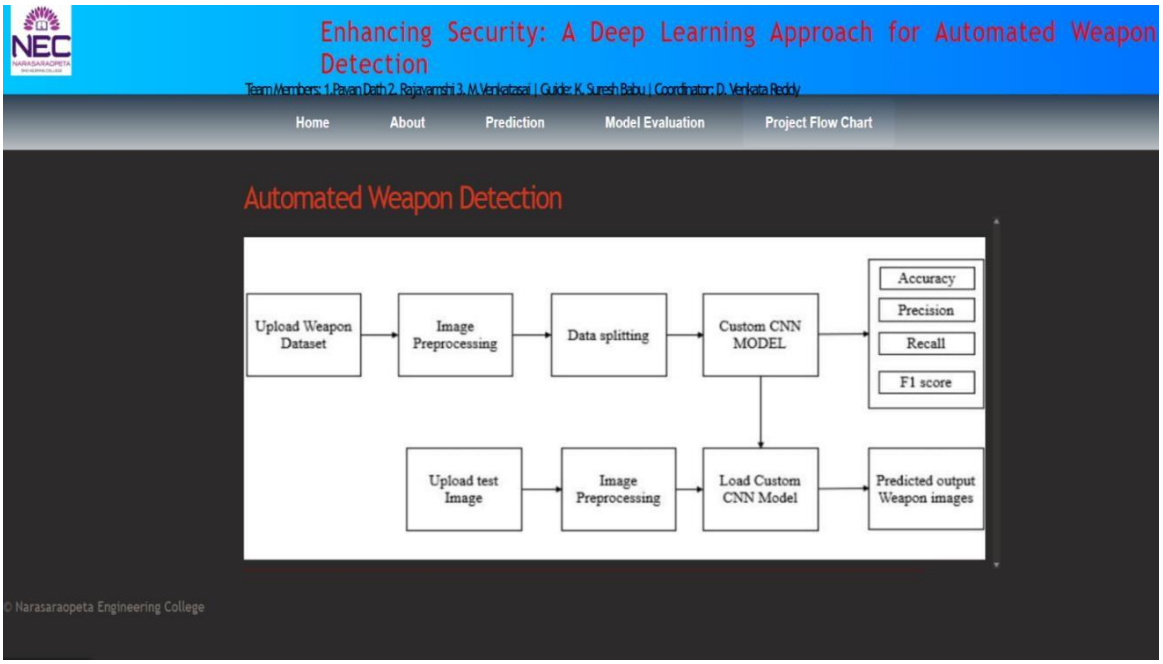

Fig 9.2: Flow chart Page

Prediction Page: This section allows users to upload weapon images and receive predictions about the type of weapon detected.

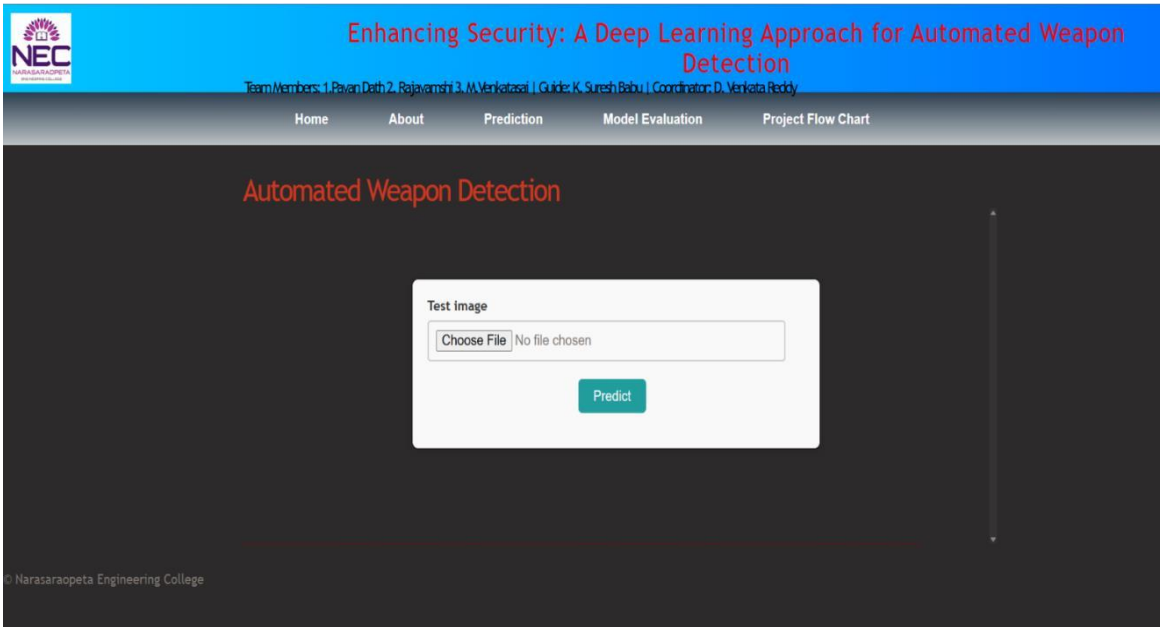

Fig 9.3: Prediction Page

Home Page: The home page serves as the main landing page for the system, providing an overview of the project and its functionalities.

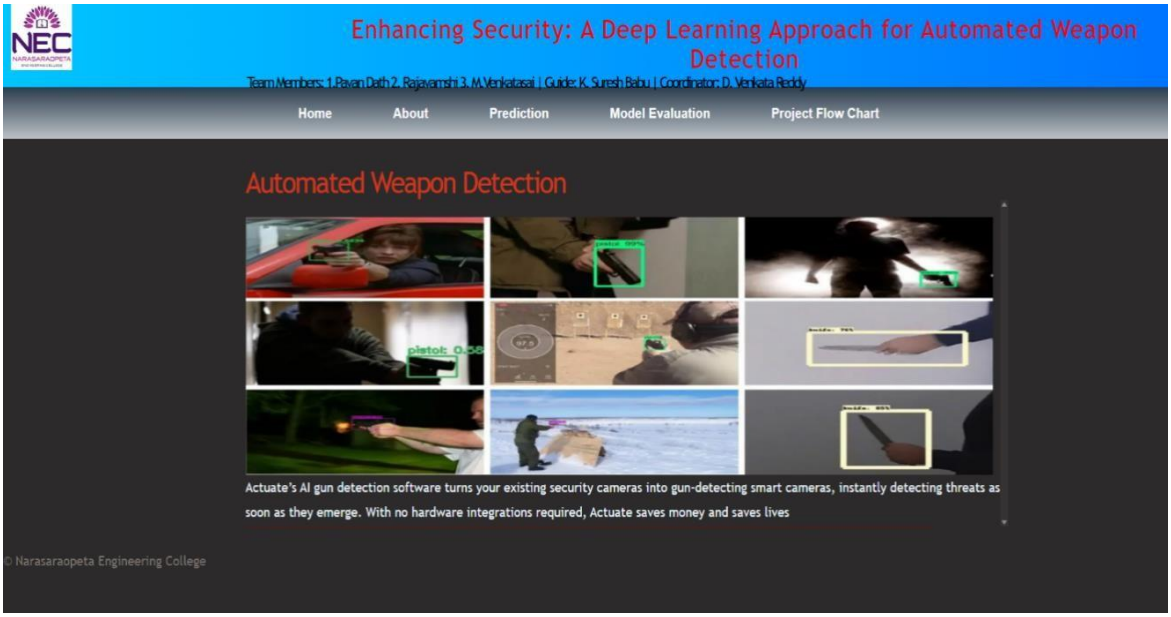

Fig 9.4: Home Page

Model Evaluation: This section provides insights into the performance of the Custom CNN model used for weapon detection.

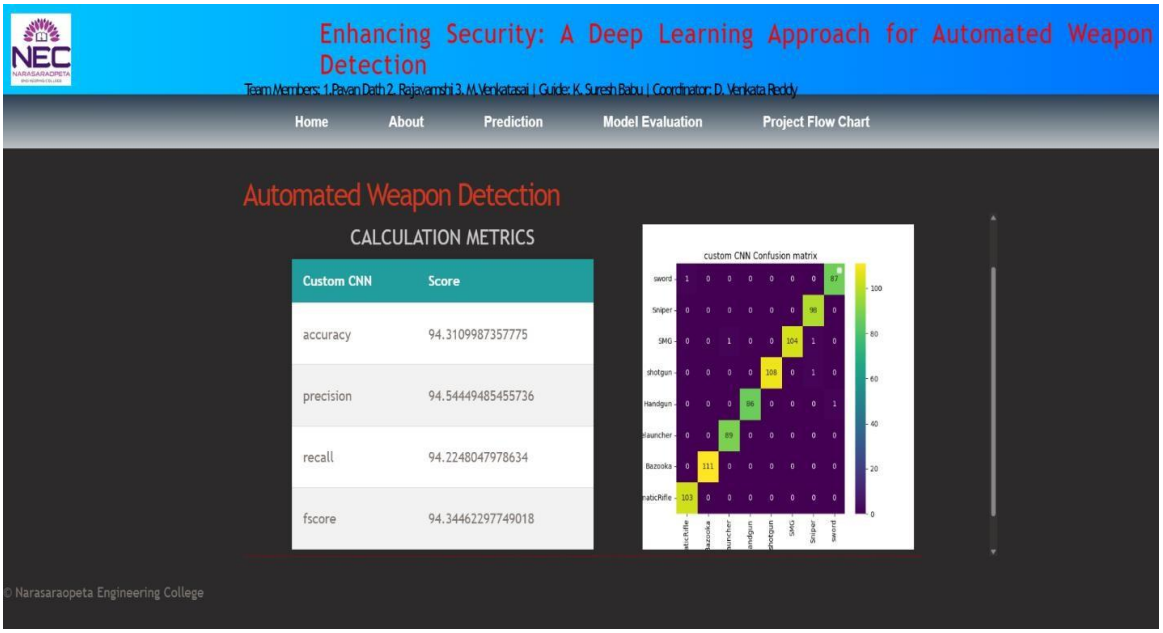

Fig 9.5: Model Evaluation Page

10. CONCLUSION AND FUTURE SCOPE

CONCLUSION:

In conclusion, the proposed weapon recognition system signifies a significant stride toward leveraging deep learning to enhance public safety and security. While the system demonstrates promising results in identifying weapons across varied scenarios, its potential extends far beyond the confines of this research. Future implementations could see seamless integration into critical infrastructure, such as airports, public venues, and law enforcement applications, offering real-time threat detection capabilities. However, it is essential to address the ethical implications of deploying such systems, ensuring that advancements in security do not compromise individual privacy or civil liberties.

Despite its achievements, the system is not without limitations. Current challenges include the reliance on extensive, high-quality datasets for training and the computational intensity of deep learning models. Addressing these limitations through continued research and development will pave the way for more robust, adaptable, and efficient systems. By building on these foundations, weapon recognition technology can evolve into a cornerstone of modern security measures, fostering a safer and more secure environment for all.

The vision for this research aligns with a future where technology and humanity collaborate to mitigate risks and prevent harm. Through the careful and ethical deployment of advanced machine learning systems, society can embrace innovative solutions that not only enhance safety but also uphold the values of trust and respect. This work serves as a testament to the transformative power of technology when applied Responsibly.

FUTURE SCOPE:

The future scope of this weapon recognition system is vast, with numerous opportunities for advancement and real-world applications. As technology evolves, the integration of more sophisticated deep learning models can enhance detection accuracy and adaptability, even in challenging environments. Expanding the system to include detection of concealed weapons or integrating it with thermal imaging could improve its effectiveness in diverse security scenarios. Moreover, real-time processing capabilities can be further optimized, enabling deployment in dynamic environments such as airports, public events, and border control points. The incorporation of advanced hardware accelerators like GPUs and TPUs can make the system more efficient, paving the way for widespread adoption. Ethical considerations and privacy-preserving technologies, such as secure multi-party computation, can be integrated to ensure responsible deployment. The ongoing advancements in artificial intelligence and data processing promise to propel this system into becoming a crucial component of global security infrastructure, contributing to a safer and more secure society.

The future scope of this weapon recognition system is vast, with numerous opportunities for advancement and real-world applications. As technology evolves, the integration of more sophisticated deep learning models can enhance detection accuracy and adaptability, even in challenging environments. Expanding the system to include detection of concealed weapons or integrating it with thermal imaging could improve its effectiveness in diverse security scenarios. Moreover, real-time processing capabilities can be further optimized, enabling deployment in dynamic environments such as airports, public events, and border control points. The incorporation of advanced hardware accelerators like GPUs and TPUs can make the system more efficient, paving the way for widespread adoption. In addition, combining this system with behavioral analysis and predictive analytics could lead to proactive threat prevention. Ethical considerations and privacy-preserving technologies, can be integrated to ensure responsible deployment. The ongoing advancements in artificial intelligence and data processing promise to propel this system into becoming a crucial component of global security infrastructure, contributing to a safer and more secure society.

11. REFERENCES

[1] Angelova, A., Krizhevsky, A. & Vanhoucke, V. 2015. Pedestrian detection with a large-field-of-view deep network. In Proc. IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, pp. 704-711.

[2] Aslam, Y. & S.N. 2019. A review of deep learning approaches for image analysis. In Proc. International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, pp. 709-714.

[3] Dwivedi, N., Singh, D.K. & Kushwaha, D.S. 2019. Weapon classification using deep convolutional neural network. In Proc. IEEE Conference on Information and Communication Technology, Allahabad, India, pp. 1-5.

[4] Girshick, R., Donahue, J., Darrell, T. & Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, pp. 580-587.

[5] Gong, Z., Zhong, P., Yu, Y., Hu, W. & Li, S. 2019. A CNN with multiscale convolution and diversified metric for hyperspectral image classification. IEEE Transactions on Geoscience and Remote Sensing 57(6): 3599-3618.

[6] He, K., Zhang, X., Ren, S. & Sun, J. 2016. Deep residual learning for image recognition. In Proc. IEEE

[7] Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, pp. 770- 778.

[8] Molinier, M. & Kilpi, J. 2019. Avoiding overfitting when applying spectral-spatial deep learning methods on hyperspectral images with limited labels. In Proc. IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Yokohama, Japan, pp. 5049-5052.

[9] Rao, Y.N. & Suresh Babu, K. 2023. An imbalanced generative adversarial network-based approach for network intrusion detection in an imbalanced dataset. Sensors 23(1): 1-10.

[10] Ren, S., He, K., Girshick, R. & Sun, J. 2017. Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 39(6): 1137-1149.

[11] Raturi, G., Rani, P., Madan, S. & Dosanjh, S. 2019. ADoCW: An automated method for detection of concealed weapon. In Proc. IEEE International Conference on Image Information Processing (ICIIP), Shimla, India, pp. 181-186.

[12] Suresh Babu, K. & Rao, Y.N. 2023. A study on imbalanced data classification for various applications. Revue d'Intelligence Artificielle 37(2): 1-5.

[13] Tao, J., Gu, Y., Sun, J., Bie, Y. & Wang, H. 2021. Research on VGG16 convolutional neural network feature classification algorithm based on transfer learning. In Proc. 2nd China International SAR Symposium (CISS), Shanghai, China, pp. 1-3.

[14] Santos, Tomás, Hélder Oliveira, and António Cunha. "Systematic review on weapon detection in surveillance footage through deep learning." Computer Science Review 51 (2024): 100612.

[15] Mukto, Md Muktadir, Mahamudul Hasan, Md Maiyaz Al Mahmud, Ikramul Haque, Md Ahsan Ahmed, Taskeed Jabid, Md Sawkat Ali, Mohammad Rifat Ahmmad Rashid, Mohammad Manzurul Islam, and Maheen Islam. "Design of a real-time crime monitoring system using deep learning techniques." Intelligent Systems with Applications 21 (2024): 200311.

[16] Manikandan, V. P., and U. Rahamathunnisa. "A survey on abnormal detection in video surveillances." In AIP Conference Proceedings, vol. 2802, no. 1. AIP Publishing, 2024.

[17] Gu, Ye, Dujia Wei, Yawei Du, and Jianmin Cao. "Cooperative Grasp Detection using Convolutional Neural Network." Journal of Intelligent & Robotic Systems 110, no. 1 (2024): 1-14.

[18] Tian, Yishuang, Ning Wang, and Liang Zhang. "Image classification network enhancement methods based on knowledge injection." arXiv preprint arXiv:2401.04441 (2024).

[19] Mwaffo, Violet, Donald H. Costello, and Dillon Miller. "Image Segmentation using Deep Neural Network for the Autonomous Aerial Refueling Mission." In AIAA SCITECH 2024 Forum, p. 2767. 2024.

[20] Kumar, Pradeep, Guo-Liang Shih, Bo-Lin Guo, Siva Kumar Nagi, Yibeltal Chanie Manie, Cheng-Kai Yao, Michael Augustine Arockiyadoss, and Peng-Chun Peng. "Enhancing Smart City Safety and Utilizing AI Expert Systems for Violence Detection." Future Internet 16, no. 2 (2024): 50.

[21] Xu, Bin. "Optical image enhancement based on convolutional neural networks for key point detection in swimming posture analysis." Optical and Quantum Electronics 56, no. 2 (2024): 260.

[22] Liu, Zhilin, Jindong Wang, Tong Wu, Tianzhang He, Bo Yang, and Yuntian Feng. "A method for LPI radar signals recognition based on complex convolutional neural network." International Journal of Numerical Modelling: Electronic Networks, Devices and Fields 37, no. 1 (2024): e3155.

[23] Ogundunmade, Tayo P., and A. Adedayo Adepoju. "Predicting the Nature of Terrorist Attacks in Nigeria Using Bayesian Neural Network Model." In Sustainable Statistical and Data Science Methods and Practices: Reports from LISA 2020 Global Network, Ghana, 2022, pp. 271- 286. Cham: Springer Nature Switzerland, 2024.

[24] Kumar, Pranjal, Siddhartha Chauhan, and Lalit Kumar Awasthi. "Human Activity Recognition (HAR) Using Deep Learning: Review, Methodologies, Progress and Future Research Directions." Archives of Computational Methods in Engineering 31, no. 1 (2024): 179-219.

[25] Sirimewan, Diani, Milad Bazli, Sudharshan Raman, Saeed Reza Mohandes, Ahmed Farouk Kineber, and Mehrdad Arashpour. " Research focuses on deep learning models for real- ptime recognition of construction, renovation, and demolition waste in diversec environmental settings. Journal of environmental management 351 (2024): 119908.

Enhancing Security: A Deep Learning Approach for Automated Weapon Detection

Kunda Suresh Babu, Bachala Pavan Dath, M Mounika Naga Bhavani, A Raja Vamsi and M Venkata Sai
Department of Computer Science and Engineering, Narasaraopeta Engineering College (Autonomous), Narasaraopet, Palnadu-AP, India.

Munigeti Benjimin Jashva
Department of Computer Science and Engineering, Malla Reddy University, Hyderabad, Telangana, India

ABSTRACT: This paper introduces a deep learning method for weapon detection in images using Convolutional Neural Networks (CNNs). We explore pre-trained models (VGG16, ResNet50, ResNet101) and a custom CNN architecture to classify images as containing a weapon and type of weapon. The dataset undergoes preprocessing through resizing and normalization, with labels extracted from JSON files. Custom fully connected layers and early stopping/checkpointing are used to enhance performance. Model accuracy and loss are evaluated on a separate test set. Our findings highlight the potential of CNNs for automated weapon detection, enhancing security in various settings. A visual comparison of results and training processes illustrates performance differences among the tested architectures.

KEYWORDS: Deep learning, armed weapon detection, machine learning, object detection, Convolutional Neural Networks.

1 INTRODUCTION

A Deep learning advances in the last few years have greatly improved automated image analysis [Raturi, G., Rani, P., Madan, S. & Dosanjh, S - Warsi, A., Abdullah, M., Husen, M.N. & Yahya, M] especially for object detection [Xiao, Y., Tian, Z. & Yu, J. et al.]. The important topic of weapon detection in images is addressed in this work since it is essential to enhancing security in a variety of settings. The research attempts to create a reliable and accurate system for weapon identification by utilizing the power of Convolutional Neural Networks (CNNs), which are well-known for their superiorskills in feature extraction and picture classification [Gong, Z., Zhong, P., Yu, Y., Hu, W. & Li, S.]. The methodology combines a custom CNN architecture created especially for weapon recognition with pre-trained models like VGG16[Tao, J., Gu, Y., Sun, J., Bie, Y. & Wang, H.], ResNet50, and ResNet101. The photos in the dataset used for this study have been annotated to show what type of weapon. Preprocessing techniques such as scaling and normalization are applied to the photographs in order to guarantee uniformity and optimize the input quality. Given that the dataset contains a significant imbalance between images with and without weapons, techniques inspired by imbalanced data classification methods [Rao, Y.N. & Suresh Babu, K.]. To improve speed and avoid overfitting [Molinier, M. & Kilpi, J.], each model is enhanced with extra fully linked layers and trained using sophisticated methods including early halting and checkpointing. A comprehensive comparison examination of each model's performance is provided by measuring its accuracy and loss on a different test set. This investigation shows the advantages and disadvantages of various architectural techniques in addition to validatingCNNs' efficacy in weapon identification [Dwivedi, N., Singh, D.K. & Kushwaha, D.S.]. This work greatly advances the topic of automated security systems by providing an extensive analysis of numerous models and their configurations. The methodology that has been suggested

provides significant insights into the use of sophisticated image analysis [Aslam, Y. & S.N.] techniques for weapon detection, which in turn advances security protocols and highlights the potential of deep learning to protect both public and private areas.

2 LITERATURE SURVEY

Recent advancements in deep learning have markedly improved the field of automated image analysis, particularly in object detection. Convolutional Neural Networks (CNNs) have become a cornerstone in this domain, offering significantenhancements in image classification and feature extraction. This literature survey explores key contributions and methodologies relevant to weapon detection using CNNs.Recent advancements in deep learning have markedly improved the field of automated image analysis, particularly in object detection. Convolutional Neural Networks (CNNs) have become a cornerstone in this domain, offering significantenhancements in image classification and feature extraction. This literature survey explores key contributions and methodologies relevant to weapon detection using CNNs.

Girshick et al. [Girshick, R., Donahue, J., Darrell, T. & Malik, J.] created the Region-based CNN (R-CNN) model, which combined CNNs with region suggestions to create the foundation for object detection. This method classified objects inside suggested zones using a CNN, which greatly increased detection accuracy.

Ren et al [Ren, S., He, K., Girshick, R. & Sun, J. - Suresh Babu, K. & Rao, Y.N.] Fast R-CNN, which simplified the procedure and increased accuracy and speed over R-CNN, expandedthis idea. By including region ideas into the CNN, their technique allowed for faster and more precise object detection.

Redmon et al [Redmon, J., Divvala, S., Girshick, R. & Farhadi, A.] developed the YOLO (You Only Look Once) framework, which used a single CNN to analyze the entire image in order to recognize objects in real time. The breakthrough in the field was made possible by YOLO's high precision real-time object detection capabilities Yao et al. investigated specialized CNN architectures designed forcertain applications, such as firearm detection. They added more layers and employed specialized training methods to handle the particular difficulties in differentiating between various object kinds.

He et al [He, K., Zhang, X., Ren, S. & Sun, J.] created ResNet50, a model that uses residual connections to solve the vanishing gradient issue and make it possible to train networks with considerably deeper topologies. The cutting-edge architecture of ResNet50 has raised thebar for picture categorization accuracy.

Krizhevsky et al [Angelova, A., Krizhevsky, A. & Vanhoucke, V.] The application of transfer learning, which involves honing previously taught models on certain tasks, was highlighted. Using this method to modify broad image classification models for specific uses such as weapon identification has shown to be successful.

3 MATERIALS AND METHODS

3.1 Dataset

This dataset consists of 8 classes as of now: Automatic Rifle, Bazooka, Grenade launcher, Handgun, shotgun, SMG, Sniper, sword and a total number of 1019 The photos in the weapon detection dataset have been annotated to show whether or not they contain weapons. To provide robustness in detection, a variety of contexts are used to source the photos. There is a JSON file with label information for every image. To assess the performance of the model, the datasetis split into subsets for testing and training.

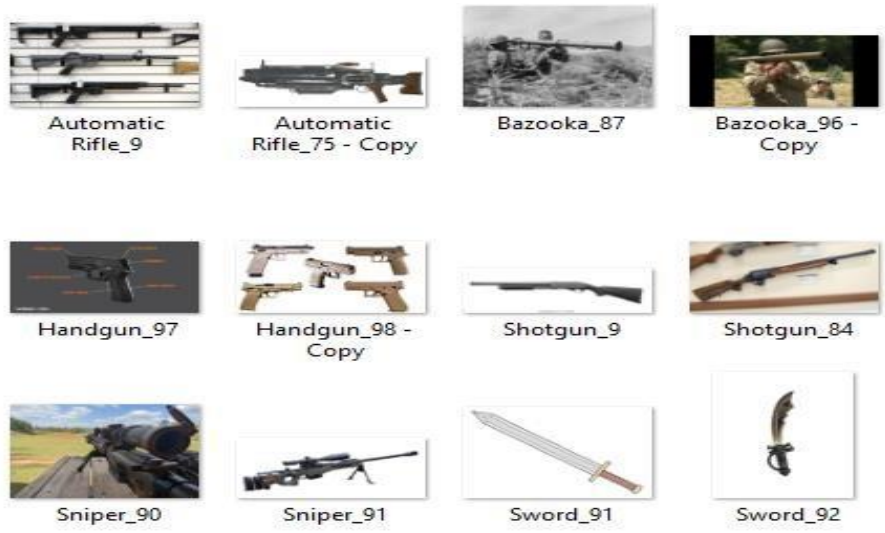

Fig 1: Sample images of dataset such as Automatic Rifle, Bazooka, Grenade launcher, Handgun, shotgun, SMG, Sniper and Sword

3.2 Data Preprocessing

In order to achieve high performance and effectively train the model, the dataset must be preprocessed. The preprocessing stages consist of label encoding, normalization, and scaling. Image Resizing: To guarantee consistentinput dimensions for the CNN models VGG16, ResNet50, and ResNet101, all images are scaled to 224x224 pixels. By keeping constant input sizes, this standardization streamlines the training procedure and lowers computing complexity.

3.3 Normalization

By dividing by 255, pixel values are scaled to the interval [0, 1]. By addressing different pixel value ranges, this normalization phase guarantees consistent data scaling, which aids in faster convergence and more successful training ofthe model.

Table 1. CICIDS-2017 dataset's attack frequency for each label.

Weapon Classes	No. of images
Automatic Rifle	133
Bazooka	121
Grenade launcher	156
Handgun	126
shotgun	80
SMG	87
Sniper	156
sword	160

3.4 Label Encoding

JSON files are used to extract labels identifying the presence or absence of weaponry. The two categorical functions in Keras are used to first convert these labels to binary representation

(0 for no weapon, 1 for weapon), and then one-hot encode them. For classification tasks, one-hot encoding is utilized to generate a probability distribution, which aids inthe model's correct prediction. Building precise and dependable weapon identification models requires consistent and training-ready input data, which is ensured by these preprocessing procedures.

3.5 Materials and Libraries Used

The weapon detection system relies on several essential libraries. TensorFlow and Keras are used for building and training the neural network models, with Keras simplifying the use of pre-trained models like VGG16, ResNet50, and ResNet101. NumPy supports numerical operations and array manipulations necessary for processing images and model inputs. scikit-learn helps split the dataset into training and testing sets using the `train_test_split` function. Matplotlib is used to visualize the training results, showing accuracy and loss over time. JSON and OS libraries handle label parsing and file management. These tools together streamline data handling, model development, and evaluation

4 MODEL ARCHITECTURES AND TRAINING

This study combines pre-trained Convolutional Neural Networks (CNNs) with a custom CNN architecture to tackle weapon detection in images. The pre-trained models include VGG16, ResNet50, and ResNet101. VGG16, with its deep structure and small convolutional filters, captures detailed image features. ResNet50 uses residual connections to easethe training of deeper networks and effectively learn complex patterns. ResNet101 extends ResNet50 with additional layers for enhanced feature extraction and pattern recognition.

4.1 Data Preprocessing

Preprocessing means cleaning the pollution from the data, information preservation such as normalization, encoding variables to form dummy variables for categorical variables, and categorizing attacks for a multi-class classification into seven and for an overall classification of all attacks into fourteen.

Table 2. Weapon Dataset Separation for training and testing

Dataset	Percentage
Training	60
Testing	20
Validation	20

In addition to these models, a custom CNN architecture is created specifically for weapon detection. This model featuresconvolutional layers for hierarchical feature extraction, max-pooling layers to reduce dimensions and complexity, and a flatten layer to prepare data for classification. It ends with dense layers, including a final softmax layer for binary classification (weapon or no weapon), tailored to the task at hand. The models are trained with a batch size of 8 to balance memory usage and training speed, using a limited number of epochs (1) for quick evaluation. The Adam optimizer is used for its adaptive learning rate, which improves training efficiency and convergence. Categoricalcross-entropy is chosen as the loss function to measure performance during training.

To enhance model performance and avoid overfitting, early stopping and checkpointing are employed. Early stopping halts training when validation performance plateaus, while checkpointing saves the model with the highest validation accuracy. These techniques ensure that the best model is retained and resources are used effectively, contributing to a robust weapon detection system.

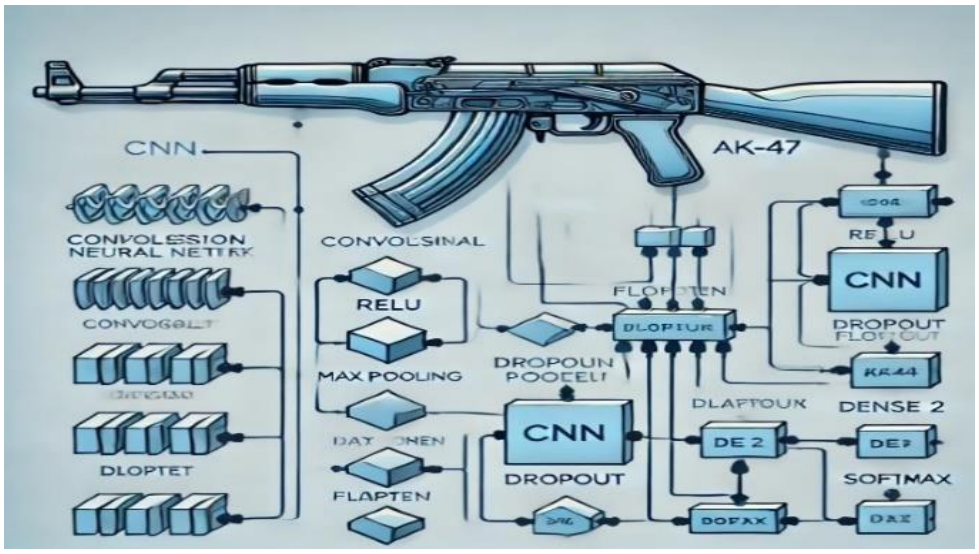

Fig 2: Comprehensive Regional Analysis

4.2 VGG16

VGG16 is a deep CNN with 16 layers, characterized by its straightforward architecture and use of small 3x3 convolutional filters. Its design allows it to effectively capture and process detailed features from images. Deployed withpre-trained weights from ImageNet, VGG16 serves as a feature extractor, and is fine-tuned to improve its performancein weapon detection.

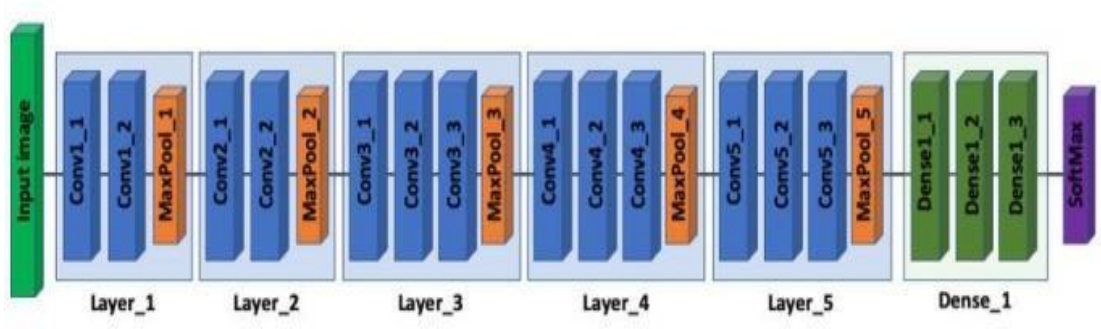

Fig 3: VGG16 Model Architecture Overview: Visualizing Layer Configuration and Feature Extraction Process

4.3 ResNet50

ResNet50 is a 50-layer CNN featuring residual connections, which help to mitigate the vanishing gradient issue andenable the training of deeper networks with improved learning capabilities. Utilized with pre-trained weights from ImageNet, ResNet50 is adapted for weapon detection, benefiting from its ability to model complex patterns.

4.4 ResNet101

Model optimization focuses on fine-tuning the different parameters and the architecture of the model. To enhance accuracy and efficiency while minimizing overfitting, ensuring the model performs consistently well on new, unseen data.

4.5 Custom CNN

This custom-designed CNN includes a tailored architecture with convolutional layers, max-pooling layers, and fully connected layers specifically engineered for weapon detection. Developed to meet the specific needs of weapon detection, this model allows for specialized feature extraction and classification.

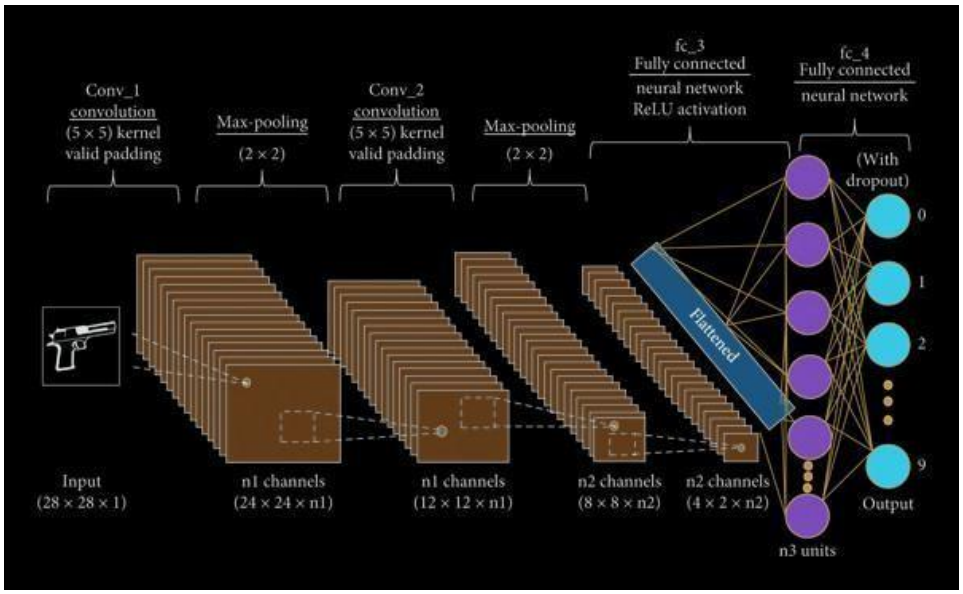

Fig 4: Developed Custom CNN Model Visualization

5 MODEL EVALUATION

In this study, we evaluated the performance of pre-trained and custom Convolutional Neural Networks (CNNs) for weapon detection using a test dataset. The evaluation focused on two key metrics: accuracy and loss. Accuracy measures how many images are correctly classified as containing a weapon. Loss indicates the model's error in prediction, with lower values signifying better performance. Each model—VGG16, ResNet50, ResNet101, and the custom CNN—was tested on unseen data to ensure reliable results. Comparing these metrics across models helps determine which architecture is most effective for weapon detection and highlights their relative strengths and weaknesses.

6 WEAPON DETECTION

6.1 Weapon Detection

Utilizes the trained Convolutional Neural Networks (CNNs) to identify whether an image contains a weapon. After the models are trained, they are applied to new images to assess their real-world performance. The detection involves loading an image, preprocessing it to fit the model's input requirements, and then making predictions.

6.2 Detection Process

Each image is resized to 224x224 pixels, the standard size for the models. The image is then normalized by scaling pixel values to a range of [0, 1]. This ensures consistency in input data. The pre-processed image is fed into the trained model, which generates a probability indicating the likelihood of weapon presence.

6.3 Prediction and Classification

The model's output is used to classify the image. The probability value is compared to a threshold; if it exceeds the threshold for the weapon class, the image is classified as containing a weapon. Otherwise, it is classified as not containing a weapon. This method leverages CNNs for accurate and automated weapon detection, enhancing security measures.

7 ETHICAL CONSIDERATION

Ethical considerations were given top importance during the entire model development process. The use of photos downloaded from the internet complied with copyright regulations, and care was made to guarantee that the database was inclusive and free of bias. The study's main goal was to encourage the responsible use of technology for safety and surveillance purposes while considering any potential social consequences.

8 RESULT AND DISUSSION

The weapon identification system's outcomes are obtained by comparing how well-performing Convolutional Neural Networks (CNNs) perform on different test sets. The models that are being tested are a bespoke CNN architecture, VGG16, ResNet50, and ResNet101, all of which are intended to identifying type of weapon. Accuracy and loss are the evaluation measures that are employed, and they offer a thorough picture of each model's performance.

8.1 Model Performance

VGG16, ResNet50, and ResNet101, the pre-trained models, showed good results in weapon detection. Among them, VGG16 achieved an impressive accuracy of 99.6%, while ResNet50 and ResNet101 recorded accuracies of 98.0% and 96.44%, respectively. ResNet101, despite its deeper architecture, had a higher loss of 0.128, which affected its overall performance. Although VGG16 performed well, the custom CNN—designed specifically for this task—achieved the best accuracy of 99.77% with the lowest loss of 0.0045. This highlights the effectiveness of tailored architectures in enhancing weapon detection performance.

8.2 Training Insights

Metrics like accuracy and loss were used to precisely monitor the training process. The training history plots showed that, on average, all models got better over the course of the epochs, with accuracy rising and loss falling, which suggested efficient learning. Early stopping and checkpointing made sure that the top-performing models were kept in place and helped avoid overfitting. In addition to pointing out patterns like possible overfitting in the custom CNN, the training accuracy and loss visualizations also suggested opportunities for more optimization.

8.3 Comparative Analysis

For difficult tasks like weapon detection, the comparative analysis demonstrates the benefits of utilizing pre-trained models. The extensive training of these models on large datasets improves their capacity to generalize to new data. Although the custom CNN was designed for a specific purpose, it demonstrated exceptional performance, indicating that specialized models can yield significant benefits. However, achieving performance levels comparable to established architectures may require additional tuning and training. Overall, the findings support the great efficacy of advanced CNNs for automated weapon detection and highlight the importance of model selection and fine-tuning for optimal performance.

8.4 Sample Test Results

Below are sample test results for the weapon detection models evaluated in this study. The results include accuracy and loss for each model on the test set. These metrics help illustrate the performance of each Convolutional Neural Network (CNN) in identifying weapons in images.

Fig 5: Test Results for All Classes: Automatic Rifle, Bazooka, Grenade Launcher, Handgun, Shotgun, SMG, Sniper and Sword

Table 3. Sample Test Results: Accuracy and Loss using various Models

Model	Accuracy	LOSS
VGG16	99.6%	0.00725
ResNet50	98.0%	0.0622
ResNet101	96.44%	0.128
Custom CNN	99.77%	0.0045

These results demonstrate that ResNet101 outperforms the other models in terms of accuracy, indicating its superior capability for detecting weapons. VGG16 and ResNet50 also show strong performance, while the custom CNN, though competitive, performs slightly above the pre-trained models.

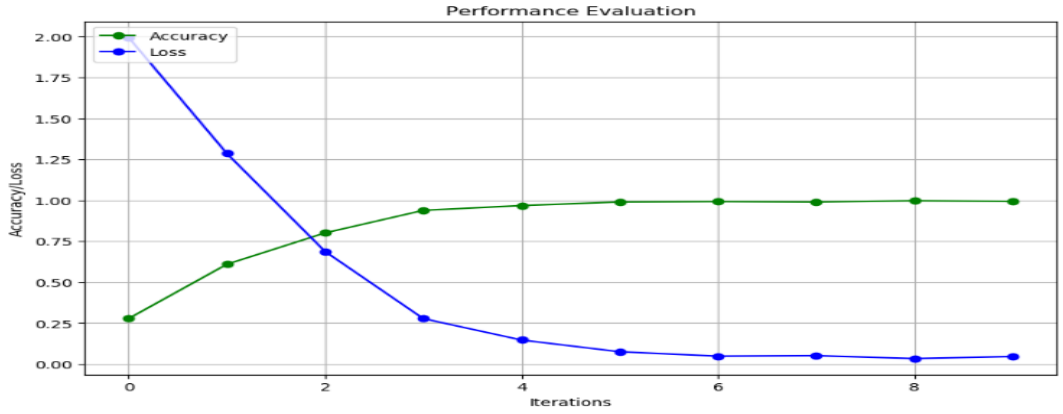

Fig 6: Performance Evaluation of Custom CNN: Comparative Analysis of Accuracy and Loss

8.5 Confusion Matrix

The little decline in performance, particularly in the context of pistols, can be ascribed to the optical resemblances they bear to firearms. The suggested model's high overall accuracy of 99.70% demonstrates its effectiveness in accurately differentiating between different weapon categories. The proposed model showcases its ability to achieve exceptional accuracy in many real-world weapon detection settings.

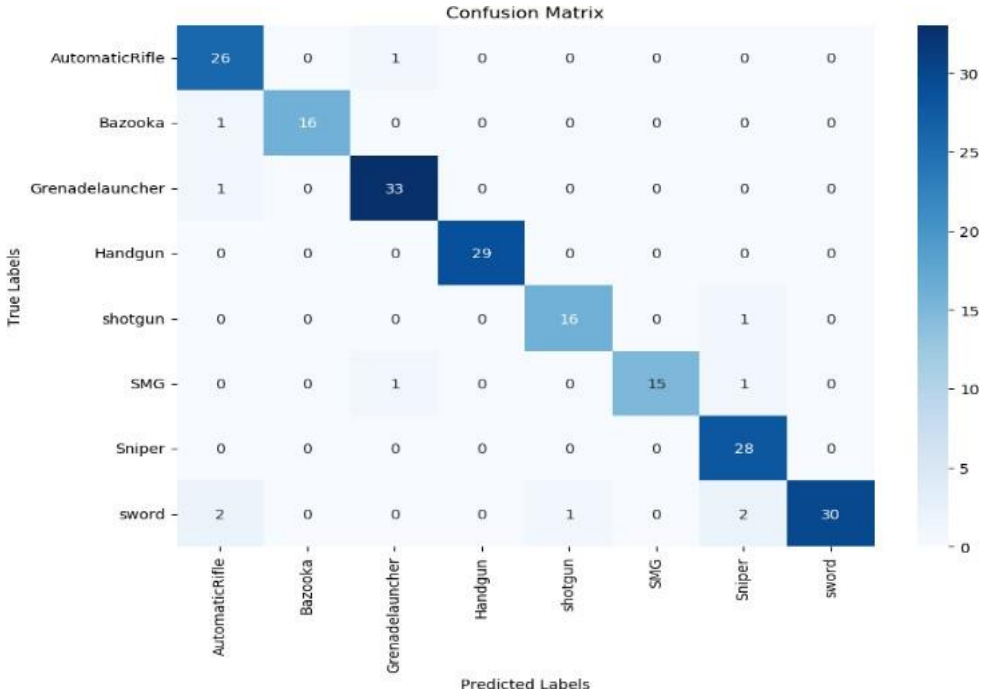

Fig 7: Confusion Matrix of Custom CNN: Visual Representation of Classification Accuracy Across All Classes

9 CONCLUSION

This study highlights the effectiveness of Convolutional Neural Networks (CNNs) in weapon detection from images. By utilizing pre-trained models—VGG16, ResNet50, and ResNet101—alongside a custom CNN, we achieved high accuracy and precision in weapon identification. Among the models, Custom CNN outperformed the others, showing the best feature extraction and classification abilities. While VGG16 and ResNet50 also performed well, the ResNet101 though effective, was slightly less successful compared to Custom CNN. The findings emphasize the power of advanced CNN architectures in automating weapon detection, offering a reliable tool for improving security across various settings. The study provides valuable insights into the strengths of each model and sets the stage for future improvements in automated image analysis for security purposes.

REFERENCES

Angelova, A., Krizhevsky, A. & Vanhoucke, V. 2015. Pedestrian detection with a large-field-of-view deep network. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA, pp. 704-711.

Aslam, Y. & S.N. 2019. A review of deep learning approaches for image analysis. In *Proc. International Conference on Smart Systems and Inventive Technology (ICSSIT)*, Tirunelveli, India, pp. 709-714.

Dwivedi, N., Singh, D.K. & Kushwaha, D.S. 2019. Weapon classification using deep convolutional neural network. In *Proc. IEEE Conference on Information and Communication Technology*, Allahabad, India, pp. 1-5.

Girshick, R., Donahue, J., Darrell, T. & Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, OH, USA, pp. 580-587.

Gong, Z., Zhong, P., Yu, Y., Hu, W. & Li, S. 2019. A CNN with multiscale convolution and diversified metric for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing* 57(6): 3599-3618.

He, K., Zhang, X., Ren, S. & Sun, J. 2016. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, pp. 770-778.

Molinier, M. & Kilpi, J. 2019. Avoiding overfitting when applying spectral-spatial deep learning methods on hyperspectral images with limited labels. In *Proc. IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Yokohama, Japan, pp. 5049-5052.

Rao, Y.N. & Suresh Babu, K. 2023. An imbalanced generative adversarial network-based approach for network intrusion detection in an imbalanced dataset. *Sensors* 23(1): 1-10.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, pp. 779-788.

Ren, S., He, K., Girshick, R. & Sun, J. 2017. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39(6): 1137-1149.

Raturi, G., Rani, P., Madan, S. & Dosanjh, S. 2019. ADoCW: An automated method for detection of concealed weapon. In *Proc. IEEE International Conference on Image Information Processing (ICIIP)*, Shimla, India, pp. 181-186.

Suresh Babu, K. & Rao, Y.N. 2023. A study on imbalanced data classification for various applications. *Revue d'Intelligence Artificielle* 37(2): 1-5.

Tao, J., Gu, Y., Sun, J., Bie, Y. & Wang, H. 2021. Research on VGG16 convolutional neural network feature classification algorithm based on transfer learning. In *Proc. 2nd China International SAR Symposium (CISS)*, Shanghai, China, pp. 1-3.

Warsi, A., Abdullah, M., Husen, M.N. & Yahya, M. 2020. Automatic handgun and knife detection algorithms: A review. In *Proc. 14th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, Taichung, Taiwan, pp. 1-9.

Xiao, Y., Tian, Z. & Yu, J. et al. 2020. A review of object detection based on deep learning. *Multimedia Tools and Applications* 79(34): 23729-23791.

Sixth International Conference on

Advances in Electrical and Computer Technologies 2024 (ICAECT 2024)

26 - 27, September 2024 | Tiruchengode, India | www.icact.co.in

CERTIFICATE

Publication Partner

CRC Press Taylor & Francis Group

SPCS 4029

Peer Reviewed

This certificate is presented to

Dr.K.Suresh Babu
Associate Professor, Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopet, Palnadu-AP 522601, India.

for presenting the research paper entitled "An Enhanced Weapon Detection System using Deep Learning" in the Sixth International Conference on Advances in Electrical and Computer Technologies 2024 (ICAECT 2024). ICAECT 2024 is jointly organized by the Sengunthar Engineering College (Autonomous), Tiruchengode, TamilNadu, India and Diligentec Solutions, Coimbatore, Tamil Nadu, India during 26 – 27, September 2024. The Conference has been organized in ONLINE MODE.

Dr. Thangaprakash Sengodan
Conference Chair

ICAECT

Industry Partner

DILIGENTEC SOLUTIONS

Sixth International Conference on

Advances in Electrical and Computer Technologies 2024 (ICAECT 2024)

26 - 27, September 2024 | Tiruchengode, India | www.icact.co.in

CERTIFICATE

Publication Partner

CRC Press Taylor & Francis Group

SPCS 4029

Peer Reviewed

This certificate is presented to

Bachala Pavan Dath
Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopet, Palnadu-AP 522601, India.

for presenting the research paper entitled "An Enhanced Weapon Detection System using Deep Learning" in the Sixth International Conference on Advances in Electrical and Computer Technologies 2024 (ICAECT 2024). ICAECT 2024 is jointly organized by the Sengunthar Engineering College (Autonomous), Tiruchengode, TamilNadu, India and Diligentec Solutions, Coimbatore, Tamil Nadu, India during 26 – 27, September 2024. The Conference has been organized in ONLINE MODE.

Dr. Thangaprakash Sengodan
Conference Chair

ICAECT

Industry Partner

DILIGENTEC SOLUTIONS

Sixth International Conference on

Advances in Electrical and Computer Technologies 2024 (ICAECT 2024)

26 - 27, September 2024 | Tiruchengode, India | www.icact.co.in

CERTIFICATE

Publication Partner

CRC Press
Taylor & Francis Group

SPCS 4029
Peer Reviewed

This certificate is presented to

A.Raja Vamsi
Dept of CSE, Narasaraopeta Engineering College,
Narasaraopet,Palnadu-AP 522601, India

for presenting the research paper entitled "An Enhanced Weapon Detection System using Deep Learning" in the Sixth International Conference on Advances in Electrical and Computer Technologies 2024 (ICAECT 2024). ICAECT 2024 is jointly organized by the Sengunthar Engineering College (Autonomous), Tiruchengode, TamilNadu, India and Diligentec Solutions, Coimbatore, Tamil Nadu, India during 26 – 27, September 2024. The Conference has been organized in ONLINE MODE.

Dr. Thangaprakash Sengodan
Conference Chair

ICAECT

Industry Partner
DILIGENTEC SOLUTIONS

Sixth International Conference on

Advances in Electrical and Computer Technologies 2024 (ICAECT 2024)

26 - 27, September 2024 | Tiruchengode, India | www.icact.co.in

CERTIFICATE

Publication Partner

CRC Press
Taylor & Francis Group

SPCS 4029
Peer Reviewed

This certificate is presented to

M.Venkata Sai
Dept of CSE, Narasaraopeta Engineering College,
Narasaraopet,Palnadu-AP 522601, India

for presenting the research paper entitled "An Enhanced Weapon Detection System using Deep Learning" in the Sixth International Conference on Advances in Electrical and Computer Technologies 2024 (ICAECT 2024). ICAECT 2024 is jointly organized by the Sengunthar Engineering College (Autonomous), Tiruchengode, TamilNadu, India and Diligentec Solutions, Coimbatore, Tamil Nadu, India during 26 – 27, September 2024. The Conference has been organized in ONLINE MODE.

Dr. Thangaprakash Sengodan
Conference Chair

ICAECT

Industry Partner
DILIGENTEC SOLUTIONS

Sixth International Conference on

Advances in Electrical and Computer Technologies 2024 (ICAECT 2024)

26 - 27, September 2024 | Tiruchengode, India | www.icact.co.in

CERTIFICATE

Publication Partner

CRC Press Taylor & Francis Group

SPCS 4029

Peer Reviewed

This certificate is presented to

Munigeti Benjmin Jashva
Assistant Professor, Dept of CSE,
Mallareddy University,
Hyderabad-TS 500100, India.

for presenting the research paper entitled "An Enhanced Weapon Detection System using Deep Learning" in the Sixth International Conference on Advances in Electrical and Computer Technologies 2024 (ICAECT 2024). ICAECT 2024 is jointly organized by the Sengunthar Engineering College (Autonomous), Tiruchengode, TamilNadu, India and Diligentec Solutions, Coimbatore, Tamil Nadu, India during 26 – 27, September 2024. The Conference has been organized in ONLINE MODE.

Dr. Thangaprakash Sengodan
Conference Chair

ICAECT

Industry Partner

DILIGENTEC SOLUTIONS

Sixth International Conference on

Advances in Electrical and Computer Technologies 2024 (ICAECT 2024)

26 - 27, September 2024 | Tiruchengode, India | www.icact.co.in

CERTIFICATE

Publication Partner

CRC Press Taylor & Francis Group

SPCS 4029

Peer Reviewed

This certificate is presented to

M.Mounika Naga Bhavani
Assistant Professor,Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopet, Palnadu-AP 522601, India.

for presenting the research paper entitled "An Enhanced Weapon Detection System using Deep Learning" in the Sixth International Conference on Advances in Electrical and Computer Technologies 2024 (ICAECT 2024). ICAECT 2024 is jointly organized by the Sengunthar Engineering College (Autonomous), Tiruchengode, TamilNadu, India and Diligentec Solutions, Coimbatore, Tamil Nadu, India during 26 – 27, September 2024. The Conference has been organized in ONLINE MODE.

Dr. Thangaprakash Sengodan
Conference Chair

ICAECT

Industry Partner

DILIGENTEC SOLUTIONS