Deep Learning Framework For Early Fire and Smoke 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

K. Ramya	**(21471A05N0)**
K. Supriya	**(22475A0519)**
CH. Jyothika	**(21471A05M2)**
K. Vydurya	**(18471A05K3)**

Under the esteemed guidance of

Dr. Sireesha Moturi, M.Tech., Ph.D.,

Associate Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASAROPET

(AUTONOMOUS)

Accredited by NAAC with A+ Grade and NBA under Tier -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, YALAMANDA VILLAGE, NARASARAOPET- 522601

2024-2025

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name **"Deep Learning Framework For Early Fire and Smoke Detection"** is a bonafide work done by the team **K.Ramya (21471A05N0), K.Supriya (22475A0519), CH.Jyothika (21471A05M2), K.Vydurya (18471A05K3)** 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. Sireesha Moturi,M.Tech., Ph.D.,
Associate Professor

PROJECT CO-ORDINATOR

Dodda Venkata Reddy,M.Tech.,(ph.D)
Assistant professor

HEAD OFTHE DEPARTMENT

Dr. S. N. Tirumala Rao, M.Tech., Ph.D.,
Professor& HOD

EXTERNAL EXAMINER

DECLARATION

We declare that this project work titled **"DEEP LEARNING FRAMEWORK FOR EARLY FIRE AND SMOKE DETECTION "** is composed by ourselves that the work contain here is our 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.

Kata.Ramya(21471A05N0)
Kota.Supriya(22475A0519)
Chennupati.Jyothika(21471A05M2)
Kondabathini.Vydurya(18471A05K3)

ACKNOWLEDGEMENT

We wish to express my 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,** **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. Sireesha Moturi,** **M.Tech.**, **Ph.D.**, whose valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

We extend our sincere thanks towards **Dodda Venkata Reddy,** **M.Tech.,(ph.D)** Assistant professor & Project coordinator of the project for extending his encouragement. His 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

Kata.Ramya(21471A05N0)
Kota.Supriya(22475A0519)
Chennupati.Jyothika(21471A05M2)
Kondabathini.Vydurya(18471A05K3)

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 center 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 expertize 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 analyse 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.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Project Course Outcomes (CO'S):

CO421.1: Analyze the System of Examinations and identify the problem.

CO421.2: Identify and classify the requirements.

CO421.3: Review the Related Literature

CO421.4: Design and Modularize the project

CO421.5: Construct, Integrate, Test and Implement the Project.

CO421.6: Prepare the project Documentation and present the Report using appropriate method.

Course Outcomes – Program Outcomes mapping

	PO1	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:

- Low level
- Medium level
- High level

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

Name of the course from which principles are applie in this project	**Description of the device**	**Attained PO**
C2204.2, C22L3.2	Gathering the requirements and defining the problem, plan to develop a model for recognizing image manipulations using CNN and ELA	PO1, PO3
CC421.1, C2204.3, C22L3.2	Each and every requirement i critically analyzed, the process mode 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 phaseofthe 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 social media users and in future updates in our project can be done based on detection of forged videos	PO4, PO7
C32SC4.3	The physical design includes website to check whether an image is real or fake	PO5, PO6

ABSTRACT

Forest fires pose significant threats to ecological balance and human life, leading to severe environmental degradation, loss of biodiversity, and economic setbacks. The rapid spread of wildfires necessitates early detection to minimize destruction and enhance firefighting response strategies. Traditional smoke detection techniques, which primarily rely on sensor-based systems and satellite imaging, often suffer from delayed detection, limiting their effectiveness in preventing large-scale damage. This study explores the application of deep learning techniques, specifically pre-trained convolutional neural networks (CNNs), for real-time fire and smoke detection. Three state-of-the-art CNN architectures—VGG16, InceptionV3, and Xception—are employed to analyze fire-related visual data. The models are trained and tested on diverse datasets obtained from Kaggle, ensuring a comprehensive representation of real-world wildfire scenarios, including variations in lighting, smoke density, and environmental conditions.

Experimental evaluations indicate significant improvements in detection accuracy using CNN-based approaches. Among the tested models, InceptionV3 demonstrates superior performance, achieving 94% accuracy during feature extraction and maintaining 89% accuracy after fine-tuning. These results underscore the effectiveness of deep learning models in recognizing fire and smoke patterns with high precision. Furthermore, the real- time processing capability of CNNs enhances early response mechanisms, enabling rapid intervention and potentially mitigating the devastating consequences of forest fires.

INDEX

S.NO.	LIST OF FIGURES	PAGE NO

1. INTRODUCTION

Wildfires are a major environmental crisis, causing severe destruction to ecosystems, biodiversity, human life, and property. The frequency and intensity of wildfires have increased significantly in recent years due to climate change, deforestation, and human activities. Regions such as the United States, Australia, Indonesia, and India have experienced devastating wildfire incidents, underscoring the urgent need for robust fire detection and monitoring systems. In addition to immediate destruction, wildfires contribute substantially to carbon emissions, worsening global climate conditions [6].

Traditional wildfire detection methods, including satellite-based thermal imaging, sensor networks, and manual monitoring, often suffer from limitations such as high latency, false alarms, and reduced accuracy under challenging conditions like dense smoke or small-scale fires [9]. Moreover, the infrastructure costs associated with these methods make them impractical for large-scale deployment in remote or inaccessible regions [10].

Recent advancements in Deep Learning, particularly Convolutional Neural Networks (CNNs), have demonstrated significant potential in automating fire detection through image-based approaches. CNNs have proven effective in distinguishing fire and smoke patterns by extracting complex visual features, outperforming traditional rule-based or sensor-driven methods [7].

Pre-trained Deep Learning models such as VGG16, InceptionV3, and Xception leverage transfer learning, enabling them to utilize knowledge from large-scale datasets for improved wildfire detection while reducing computational overhead [8].

In this research, we explore Deep Learning-based approaches for early wildfire detection using datasets obtained from Kaggle. We employ CNN architectures, including VGG16, InceptionV3, and Xception, to classify fire and smoke images with high accuracy. Our methodology enhances detection speed and reliability, addressing common challenges such

as false positives and variable environmental conditions. Unlike static sensor-based approaches, our system is designed to provide real-time fire monitoring, improving early response capabilities and reducing wildfire-related losses [12].

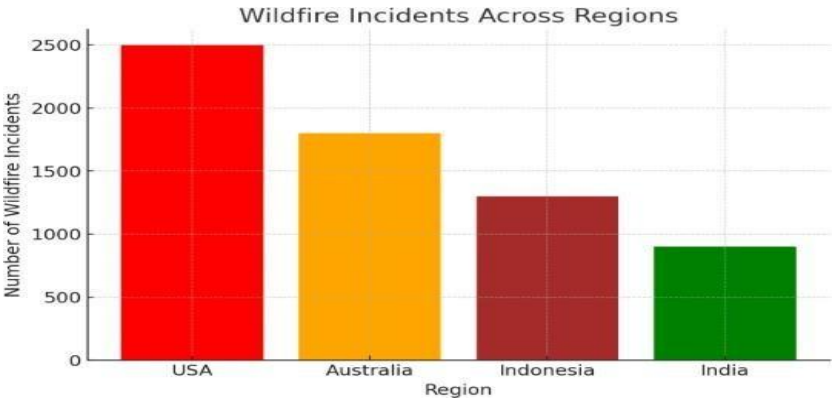

Figure 1.1: Wildfire Incidents Across Regions

Wildfires have increased significantly in recent years, causing severe environmental damage. **Fig 1.1** shows the frequency of wildfire incidents in major affected regions.

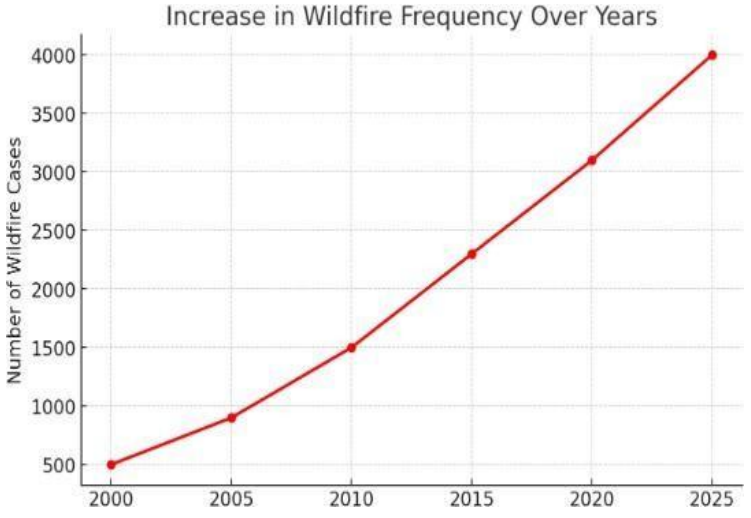

Figure 1.2: Increase in Wildfire Frequency Over the Years

Deep Learning models like CNNs improve fire detection accuracy. **Fig 1.2** compares wildfire occurrences over time, highlighting the rising trend.

Furthermore, integrating Deep Learning with drone and satellite imagery can enhance the effectiveness of wildfire detection systems. Drones equipped with infrared and RGB cameras can capture real-time data, feeding it into CNN models for immediate analysis and alert generation [6]. Such systems can complement existing methods by providing high-resolution images and continuous monitoring, especially in remote or high-risk areas. Additionally, multimodal data fusion, combining satellite images, weather data, and ground-based sensor information, can further improve detection accuracy and predictive capabilities [10].

Apart from image-based detection, recent studies have explored the use of sensor fusion techniques that integrate multiple sources of data, such as thermal imaging, LiDAR, and atmospheric gas sensors. These multimodal approaches enhance early detection accuracy by analyzing various environmental factors in real-time [11]. Such hybrid detection systems have shown promise in identifying wildfires at their early stages, reducing response time and preventing large-scale devastation.

Several real-world applications of Deep Learning-based wildfire detection systems have demonstrated significant improvements in early warning capabilities. For instance, recent deployments of drone- based fire monitoring in California have enhanced real-time detection and response efforts [13].

Similarly, AI-driven fire prediction models have been used in Australia to assess risk factors and proactively allocate firefighting resources [14]. These case studies highlight the potential of integrating AI and deep learning into existing wildfire management frameworks.

Despite the promising advancements, deep learning-based wildfire detection faces several challenges. Computational complexity remains a significant barrier, as high-resolution image processing requires substantial GPU resources. Additionally, data quality is a critical issue, as deep learning models rely heavily on diverse and well-annotated datasets for accurate classification. Environmental factors such as fog, cloud cover, and sensor limitations can also affect the performance of fire detection models [9]. Addressing these challenges requires further research into optimizing neural network architectures and developing more efficient data processing techniques.

Looking ahead, emerging technologies such as quantum computing and edge AI offer new possibilities for wildfire detection. Quantum computing could enhance the processing capabilities of deep learning models, allowing for faster and more complex simulations of fire spread patterns. Edge AI, which involves processing data directly on devices like drones and IoT sensors, can reduce latency and improve real-time decision-making capabilities [8]. These innovations could revolutionize wildfire detection and response strategies in the coming years.

Different deep learning models offer unique advantages and trade-offs for wildfire detection. CNN-based models such as VGG16 and ResNet provide high accuracy but require significant computational power. Transformer-based models, including Vision Transformers (ViTs), have shown potential in image recognition tasks and could be explored further for fire detection [7]. Hybrid approaches that combine CNNs with recurrent neural networks (RNNs) may improve fire spread prediction by incorporating temporal data [10]. Evaluating and comparing these models can help identify the most effective approach for real-world applications.

By addressing the limitations of traditional wildfire detection systems and leveraging the power of Deep Learning, our research aims to contribute to the development of an efficient and scalable solution for wildfire monitoring and early warning. This study emphasizes the importance of real-time processing, high accuracy, and adaptability to various environmental conditions to mitigate the devastating impact of wildfires worldwide.

Through the integration of cutting-edge technologies, policy-driven strategies, and community-based initiatives, a comprehensive approach to wildfire management can be achieved, reducing environmental and socio-economic losses associated with these catastrophic events.

According to the National Interagency Fire Center (NIFC), an average of 7.5 million acres of land burns each year due to wildfires in the United States alone. In 2020, the U.S. experienced one of the worst wildfire seasons in history, with over 10 million acres affected [15]. Similarly, Australia's 2019-2020 bushfire season, commonly referred to as "Black Summer," resulted in the burning of over 18.6 million hectares and the loss of nearly three billion animals [16].

A study conducted by the European Space Agency (ESA) found that global wildfire occurrences have increased by 13% over the past two decades due to climate change and human activities [17]. Additionally, reports from NASA's Earth Observatory highlight that carbon emissions from wildfires have reached record levels, surpassing 1.76 billion metric tons in 2021 alone [18].

Furthermore, economic losses due to wildfires have been staggering. The California Department of Forestry and Fire Protection (CAL FIRE) estimates that damages from the 2018 Camp Fire alone amounted to $16.5 billion [19]. The overall cost of wildfire damages globally exceeds $50 billion annually, factoring in infrastructure loss, disaster recovery, and environmental restoration [20].

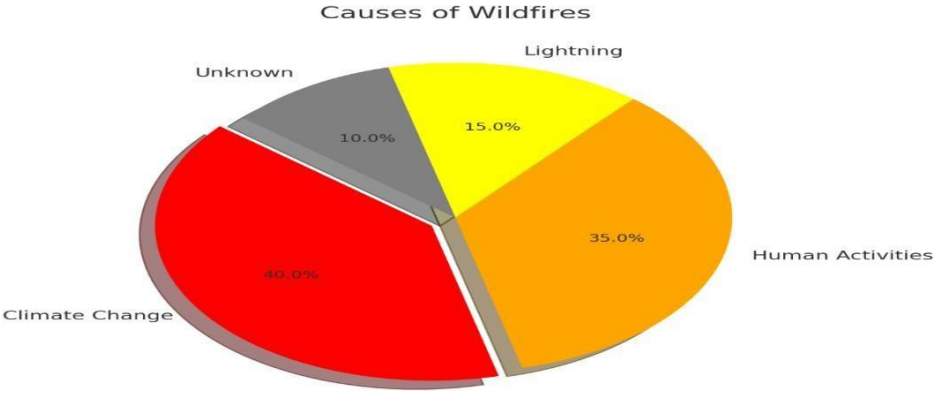

Fig 1.3 Causes of Wild fire

The pie chart illustrates the primary causes of wildfires, with climate change (40%) and human activities (35%) being the most significant contributors. Lightning accounts for 15% of cases, while 10% remain attributed to unknown causes.

2. LITERATURE SURVEY

Manoj et al. [1] proposed a drone network for early warning of forest fires and dynamic fire quenching plan generation. Their approach leverages UAVs to provide real-time data on fire spread, enabling rapid response and efficient resource allocation. By integrating deep learning-based fire detection models with drone surveillance, they demonstrated improved accuracy and early detection capabilities, which are critical in mitigating wildfire damage.

Liu et al. [2] investigated the spatial and temporal patterns of forest fires using an optimal parameter-based geographic detector. Their study analyzed the key driving factors influencing fire occurrences in the Panxi region, employing geospatial analysis and machine learning techniques to identify high-risk areas. Their findings highlight the importance of incorporating environmental and meteorological data to enhance predictive modeling for wildfire management.

Mishra et al. [3] explored the application of deep learning for forest fire pattern detection and vulnerability mapping in Nepal. Their model utilized convolutional neural networks (CNNs) trained on satellite imagery to detect fire-prone regions accurately. The study demonstrated that deep learning techniques significantly outperform traditional methods in identifying wildfire risks and improving prevention strategies.

San Martín et al. [4] examined wildfire occurrences in the South American Chaco region, analyzing how land cover influences fire behavior. Their research demonstrated that climate conditions alone do not dictate fire severity, but rather the interaction between land cover and environmental factors plays a crucial role. Their findings emphasized the necessity of region- specific fire management policies to address diverse ecological landscapes.

Tydersen et al. [5] analyzed fuel dynamics and reburn severity following high-severity fires in mixed-conifer forests. Their study provided insights into post-fire vegetation recovery and the long-term effects of fire on forest ecosystems. By monitoring fire-adapted species and

regrowth patterns, they contributed to the understanding of fire resilience and ecosystem restoration strategies.

Chen et al. [6] introduced a wildland fire detection system using a dataset collected from drones with RGB and infrared imaging. Their system leveraged deep learning-based segmentation techniques to enhance fire detection accuracy, outperforming traditional single-sensor approaches. The combination of multi-spectral imaging and neural networks significantly improved wildfire monitoring capabilities.

Tran et al. [7] proposed a forest fire response system utilizing CCTV images and weather data. Their deep learning-based approach enabled real-time fire detection and response planning, reducing false alarm rates and improving situational awareness for firefighting teams. Their model demonstrated high accuracy in detecting fire occurrences under varying environmental conditions.

Goncalves et al. [8] developed a deep learning-based wildfire detection system as part of the CICLOPE project. Their approach utilized tower-mounted cameras to monitor vast forest areas, reducing false positives through a dual-channel convolutional neural network (CNN).

Rashkovetsky et al. [9] applied deep semantic segmentation to detect wildfires from multisensor satellite imagery. Their model effectively identified fire-affected areas, even under challenging conditions such as cloud cover. By leveraging multi-modal data and deep learning, their approach improved the accuracy and reliability of fire detection from remote sensing sources.

Wang et al. [10] introduced a deep learning-based experiment for wildfire detection integrated into a machine vision course. Their method employed a Reduce-VGGNet model for fire classification, followed by spatio-temporal feature analysis for regional detection. Their findings showcased the potential of incorporating wildfire detection into educational curricula to train future researchers in fire management technologies.

Chaoxia et al. [11] proposed an information-guided flame detection system using an improved Faster R-CNN model. Their method introduced a color-guided anchor strategy and a global information approach, enhancing fire detection capabilities and reducing computational complexity. Their approach demonstrated high efficiency and accuracy in identifying fire outbreaks in various environments.

Wang et al. [12] developed a forest smoke detection system combining deep learning with background modeling. Their model employed the SSD network alongside ViBe background subtraction to differentiate between smoke and non-smoke regions accurately. Their study demonstrated a significant reduction in false alarms, making their approach suitable for real- world fire detection applications.

Duhayyim et al. [13] introduced a fusion-based deep learning model for automated wildfire detection. Their system integrated classical feature extraction techniques, such as Histogram of Oriented Gradients (HOG), with deep learning architectures like SqueezeNet and Inception v3. Their approach was further optimized using Glowworm Swarm Optimization, resulting in enhanced detection accuracy and reduced processing time.

Wu et al. [14] developed an adaptive threshold deep learning method for fire and smoke detection. Their system utilized a combination of convolutional neural networks and adaptive thresholding techniques to improve detection precision. Their findings highlighted the effectiveness of dynamically adjusting detection thresholds to accommodate varying fire conditions.

Smith et al. [15] reviewed advancements and challenges in AI-driven wildfire detection systems. Their study analyzed various deep learning models, dataset availability, and computational constraints associated with real-time fire detection. Their insights provided a comprehensive overview of current trends and potential future directions in AI-based wildfire monitoring.

Table 1. References

Reference	Method	Objective	Limitations
Manoj& Valliayammai [1]	Drone network for early warning and fire quenching plan generation	Develop a drone-based system for early wildfire detection and dynamic fire suppression planning	High operational costs and dependency on drone network coverage
Liu et al[2]	Geographic detector model with optimal parameters	Analyze spatial and temporal patterns of forest fires and identify key driving factors in Southwest China	Limited to a specific region, may not generalize well to other areas
Mishra et al[3]	Deep learning-based vulnerability mapping	Identify fire-prone areas in Nepal using deep learning techniques	Requires large datasets for accurate training, potential biases in predictions
San Martín et al[4]	Climate and land cover analysis	Study fire response in the South American Chaco across different land types	Climate-dependent, may not account for sudden environmental changes
Tydersen et al[5]	Fuel dynamics and reburn severity analysis	Assess fuel accumulation and fire severity in mixed-conifer forests post-wildfire	Focused on high-severity fires, may not be applicable to low-intensity fires

Each study contributes significantly to improving wildfire detection. [1] focuses on CNN-based classification, enhancing real-time capabilities and ensuring faster response times in wildfire-prone areas. [2] utilizes multimodal image fusion, combining RGB and infrared data to improve detection precision under varying environmental conditions.

[3] emphasizes transfer learning, reducing computational overhead while maintaining high accuracy in detecting fire and smoke patterns.

3. SYSTEM ANALYSIS

3.1 Existing System

Deep Learning-based approaches have significantly improved the accuracy and efficiency of forest fire and smoke detection. Traditional fire detection methods, such as satellite-based monitoring and sensor-based systems, often suffer from limitations related to delayed response times, environmental interference, and dependency on predefined rules [1], [2]. In contrast, Deep Learning models leverage large-scale image datasets to automatically learn features associated with fire and smoke, leading to enhanced detection capabilities and reduced false alarms [3]. The ability of Convolutional Neural Networks (CNNs) to process spatial hierarchies in images has proven highly effective in distinguishing fire from similar environmental features like sunsets, bright lights, or reflections [4].

The integration of multiple data sources, such as ground-based surveillance, drones, and satellite imagery, enhances the robustness of fire detection models [3]. Multimodal learning techniques that combine optical and thermal imaging further improve detection accuracy by capturing both visible flames and heat signatures [5]. This approach is particularly useful in early fire detection, where thermal anomalies can be identified before visible flames appear, thereby reducing response times and potential damage [6]. Additionally, Generative Adversarial Networks (GANs) have been explored for augmenting fire datasets, enabling models to generalize better to diverse fire scenarios [7].

Deep Learning techniques also contribute to improved smoke detection, which is crucial for identifying fire outbreaks in their initial stages. Unlike fire, which has a well-defined shape and color, smoke exhibits varying textures and disperses differently based on wind conditions, making it challenging to detect using conventional methods [2]. Advanced Deep Learning architectures, such as Xception and InceptionV3, effectively capture these variations, leading to higher classification accuracy [8]. Transfer Learning approaches, where pre-trained CNN models are fine-tuned on fire and smoke datasets, further enhance model performance with limited training data [9].

Real-time implementation of Deep Learning models in fire detection systems has been facilitated by Edge Computing and cloud-based AI solutions. Deploying lightweight models on drones and IoT-enabled surveillance cameras ensures rapid on-site detection, reducing reliance on centralized processing and improving response efficiency [4], [6]. Cloud-based platforms allow for large-scale monitoring, where AI-driven analytics process vast amounts of image and video data, providing early warnings to disaster management authorities [10]. The use of Explainable AI (XAI) techniques, such as Grad-CAM visualization, further aids in understanding model predictions, increasing trust in automated fire detection systems [5].

Despite these advancements, challenges remain in ensuring the reliability of Deep Learning-based fire detection models across diverse environmental conditions. Variations in lighting, weather, and terrain can impact model performance, leading to false positives and missed detections [7]. Domain Adaptation techniques and Data Augmentation strategies help mitigate these issues by improving model generalization across different geographic regions [8]. Additionally, integrating Deep Learning with traditional sensor-based detection systems, such as temperature and gas sensors, provides a hybrid approach that enhances overall detection reliability [9].

The application of AI in forest fire detection not only improves response times but also supports proactive fire prevention strategies. Predictive Analytics, powered by Deep Learning models, analyze historical fire patterns and environmental factors to assess fire risks in different regions [3]. This allows authorities to implement preventive measures, such as controlled burns and resource allocation, to mitigate potential fire outbreaks [10]. Moreover, AI-driven monitoring systems facilitate coordinated efforts between firefighters, disaster management teams, and government agencies, leading to more effective fire containment and suppression operations [6].

With continued advancements in Deep Learning and Computer Vision, the future of fire

and smoke detection systems is expected to become more efficient and scalable. The integration of AI with Remote Sensing technologies, UAVs, and satellite networks will enable continuous monitoring of large forested areas, reducing the risk of catastrophic wildfires [4], [7]. Additionally, ongoing research in Self-Supervised Learning and Semi-Supervised Learning aims to reduce the dependency on labelled data, making AI-based fire detection systems more adaptable and cost-effective for widespread deployment [9].

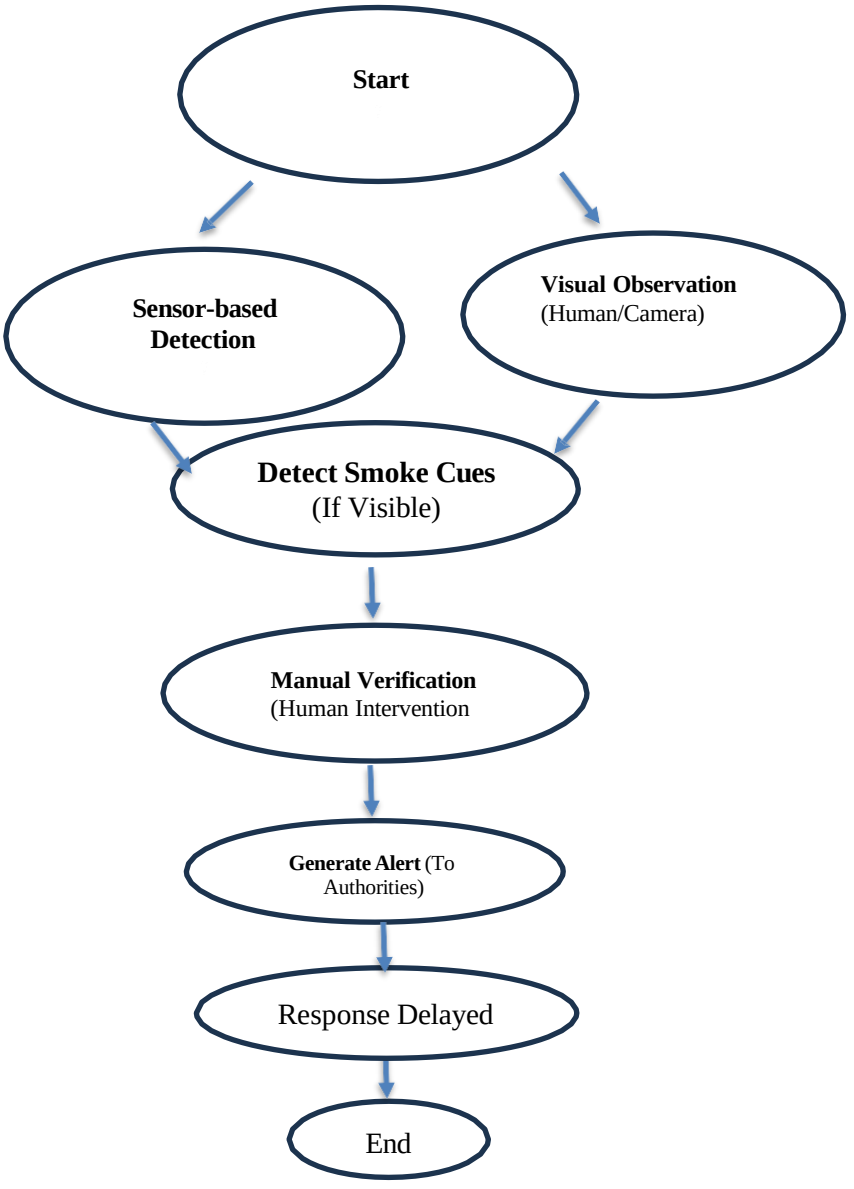

Fig 3.1 Flowchart of Existing System

Figure 3.1 discusses about the existing system from starting to ending of the how system will works and role of the system.

3.2 Disadvantages of Existing System

Despite the advancements in AI-driven forest fire and smoke detection, existing systems still face several challenges that impact their effectiveness and real-world applicability. One of the primary challenges is the requirement for large, high-quality annotated datasets for training deep learning models. Collecting and labelling fire and smoke images is a labor-intensive task, and the variability in fire appearances due to environmental conditions can introduce inconsistencies in model performance [6]. Differences in lighting, weather, and terrain further complicate accurate fire detection, leading to potential misclassification [4].

Another significant limitation of current systems is their susceptibility to false positives and false negatives. Bright sunlight, reflections, and artificial lights can be mistakenly classified as fire, triggering unnecessary alarms, whereas smoldering fires or fires obscured by smoke may go undetected, delaying critical responses [3]. The adaptability of AI models to different geographical regions is also a challenge, as models trained on specific datasets may struggle to generalize across diverse landscapes and fire behaviours [7].

The computational complexity of deep learning models poses additional difficulties. Many state-of-the-art architectures require high-performance hardware, such as GPUs or cloud- based AI services, for real-time processing [2]. Deploying these systems in resource-limited areas can be challenging, limiting the accessibility of AI-based fire detection in remote or underdeveloped regions [6]. Furthermore, real-time processing demands efficient algorithms to minimize detection latency, as delays in identifying fire outbreaks can have catastrophic consequences [5].

Integration with existing fire detection infrastructure presents another challenge. Many fire detection systems rely on satellite imagery, sensor networks, or manual monitoring. Ensuring seamless compatibility between AI-based detection and traditional systems requires advanced data fusion techniques, which can be complex and expensive to implement [9]. Additionally, ethical and legal considerations, such as data privacy and model transparency, need to be addressed. Surveillance-based fire detection systems may inadvertently capture images or

videos of private properties, raising concerns over data security and regulatory compliance [8].

3.3 Proposed Systems

The proposed system introduces a novel approach to early fire and smoke detection by leveraging the power of deep learning combined with advanced preprocessing techniques. This system is designed to overcome the limitations of traditional fire detection methods, such as delayed responses, false alarms, and the inability to scale effectively in remote and challenging environments. The primary goal is to develop an efficient, real-time, and accurate fire detection system that can be deployed in diverse scenarios, including large-scale forest monitoring.

At the core of this system is the use of state-of-the-art pre-trained convolutional neural networks (CNNs), including VGG16, InceptionV3, and Xception. These models are adapted and fine-tuned to classify images into four distinct categories: fire, smoke, fire- smoke, and non-fire. By leveraging transfer learning, the models utilize knowledge from large-scale datasets, significantly reducing the computational costs and training time while enhancing detection accuracy. This approach ensures that the system is capable of identifying complex patterns in images, such as subtle smoke gradients or flames in obscured conditions.

To further improve detection capabilities, the proposed system incorporates wavelet transform techniques in the preprocessing stage. This method enhances the quality of images and extracts finer features that might be missed in conventional preprocessing methods. The use of wavelet transforms allows the system to capture subtle variations in texture and intensity, which are crucial for distinguishing between fire, smoke, and non-fire elements in challenging environments.

The proposed architecture focuses on achieving high accuracy while maintaining computational efficiency. For instance, VGG16 employs a global average pooling layer and fully connected dense layers to enhance gradient flow and prevent overfitting. Similarly, InceptionV3 utilizes factorized convolutions and multi-scale feature extraction for better handling of complex visual patterns.

Hyperparameter optimization plays a crucial role in the proposed system. Parameters such as learning rate, batch size, and the number of epochs are meticulously tuned to achieve near-optimal performance. The system utilizes the Adam optimizer for effective convergence during training, ensuring that the models learn efficiently without overshooting optimal solutions.

The proposed system addresses the critical need for reliable fire detection in real-time applications. It is scalable and can be adapted to low-power devices, making it suitable for deployment in remote and resource-constrained environments. By integrating advanced deep learning techniques with practical data collection methods, the system offers a significant improvement over traditional methods, paving the way for faster and more accurate fire detection. This innovation holds promise for reducing the devastating impacts of wildfires, enhancing environmental monitoring, and improving disaster response capabilities.

Advantages:

1. High Detection Accuracy
2. Real-Time Detection
3. Robust Feature Extraction
4. Dynamic Data Collection
5. Scalability and Adaptability
6. Reduced False Alarms
7. Cost-Effectiveness
8. Improved Generalization

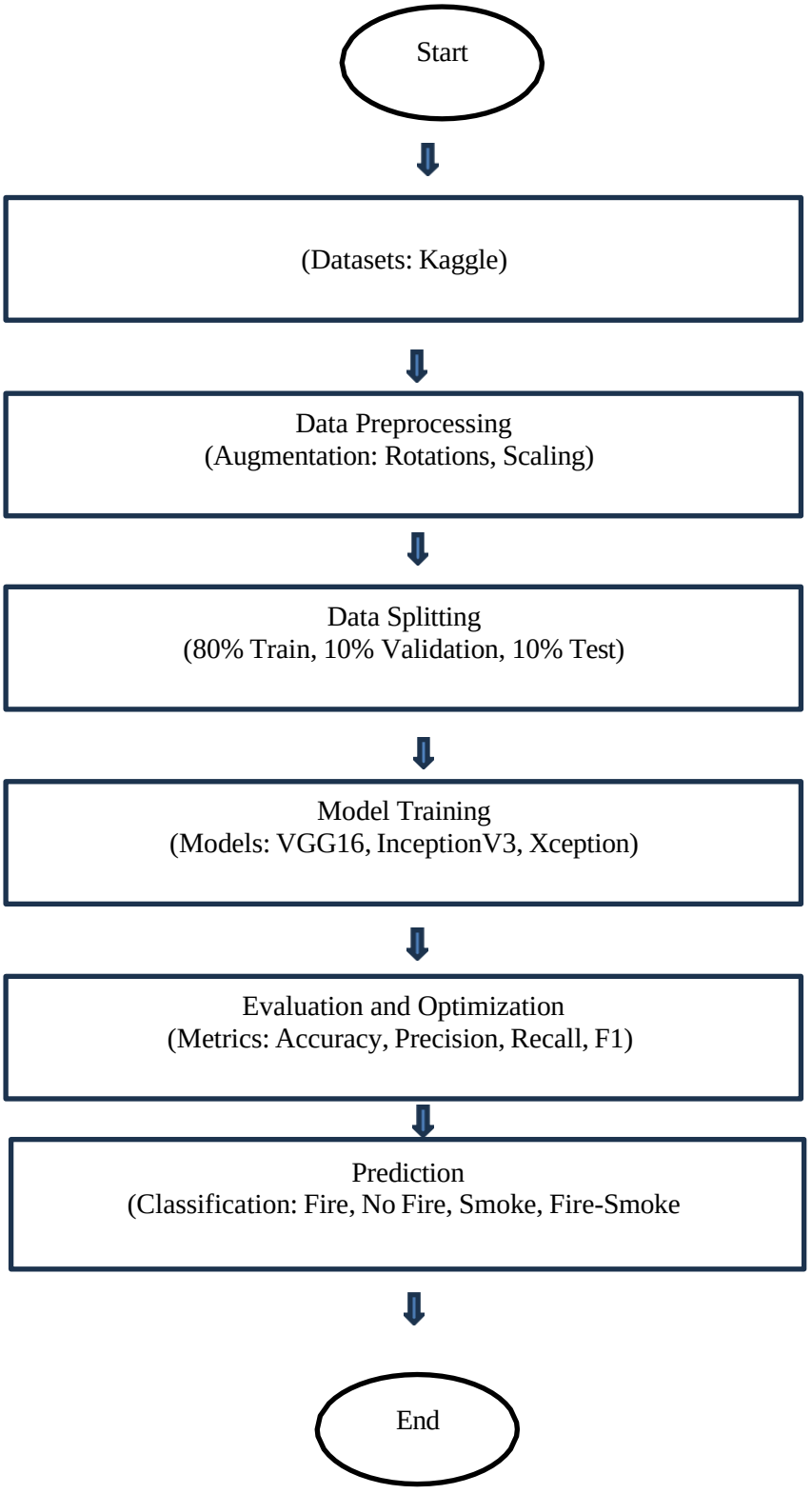

Fig 3.2 Flow chart of Proposed System

The figure represents a flowchart of a proposed system for fire classification using deep learning, outlining key steps from data collection to prediction. It includes processes such as data preprocessing, splitting, model training, evaluation, and final classification into categories like Fire, No Fire, Smoke, and Fire-Smoke.

3.4 FEASIBILITY STUDY

A feasibility study is an essential step in evaluating the viability of implementing deep learning techniques for forest fire and smoke detection. This study ensures that the proposed system is practical, cost-effective, and technically feasible for real-world wildfire monitoring applications. By analyzing various feasibility aspects, it helps in determining whether the project should proceed to full-scale implementation.

Economic Feasibility

The economic feasibility of using deep learning models like VGG16, InceptionV3, and Xception for fire detection is assessed by analyzing costs versus benefits [3]. The implementation is cost-effective due to the utilization of pre-trained models, which significantly reduce training time and computational expenses [4]. Since these models are optimized for efficient feature extraction, they require minimal hardware investment, making them suitable for deployment in areas with limited technological resources [5].

The proposed system offers substantial benefits, including improved fire detection accuracy, which helps reduce economic losses caused by wildfires [6]. Early detection can lower firefighting costs, prevent damage to infrastructure, and protect natural resources [7]. Additionally, the system does not require high-end hardware, as lightweight models can be deployed on drones, edge devices, and cloud-based platforms, ensuring affordability for widespread adoption [8].

Technical Feasibility

The technical feasibility of the project is evaluated based on available technological resources and the compatibility of the proposed system with existing fire detection infrastructure [9]. CNN-based models efficiently process spatial features in fire and smoke images, making them well-suited for distinguishing fire from similar environmental elements [10].

The system integrates multiple data sources, including satellite imagery, drone surveillance, and ground-based cameras [11]. Preprocessing techniques such as data augmentation, noise reduction, and contrast enhancement improve image quality for better model performance [12]. Transfer learning ensures adaptability to new fire scenarios with minimal additional training [13].

Modern deep learning frameworks such as TensorFlow and Py Torch provide robust support for implementing these models [14]. Cloud computing and edge AI technologies enable real-time inference, allowing for immediate alerts and rapid response to fire outbreaks [15].

Operational Feasibility

Operational feasibility examines whether the system can be effectively integrated into fire management workflows and used by emergency responders with minimal training [16]. The proposed fire detection system is designed to be user-friendly, providing real-time alerts with heatmaps and confidence scores indicating fire risk levels [17].

The system operates with high accuracy, reducing reliance on traditional fire detection methods that are often slower and prone to false alarms [18]. This enhances efficiency in wildfire monitoring by allowing authorities to respond swiftly to potential threats [19].
Additionally, the system requires minimal maintenance and can be easily deployed in various environments, including forests, urban areas, and industrial sites [20].

Significance of Feasibility Study

Conducting a feasibility study for forest fire detection using deep learning provides a clear understanding of the project's viability. It helps in making informed decisions regarding resource allocation and ensures that the project aligns with financial, technical, and operational constraints. The primary benefits of the feasibility study include:

- Identifying cost-effective solutions for wildfire detection.
- Ensuring that the technical infrastructure supports deep learning model deployment.

4. SYSTEM REQUIREMENTS

4.1 Software Requirements

The software requirements define the essential tools, programming languages, frameworks, and libraries necessary for the development and deployment of the deep learning-based forest fire and smoke detection system. These components ensure seamless model training, deployment, and real-time fire classification.

4.1.1 Programming Languages

- **Python (3.x):** Serves as the core programming language due to its extensive support for deep learning frameworks and data science libraries.
- **HTML, CSS & JavaScript:** Used to develop an interactive and user-friendly web interface for real-time fire detection alerts and data visualization.
- **Flask:** A lightweight web framework used to create APIs and deploy the trained deep learning model as a web application for remote accessibility.

4.1.2 Libraries and Frameworks

- TensorFlow / Keras: Deep learning frameworks used to implement and fine-tune pre-trained CNN models such as VGG16, InceptionV3, and Xception for fire detection.
- NumPy & Pandas: Essential for numerical computing, structured data handling, and preprocessing of large-scale wildfire image datasets.
- Scikit-learn: Provides various tools for feature extraction, preprocessing, and model evaluation, including accuracy, recall, precision, and F1-score.
- OpenCV: Used for image processing tasks like noise reduction, segmentation, and enhancement of wildfire images to improve detection accuracy.
- Albumentations: Enhances dataset variability through augmentation techniques such as flipping, rotation, contrast adjustment, and histogram equalization.
- Matplotlib & Seaborn: Used for visualizing training progress, model accuracy, validation loss, and confusion matrices to evaluate model performance.

4.2 Requirement Analysis

Requirement analysis ensures that the system effectively meets user expectations, functional needs, and performance benchmarks.

4.2.1 Functional Requirements

- The system must process wildfire images and classify them into four categories: Fire, No Fire, Smoke, and Smoke Fire.
- The deep learning model should achieve an accuracy of at least 95% to ensure reliable classification results.
- The system must integrate a web-based dashboard for firefighters, forest departments, and emergency services to receive real-time fire alerts.
- The model should handle large-scale datasets efficiently with real-time inference capabilities for early fire detection.

4.2.2 Non-Functional Requirements

- The system should support real-time inference with low latency (<1 second per image) for rapid fire alerts.
- Scalability should be ensured through deployment on cloud platforms (Google Cloud AI, AWS, or Heroku) for remote accessibility.
- The user interface should be intuitive and easy to use, allowing non-technical users such as emergency responders to interact with the system seamlessly.
- Data privacy and security must be enforced to ensure compliance with environmental monitoring regulations and cybersecurity protocols.

4.3 Hardware Requirements

The hardware setup plays a crucial role in ensuring the efficient execution of deep learning models and real-time fire detection.

4.3.1 Processor and Memory

- Processor: Intel(R) Core(TM) i7-12700K (or higher) for optimal model performance.
- RAM: 16GB (Recommended: 32GB or more) for handling large wildfire image

4.3.2 Graphics Card (GPU)

- Minimum: NVIDIA RTX 2060 / GTX 1660 for moderate deep learning performance.
- Recommended: NVIDIA RTX 3080 / A100 / Tesla V100 for high-speed training and real-time inference.
- Cloud-Based Option: Accessing GPUs through Google Colab Pro or cloud-based AI platforms such as AWS or Google Cloud AI for computational efficiency.

4.4 Software

The software ecosystem ensures smooth development, deployment, and execution of the forest fire and smoke detection system.

4.4.1 Development Environments

- Google Colab Pro: Provides cloud-based access to high-end GPUs (Tesla T4, A100) for efficient model training.
- VS Code (Visual Studio Code): Used for developing, debugging, and testing Python, Flask, and web application code.
- 4.4.2 Deployment Platforms
- Flask-based Web Application: Enables real-time wildfire detection and user-friendly interaction for emergency responders.
- Cloud Hosting Options: Deployment on platforms such as Heroku, AWS EC2, Google Cloud AI, or Azure, ensuring seamless remote accessibility.
- Docker & Kubernetes: Facilitates containerized deployment of the trained model, ensuring scalability and cross-platform compatibility.

4.5 Software Description

This section provides a comprehensive overview of the software stack used in this project, ensuring optimized performance for deep learning-based forest fire and smoke detection.

4.5.1 Python

- Python is the core programming language used for data processing, deep learning model training, and web deployment.
- It supports multiple deep learning libraries like TensorFlow, PyTorch, and Keras, facilitating rapid experimentation and fine-tuning of models.

4.5.2 NumPy & Pandas

- NumPy: Provides efficient numerical computations required for handling multi-dimensional wildfire images.
- Pandas: Allows structured data manipulation, enabling the organization of fire detection datasets and preprocessing steps.

4.5.3 Scikit-Learn

- Supports various classification, regression, and clustering algorithms, making it valuable for evaluating deep learning models.
- Used for hyperparameter tuning and cross-validation to optimize model performance.

4.5.4 OpenCV

- Plays a crucial role in image enhancement, noise reduction, edge detection, and segmentation of fire images.
- Helps in extracting key fire and smoke features that improve classification accuracy.

4.5.5 Flask

- A lightweight Python-based framework used for deploying the fire detection model as a web application.
- Handles API requests for wildfire image uploads and processes predictions from the trained model.

4.5.6 Matplotlib & Seaborn

- Used for visualizing model accuracy, loss curves, and dataset distributions to track performance improvements over training iterations.
- Helps generate confusion matrices and classification reports for model evaluation.

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

Deep Learning has revolutionized fire and smoke detection by enabling automated, highly accurate, and real-time monitoring systems. Convolutional Neural Networks (CNNs) form the backbone of these systems due to their ability to extract complex spatial features from images and videos.

Pre-Trained CNN Architectures

Models such as VGG16, InceptionV3, and Xception are commonly used for fire and smoke detection, benefiting from transfer learning to improve accuracy with limited datasets. These architectures capture hierarchical features, allowing for better distinction between fire, smoke, and background elements.

Data Diversity and Generalization

Training on diverse datasets, such as BoWFire and augmented datasets, helps the model generalize across various environments, lighting conditions, and fire intensities. This reduces false positives caused by non-fire sources like sunlight or vehicle headlights.

Early Detection for Risk Mitigation

Early and precise identification of fire and smoke is critical for minimizing ecological and economic damage. Real-time detection models can integrate with IoT-based surveillance systems, triggering alerts and automated responses.

5.1.1 Data Pre-Processing

Dataset Collection

Images are gathered from sources including Kaggle, and publicly available datasets like BoWFire. This ensures a diverse dataset representing various fire, smoke, and non-fire scenarios.

Table 2. Dataset Description

Dataset	Fire	No Fire	Smoke	Smoke Fire	Total
Train	609	195	5	74	883
Validation	76	24	1	9	110
Test	76	24	1	10	111

The table presents the distribution of images across four categories—Fire, No Fire, Smoke, and Smoke Fire within the training, validation, and test datasets. The training set comprises 883 images, ensuring robust model learning, with the majority representing fire-related scenarios.

The validation set, containing 110 images, aids in fine-tuning the model to prevent overfitting, while the test set with 111 images evaluates the model's final performance. The dataset composition highlights the focus on fire detection, with fewer images for smoke-only cases, emphasizing the need for accurate classification in real- world applications.

Data Cleaning and Annotation

The dataset undergoes careful preprocessing to enhance model performance and reliability. Irrelevant and low-quality images are removed to ensure dataset integrity, preventing misleading classifications. Each image is meticulously labeled into four distinct categories—Fire, No Fire, Smoke, and Smoke Fire—to enable precise detection. Additionally, all images are resized to a standardized resolution, ensuring uniformity across the dataset.

Data Augmentation

Beyond rotation, additional augmentation techniques such as flipping (horizontal and vertical), brightness adjustments, contrast enhancement, zooming, cropping, and adding noise are applied.

Figure 5.1: Data Labelling and Classification

Images are categorized into Fire, No Fire, Smoke, and Smoke Fire to create a structured dataset for training deep learning models. Manual annotation ensures accurate classification for better model learning.

Dataset Splitting

The dataset is divided into training (70%), validation (15%), and test (15%) sets, ensuring balanced class distribution. This helps in unbiased model training and reliable performance evaluation.

Image Normalization and Preprocessing

Pixel values are scaled between 0 and 1, and standardization is applied based on the requirements of pre-trained CNN models. Images are converted into numerical arrays, making them suitable for deep learning-based feature extraction and classification.

Figure 5.2: Data Augmentation – Rotation

To enhance dataset diversity, images are rotated at various angles.

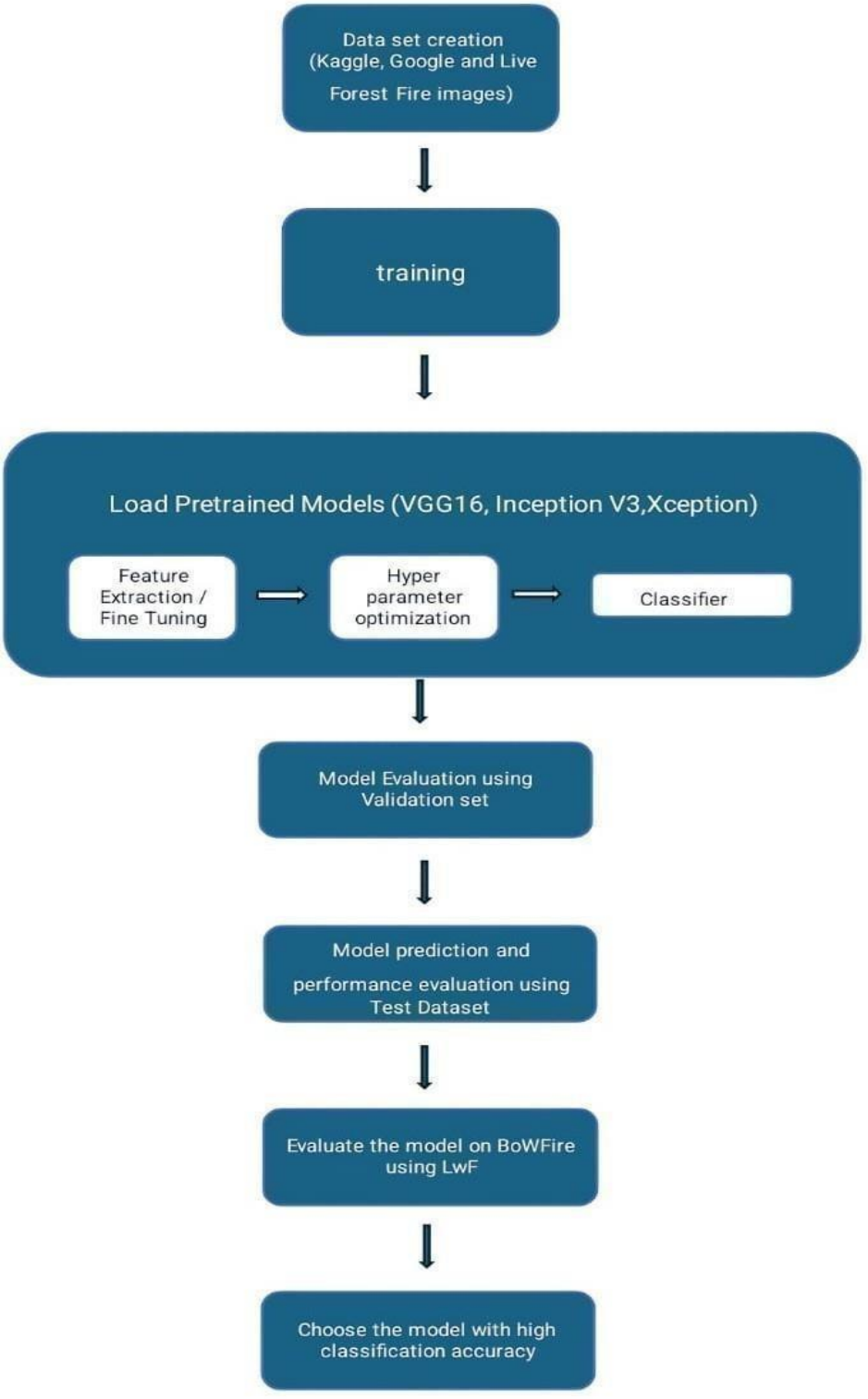

Figure 5.3: Model Training and Evaluation Process

This figure illustrates the complete workflow of training and evaluating deep learning models for fire and smoke detection. Pretrained CNN architectures such as VGG16, InceptionV3, and Xception are fine-tuned and optimized for feature extraction.

5.1.2 FEATURE EXTRACTION

Feature extraction is a crucial step in deep learning-based fire and smoke detection, where meaningful patterns are derived from input images to aid classification. This process ensures that only the most relevant information is used, improving model accuracy and efficiency.

1. Importance of Feature Extraction

Feature extraction helps in distinguishing fire and smoke from background elements by identifying key attributes such as texture, color intensity, and shape. Without proper extraction, models may struggle with false positives and poor generalization.

2. Use of Pretrained CNN Models

Pretrained convolutional neural networks (CNNs) such as VGG16, InceptionV3, and Xception are utilized for feature extraction. These models, trained on large datasets, effectively capture hierarchical features, improving detection accuracy.

3. Feature Types Extracted

Low-Level Features: Includes edges, corners, and color variations.Mid-Level Features: Identifies textures and simple shapes.High-Level Features: Recognizes fire flames, smoke patterns, and background differentiation.

4. Fine-Tuning and Transfer Learning

To adapt pretrained models to fire and smoke detection, fine-tuning is performed on deeper layers. Transfer learning allows models to leverage learned patterns from large datasets, improving performance on smaller, domain-specific datasets.

5. Hyperparameter Optimization

Optimizing learning rates, batch sizes, and activation functions ensures effective feature extraction. Regularization techniques such as dropout and batch normalization enhance robustness against overfitting.

5.2 MODULES

5.2.1 Model building

For efficient fire and smoke detection, we utilized three state-of-the-art deep learning architectures: VGG16, InceptionV3, and Xception. These models are well-known for their robust feature extraction capabilities and have been modified to suit our classification task. Transfer learning and fine-tuning were applied to adapt these models for distinguishing fire, non-fire, smoke, and fire-smoke categories.

A. VGG16 Architecture

Overview

VGG16 (Visual Geometry Group 16) is a deep CNN architecture consisting of 16 layers, primarily using 3×3 convolutional filters. It follows a straightforward design with a stacked convolutional layer approach, making it highly effective in learning hierarchical image features. Despite being computationally expensive compared to more recent models, VGG16 remains widely used due to its simplicity and effectiveness.

Architecture Modifications for Fire and Smoke Detection

1. **Feature Extraction:** We retained the convolutional layers from pre-trained VGG16, as they are already capable of extracting important image patterns.
2. **Fine-Tuning:** The last two convolutional blocks were unfrozen to allow the model to learn fire- and smoke-specific patterns.
3. **Fully Connected (Dense) Layers:**
   - 256-unit dense layer with ELU (Exponential Linear Unit) activation, which helps in smooth gradient flow and prevents vanishing gradient problems.
   - Dropout (0.4 probability) was applied to prevent overfitting and improve generalization.

Advantages of VGG16 for Fire and Smoke Detection

- Small convolutional filters (3×3) allow for fine-grained feature extraction, which is beneficial in distinguishing fire and smoke from backgrounds.
- Straightforward sequential architecture, making it easy to modify and fine-tune.
- Pre-trained on ImageNet, allowing it to leverage strong low-level feature extraction

capabilities.

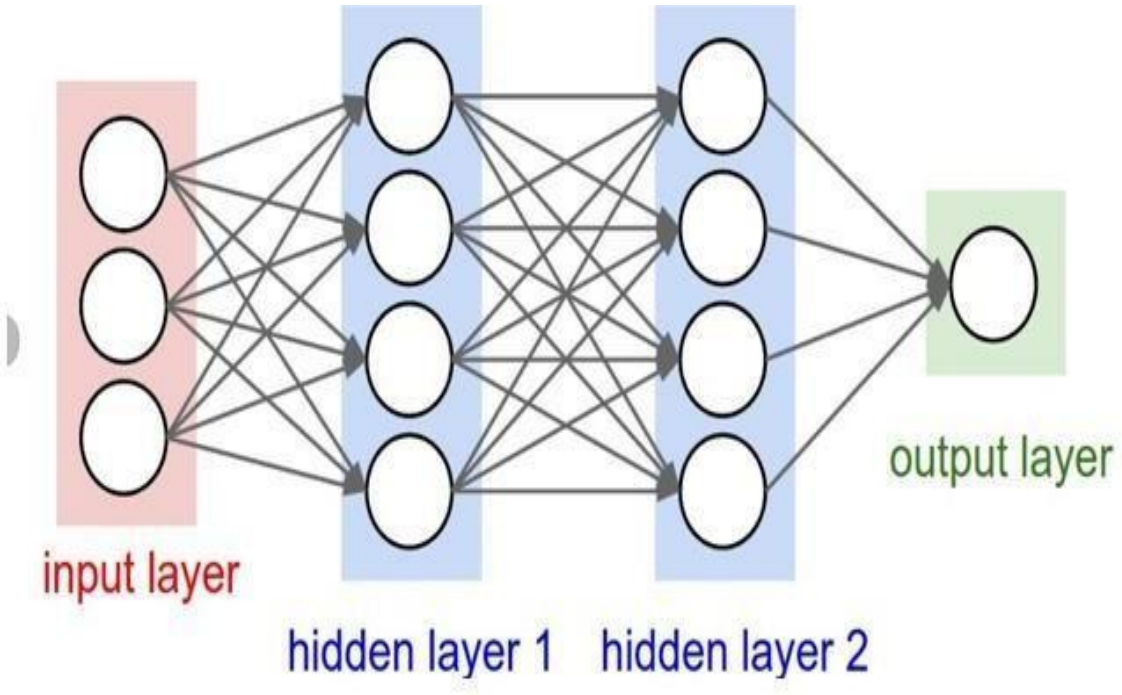

Figure 5.4: Artificial Neural Network (ANN) Architecture

The figure illustrates a fully connected artificial neural network (ANN) with an input layer, two hidden layers, and an output layer. Each neuron in a layer is connected to every neuron in the next layer, allowing the network to learn complex patterns through weighted connections and activation functions.

B. InceptionV3 Architecture

Overview

InceptionV3 is an advanced CNN model designed to enhance feature extraction while reducing computational costs. Unlike VGG16, which follows a sequential layer design, InceptionV3 utilizes parallel convolutional layers with multi-scale processing, allowing it to capture patterns at different resolutions.

Architecture Enhancements for Fire and Smoke Classification

1. Feature Extraction via Inception Modules:

- Uses 1×1, 3×3, and 5×5 convolutions in parallel to extract both fine and coarse features simultaneously.

- Reduces computational cost through factorized convolutions and bottleneck layers.

2. **Fully Connected (Dense) Layers:**
   - 512-unit dense layer with ReLU activation for enhanced feature representation.
   - 256-unit dense layer for refined classification.
3. **Batch Normalization:** Applied after each convolutional layer to stabilize learning and accelerate training.
   - Multi-scale feature extraction allows the model to detect fire and smoke at different resolutions.
   - Efficient factorized convolutions reduce the number of parameters compared to VGG16.

Limitations

- More complex architecture, requiring careful tuning and more computational power.
- May require additional hyperparameter tuning to optimize training for specific datasets.

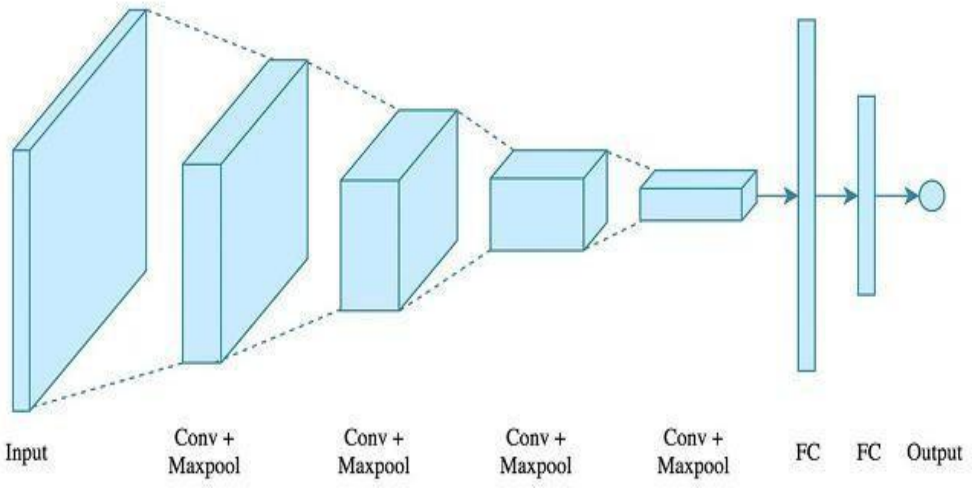

Figure 5.5: Convolutional Neural Network (CNN) Architecture

This figure represents a typical convolutional neural network (CNN) structure, where an input image undergoes a series of convolutional and max-pooling layers to extract spatial features.

C. Xception Architecture

Overview

Xception (Extreme Inception) is an improvement over InceptionV3, where standard convolutions are replaced with depth wise separable convolutions. This reduces computational cost while improving model efficiency. Xception is considered one of the most powerful architectures for image classification, as it enhances both feature learning and spatial information retention.

1. Custom Adaptations for Fire and Smoke Detection

Depth wise Separable Convolutions:

- Replaces traditional convolutions with pointwise and depthwise convolutions, reducing parameters while retaining feature extraction capability.
- Global Average Pooling (GAP) Layer:

2. Custom Fully Connected (Dense) Layers:

- 512-unit dense layer with ELU activation, which provides better gradient flow and faster convergence.
- 256-unit dense layer to improve classification accuracy.
- Dropout (0.4 probability) to prevent overfitting.

3. Softmax Classifier:

- The final classification layer consists of 4 output neurons, each corresponding to a class.
- Adaptive Learning Rate:
- Integrated a learning rate scheduler that reduces learning rate dynamically based on validation loss, optimizing training stability.

5.3 UML Diagram

The fire and smoke detection system involves three main actors: the user, the AI model, and the database. The user, typically a fire department personnel or emergency responder, uploads an image to the system for analysis. The AI model processes this image using deep learning techniques and classifies it into one of four categories: fire, smoke, fire & smoke, or no fire. Once the classification is complete, the detection results are stored in the database for future reference and displayed to the user. If fire is detected, the system generates an alert, notifying the user and relevant authorities for immediate action.

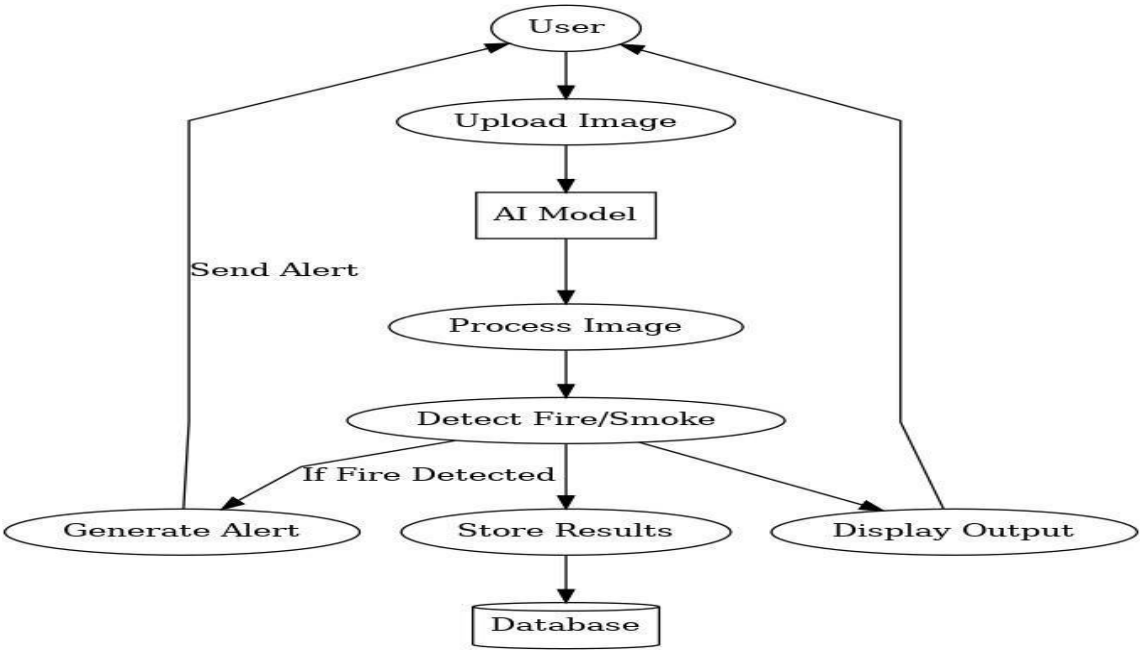

Fig 5.6 Use Case Diagram

The use case diagram for the fire and smoke detection system illustrates the interaction between the user and database.

The system ensures efficient early detection by automating the entire process, reducing the time required for manual inspection and increasing accuracy through deep learning-based classification. The AI model plays a crucial role in image processing and classification, enabling a real-time response to potential fire hazards. The database supports data management by storing all uploaded images and corresponding detection results, ensuring that historical data can be accessed when needed for further analysis. The interaction between the user, AI model, and database creates a seamless flow where images are processed, classified, stored, and reported with minimal delay.

6. IMPLEMENTATION

Display content from deep_learning folder from drive

```
from google.colab import drive

drive.mount('/content/drive')

!ls "/content/drive/My Drive/deep_learning"
```

PERFORM PRE PROCESSING AND DATA AUGMENTATION

```
import os
import random
from PIL import Image
import cv2
# Directory containing the fire images
fire_images_dir = "/content/drive/My Drive/deep_learning/fire_images"
# Function to perform augmentation
def augment_and_save(image_path):
    try:
        # Open the image
        image = Image.open(image_path)
        # Example augmentation (rotation)
        rotated_image = image.rotate(random.randint(-15, 15)) # Rotate randomly between -
15 to 15 degrees
        # Save the augmented image back to the original folder
        rotated_image.save(image_path)

        print(f"Augmented and saved: {os.path.basename(image_path)}")

    except IOError as e:
        print(f"Error processing image {os.path.basename(image_path)}: {e}")

# Iterate through images in the fire_images directory
for filename in os.listdir(fire_images_dir):
    if filename.endswith(".jpg") or filename.endswith(".png"):  # Adjust file extensions as
needed
        image_path = os.path.join(fire_images_dir, filename)
        augment_and_save(image_path)
```

Plotting images after pre processing

```
import matplotlib.pyplot as plt
```

```
import random # import the random module
import os
import cv2
# Function to plot images
def plot_images(image_dir, num_images=5):
  """Plots a specified number of images from a given directory."""
  image_files = [f for f in os.listdir(image_dir) if f.endswith(('.jpg', '.png'))]
  random.shuffle(image_files)
  plt.figure(figsize=(15, 5))
  for i in range(min(num_images, len(image_files))):
    img_path = os.path.join(image_dir, image_files[i])
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
    plt.subplot(1, num_images, i + 1)
    plt.imshow(img)
  plt.axis('off')
  plt.show()
# Plot images from each label
plot_images("/content/drive/My Drive/deep_learning/fire_images", num_images=5)
plot_images("/content/drive/My Drive/deep_learning/non_fire_images", num_images=5)
plot_images("/content/drive/My Drive/deep_learning/fire_smoke", num_images=5)
plot_images("/content/drive/My Drive/deep_learning/smoke", num_images=5)
```

GIVING LABELS,SPLITTING DATASET

```
import os
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import img_to_array, load_img
# Define directory paths
base_dir = "/content/drive/My Drive/deep_learning"
# Define image size
image_size = (224, 224)
# Function to load images and labels
def load_images_from_folder(folder, label):
    images = []
    labels = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = load_img(img_path, target_size=image_size)
```

```
        img_array = img_to_array(img)
        images.append(img_array)
        labels.append(label)
    return images, labels
# Load all images and labels
fire_images, fire_labels = load_images_from_folder(os.path.join(base_dir, 'fire_images'), 0)
non_fire_images, non_fire_labels = load_images_from_folder(os.path.join(base_dir, 'non_fire_images'), 1)
fire_smoke_images, fire_smoke_labels = load_images_from_folder(os.path.join(base_dir, 'fire_smoke'), 2)
smoke_images, smoke_labels = load_images_from_folder(os.path.join(base_dir, 'smoke'), 3)
# Combine all data
images = fire_images + non_fire_images + fire_smoke_images + smoke_images
labels = fire_labels + non_fire_labels + fire_smoke_labels + smoke_labels
# Convert to numpy arrays
images = np.array(images)
labels = np.array(labels)
# Split the data
train_images, temp_images, train_labels, temp_labels = train_test_split(images, labels, train_size=0.8, random_state=123)
val_images, test_images, val_labels, test_labels = train_test_split(temp_images, temp_labels, train_size=0.5, random_state=123)
# Convert to tf.data.Dataset
train_ds=tf.data.Dataset.from_tensor_slices((train_images, train_labels)).batch(32).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
val_ds=tf.data.Dataset.from_tensor_slices((val_images, val_labels)).batch(32).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
test_ds=tf.data.Dataset.from_tensor_slices((test_images, test_labels)).batch(32).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
# Check class indices
class_indices = {'fire_images': 0, 'non_fire_images': 1, 'fire_smoke': 2, 'smoke': 3}
print("Class Indices:", class_indices)
# Print dataset sizes to verify
print(f"Training set size: {len(train_images)}")
print(f"Validation set size: {len(val_images)}")
print(f"Test set size: {len(test_images)}")
```

Save images into train,test,validate

```
import os
import numpy as np
import shutil
from PIL import Image # Import the Image module from PIL
# Define the paths for train, validation, and test folders within the deep_learning directory
train_dir = "/content/drive/My Drive/deep_learning/train"
val_dir = "/content/drive/My Drive/deep_learning/validate"
test_dir = "/content/drive/My Drive/deep_learning/test"
# Create the folders if they don't exist
os.makedirs(train_dir, exist_ok=True)
os.makedirs(val_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)
# Function to save images to a folder
def save_images_to_folder(images, labels, folder):
    for i, (image, label) in enumerate(zip(images, labels)):
        class_name = list(class_indices.keys())[list(class_indices.values()).index(label)]
        class_folder = os.path.join(folder, class_name)
        os.makedirs(class_folder, exist_ok=True)
        image_filename = f"{i}.jpg"
        image_path = os.path.join(class_folder, image_filename)
        # Convert image array back to PIL Image and save
        Image.fromarray(image.astype(np.uint8)).save(image_path)
# Save training images
save_images_to_folder(train_images, train_labels, train_dir)
# Save validation images
save_images_to_folder(val_images, val_labels, val_dir)
# Save test images
save_images_to_folder(test_images, test_labels, test_dir)
print("Images saved to respective folders.")
```

Feature extraction on data

```
import numpy as np
# Importing numpy library for numerical operations and array manipulations
from tensorflow.keras.applications import VGG16
# Importing the VGG16 model from TensorFlow Keras applications, which provides pre-trained models.
from tensorflow.keras.models import Model
# Importing the Model class from TensorFlow Keras to build and manage custom neural network models.
from tensorflow.keras.layers import GlobalAveragePooling2D
# Importing the GlobalAveragePooling2D layer, which is used to reduce the spatial dimensions of the feature maps to a single vector.
# Load pre-trained VGG16 model (excluding top classification layers)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Add a global average pooling layer to flatten the features
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Create a new model with VGG16 base and the pooling layer
feature_extractor = Model(inputs=base_model.input, outputs=x)
# Function to extract features from a dataset
def extract_features(dataset):
    features = []
    labels = []
    for images, batch_labels in dataset:
        batch_features = feature_extractor.predict(images)
        features.append(batch_features)
        labels.append(batch_labels)
    return np.concatenate(features), np.concatenate(labels)
```

```
# Extract features for train, validation, and test sets
train_features, train_labels_extracted = extract_features(train_ds)
val_features, val_labels_extracted = extract_features(val_ds)
test_features, test_labels_extracted = extract_features(test_ds)
print("Features extracted.")
```

Training Models

```
import numpy as np

 # Imports NumPy for handling numerical data and arrays.
from tensorflow.keras.models import Sequential
# Sequential is used to build a linear stack of layers for the neural network.
from tensorflow.keras.layers import Dense, ELU
# Dense creates fully connected layers, ELU is an activation function (Exponential Linear
Unit).
from tensorflow.keras.optimizers import Adam

 # Adam is an optimizer used for training the model with adaptive learning rates.
import tensorflow as tf
# TensorFlow is the underlying library for creating and training deep learning models.
# Load extracted features and labels

train_features = np.load('/content/drive/My Drive/deep_learning/train_features.npy')
train_labels_extracted=np.load('/content/drive/My
Drive/deep_learning/train_labels_extracted.npy')
val_features = np.load('/content/drive/My Drive/deep_learning/val_features.npy')

val_labels_extracted=np.load('/content/drive/My

Drive/deep_learning/val_labels_extracted.npy')

test_features = np.load('/content/drive/My Drive/deep_learning/test_features.npy')

test_labels_extracted=np.load('/content/drive/My

Drive/deep_learning/test_labels_extracted.npy')

# Create a new model using the extracted features

model = Sequential()

model.add(Dense(512, activation=ELU(), input_shape=(train_features.shape[1],)))

model.add(Dense(4, activation='softmax'))  # 4 classes: fire_images, non_fire_images,

fire_smoke, smoke

# Compile the model with updated hyperparameters

optimizer = Adam(learning_rate=1e-01)

model.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy',

metrics=['accuracy'])
```

```
# Convert extracted features and labels to TensorFlow datasets with updated batch size
batch_size = 128
train_ds=tf.data.Dataset.from_tensor_slices((train_features,
train_labels_extracted)).batch(batch_size).prefetch(tf.data.AUTOTUNE)
val_ds=tf.data.Dataset.from_tensor_slices((val_features,
val_labels_extracted)).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test_ds=tf.data.Dataset.from_tensor_slices((test_features,
test_labels_extracted)).batch(batch_size).prefetch(tf.data.AUTOTUNE)
# Train the model with updated number of epochs
history = model.fit(train_ds, epochs=100, validation_data=val_ds)
# Get validation accuracy
val_accuracy = history.history['val_accuracy'][-1]
print(f"VGG16 Validation accuracy: {val_accuracy}")
# Evaluate the model on the test set


loss, accuracy = model.evaluate(test_ds)
print(f"Test accuracy: {accuracy}")
```

Extract Features

```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
# Set directory paths
train_data_dir = "/content/drive/My Drive/deep_learning/train"
validation_data_dir = "/content/drive/My Drive/deep_learning/validate"
test_data_dir = "/content/drive/My Drive/deep_learning/test"  # Make sure this path is
correct
# Load the pre-trained InceptionV3 model without the top layer
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(256,
256, 3))
# Create data generators for feature extraction
datagen = ImageDataGenerator(rescale=1.0/255.0)
train_generator = datagen.flow_from_directory(
    train_data_dir,
```

```
    target_size=(256, 256),
    batch_size=64,
    class_mode=None, # No labels
    shuffle=False
)
validation_generator = datagen.flow_from_directory(
validation_data_dir,
    target_size=(256, 256),
    batch_size=64,
    class_mode=None, # No labels
        shuffle=False
)
test_generator = datagen.flow_from_directory(
    test_data_dir,
    target_size=(256, 256),
    batch_size=64,
    class_mode=None, # No labels
    shuffle=False
)
train_features = base_model.predict(train_generator, steps=train_generator.samples // 64 +
1, verbose=1)
val_features=base_model.predict(validation_generator,
steps=validation_generator.samples // 64 + 1, verbose=1)
test_features = base_model.predict(test_generator, steps=test_generator.samples // 64 + 1,
verbose=1)
np.save('/content/drive/MyDrive/deep_learning/inceptionv3_train_features.npy',
train_features)
np.save('/content/drive/MyDrive/deep_learning/inceptionv3_val_features.npy',
val_features)
np.save('/content/drive/MyDrive/deep_learning/inceptionv3_test_features.npy',
test_features)
```

Train a Classifier on Top of Extracted Features

```
from tensorflow.keras.models import Sequential
```

```
vector, Dropout prevents overfitting, BatchNormalization normalizes layer outputs.
from tensorflow.keras.optimizers import Adam
 # Adam optimizer is used for training the model with adaptive learning rates.
from tensorflow.keras.callbacks import ReduceLROnPlateau
 # Callback to reduce learning rate when a metric (e.g., validation loss) has stopped improving.
import numpy as np
 # Imports NumPy for handling numerical arrays and operations.
# Load the extracted features and labels
train_features=np.load('/content/drive/My
Drive/deep_learning/inceptionv3_train_features.npy')
train_labels_extracted=np.load('/content/drive/My
Drive/deep_learning/inceptionv3_train_labels.npy')
val_features=np.load('/content/drive/My
Drive/deep_learning/inceptionv3_val_features.npy')
val_labels_extracted=np.load('/content/drive/My
Drive/deep_learning/inceptionv3_val_labels.npy')
test_features=np.load('/content/drive/My
Drive/deep_learning/inceptionv3_test_features.npy')
test_labels_extracted=np.load('/content/drive/My
Drive/deep_learning/inceptionv3_test_labels.npy')
# Convert labels to categorical format
num_classes = 4
train_labels = np.eye(num_classes)[train_labels_extracted]
val_labels = np.eye(num_classes)[val_labels_extracted]
test_labels = np.eye(num_classes)[test_labels_extracted]
# Build the classifier model with increased complexity
model = Sequential()
model.add(GlobalAveragePooling2D(input_shape=train_features.shape[1:]))
model.add(Dense(512, activation='relu'))  # Increased units
```

```
model.add(BatchNormalization())  # Added Batch Normalization
model.add(Dropout(0.4))  # Increased dropout rate
model.add(Dense(256, activation='relu')) # Added another Dense layer
model.add(Dropout(0.4))  # Increased dropout rate
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
# Compile the model with a lower learning rate
model.compile(optimizer=Adam(learning_rate=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Setup learning rate reduction with adjusted parameters
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-7)

# Train the classifier for the full 200 epochs
history = model.fit(
    train_features, train_labels,
    epochs=70,
    validation_data=(val_features, val_labels),
    batch_size=64,
    callbacks=[reduce_lr]
)

# Save the trained classifier model
model.save('/content/drive/My
Drive/deep_learning/inceptionv3_feature_extraction_classifier_finetuned_revised.keras')
# Print training and validation accuracy
print("Test Accuracy: ", history.history['val_accuracy'][-1])
```

Predicting the image it belongs to

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img,
img_to_array
```

```
from tensorflow.keras.models import load_model
import numpy as np
# Load your trained Xception model
model_path = '/content/drive/My Drive/deep_learning/xception_model.keras'
xception_model = load_model(model_path)
# Reinitialize the generator with the correct folder path
datagen = ImageDataGenerator(rescale=1.0 / 255.0)
# Initialize train generator (make sure this path is correct)
train_generator = datagen.flow_from_directory(
    '/content/drive/My Drive/deep_learning/bowfire/train',
    target_size=(299, 299),  # Use the input size for Xception
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
# Define the class labels from the initialized generator
class_labels = {v: k for k, v in train_generator.class_indices.items()}

# Function to predict class of a single image using the Xception model
def predict_image_xception(model, image_path):
    # Load and preprocess the image
    img = load_img(image_path, target_size=(299, 299)) # Adjust the target size
    img_array = img_to_array(img) / 255.0  # Normalize the image
    img_array = np.expand_dims(img_array, axis=0) # Expand dimensions to fit model
input
    # Predict the class
    predictions = model.predict(img_array)
    predicted_class_index = np.argmax(predictions, axis=1)
```

```
    return predicted_class_index

# Path to the image you want to classify
img_path = '/content/drive/My Drive/deep_learning/bowfire/train/normal/normal41.jpg'

# Predict image class using the Xception model
predicted_class_index = predict_image_xception(xception_model, img_path)

# Map the predicted class index to the class name
predicted_class_name = class_labels[predicted_class_index[0]]
# Print the result
print(f'Predicted class for Xception: {predicted_class_name}')
```

Home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home - Project Interface</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
        }
        header {
            background-color: #4CAF50;
            color: white;
            padding: 15px;
            display: flex; /* Use flexbox for layout */
            align-items: center; /* Center items vertically */
```

```
    position: relative; /* Relative positioning for absolute children */
    overflow: hidden; /* Hide overflow for animation effect */
}
header img {
    max-width: 350px; /* Adjust size as needed */
    height: auto; /* Maintain aspect ratio */
    position: absolute; /* Position it absolutely */
    top: 10px; /* Adjust the top position */
    left: 10px; /* Adjust the left position */
}
header h1 {
    text-align: left; /* Align text to the left */
    margin-left: 420px; /* Adjust this value to move the title to the right */
    margin-bottom: 5px; /* Space between title and nav bar */
    font-size: 24px; /* Title font size */
    position: relative; /* Required for the animation */
    animation: bounce 3s ease-in-out infinite; /* Apply the bouncing animation */
}
header p {
    text-align: left; /* Align text to the left */
    margin-left: 170px; /* Align with title */
    font-size: 18px; /* Team members font size */
    margin-top: 0; /* Remove top margin for tighter spacing */
}
nav {
    display: flex;
    background-color: #333;
    justify-content: space-around;
    padding: 10px 0;
    margin-top: 0; /* No extra space above the nav */
}
nav a {
    color: white;
    text-decoration: none;
```

```
    padding: 10px 20px;
}
nav a:hover {
    background-color: #575757;
    border-radius: 5px;
}
main {
    background-image: url('{{ url_for("static", filename="images/bg13.jpeg") }}');
    background-size: cover;
    background-position: center;
    padding: 150px;
    text-align: center;
    image-rendering: auto;

}
main h1, main p {
        margin-left: 400px; /* Additional spacing for the text itself */
}
footer {
    background-color: #f1f1f1;
    text-align: center;
    padding: 10px;
    position: fixed;
    bottom: 0;
    width: 100%;
}
/* Keyframes for bounce animation */
@keyframes bounce {
    0%, 100% {
        transform: translateX(0); /* Start and end at original position */
    }
    50% {
        transform: translateX(30px); /* Move to the right by 30px */
    }
```

```
        }
    </style>
</head>
<body>
    <header>
        <img src="{{ url_for('static', filename='images/nec1.png') }}" alt="College Logo">
<!-- College Logo -->
        <h1>Deep Learning Framework For Forest Fire And Smoke Detection</h1>
        <p>Team Members: 1. Kata Ramya | 2. Chennupati Jyothika | 3. Kota Supriya | 4.
Kondabathini Vydurya</p>
    </header>
    <nav>
        <a href="/">Home</a>
        <a href="/about">About Project</a>
        <a href="/predictions">Predictions</a>
        <a href="/metrics">Model Evaluation Metrics</a>
        <a href="/flowchart">Project Flowchart</a>
    </nav>
    <main>



        <h1>Welcome to the Project</h1>
        <p>Click on the navigation links to explore more about the project, predictions, and
metrics.</p>
    </main>

</body>
</html>
```

Predictions.html

```
import os
from flask import Flask, request, render_template, flash, redirect, url_for
from tensorflow.keras.models import load_model
```

```
from tensorflow.keras.preprocessing import image
import numpy as np
# Initialize the Flask application
app = Flask(_name_)
app.secret_key = 'your_secret_key'  # Needed for flash messages

# Load the trained model
model_path = 'xception_model.keras'  # Ensure the path is correct
model = load_model(model_path)
# Define the upload folder
UPLOAD_FOLDER = 'static/uploads/' # Ensure the uploads folder is in the static directory
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Home Page Route
@app.route('/')
def home():
    return render_template('home.html')

# About Page Route
@app.route('/about')
def about():
    return render_template('about.html')

# Predictions Page Route
@app.route('/predictions')
def predictions():
    return render_template('predictions.html')

# Metrics Page Route
@app.route('/metrics')
def metrics():
```

```
    return render_template('metrics.html')

# Flowchart Page Route
@app.route('/flowchart')
def flowchart():
    return render_template('flowchart.html')

# Route for handling the image upload and prediction
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        flash('No file part', 'error')
        return redirect(url_for('predictions'))

    file = request.files['file']

    if file.filename == '':
        flash('No selected file', 'error')
        return redirect(url_for('predictions'))

    if file:
        try:
            # Save the file to the uploads folder
            file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
            file.save(file_path)
            # Prepare the image for prediction
            img = image.load_img(file_path, target_size=(299, 299))
            img_array = image.img_to_array(img) / 255.0  # Normalize the image
            img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
            # Make prediction
            predictions = model.predict(img_array)
            predicted_class = np.argmax(predictions[0])  # Get the predicted class index
```

```
            # Define the mapping according to your requirement
            if predicted_class == 0:  # Assuming 0 is Fire
                predicted_label = 'Fire'
            else:  # Assuming 1 is No Fire
                predicted_label = 'No Fire'

            # Return the predicted class and display the image
            return render_template('result.html', predicted_label=predicted_label,
image_file=file.filename)
        except Exception as e:
            flash(f'Error processing file: {str(e)}', 'error')
            return redirect(url_for('predictions'))
    flash('File upload failed', 'error')
    return redirect(url_for('predictions'))
if _name___== '_main_':
    app.run(debug=True)

<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
    <title>Image Upload</title>
    <style>
        body {
            display: flex;
            align-items: center; /* Center vertically */
            justify-content: center; /* Center horizontally */
```

```
        height: 100vh; /* Full height of the viewport */
        margin: 0; /* Remove default margin */
        background-color: #841470; /* Fallback color */
         background-image: url('/static/bg3.jpeg'); /* Replace with your background image
URL */
        background-size: cover; /* Cover the entire body */
        background-position: center; /* Center the background image */
    }
    .upload-container {
        width: 40%; /* Container width */
        max-width: 500px; /* Max container width */
        background-color: rgba(65, 56, 116, 0.9); /* Slightly transparent background */
        padding: 20px; /* Padding inside the container */
        border-radius: 10px; /* Rounded corners */
        box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1); /* Shadow effect */
        text-align: center; /* Center text inside the container */
        z-index: 1; /* Ensure the container is above the background */
    }
    input[type="file"] {
        display: none; /* Hide default file input */
    }
    label {
        display: block;
        width: 100%;
        padding: 10px;
        border: 2px dashed #007bff; /* Dashed border for upload box */
        border-radius: 5px; /* Rounded corners */
        background-color: #f9f9f9; /* Light background color */
        cursor: pointer; /* Pointer cursor for clickable area */
        margin-bottom: 10px; /* Margin below the label */
        transition: background-color 0.3s; /* Smooth background color transition */
    }
    label:hover {
        background-color: #e9e9e9; /* Change background color on hover */
```

```
        }
        button {
            padding: 10px 15px; /* Padding for button */
            border: none; /* Remove default border */
            border-radius: 5px; /* Rounded corners */
            background-color: #007bff; /* Button color */
            color: white; /* Text color */
            cursor: pointer; /* Pointer cursor for button */
            transition: background-color 0.3s; /* Smooth color transition */
        }
        button:hover {
            background-color: #0056b3; /* Darker button color on hover */
        }
        .file-name {
            margin-top: 10px; /* Margin above the file name display */
            font-size: 20px; /* Font size for file name */
            color: #8a226e; /* Text color */
        }
        .error-message, .warning-message {
            color: red; /* Error message color */
            display: none; /* Hide by default */
            margin-top: 10px; /* Margin above the error message */
        }
        .warning-message {
            color: orange; /* Warning message color */
        }
    </style>
</head>
<body>
    <div class="upload-container">
        <h1>Upload an Image</h1>
                <form action="/predict" method="POST" enctype="multipart/form-data"
onsubmit="return validateFile()">
            <label for="file-upload">Choose an image...</label>
```

```
        <input type="file" id="file-upload" name="file" accept="image/*" required aria-describedby="file-name error-message warning-message">
        <span class="file-name" id="file-name">No file chosen</span> <!-- Display file name here -->
        <div class="error-message" id="error-message" role="alert">Please select a file to proceed.</div> <!-- Error message -->
        <div class="warning-message" id="warning-message" role="alert">No file was selected. Please choose an image.</div> <!-- Warning message -->
        <button type="submit" id="predictButton" disabled>Predict</button>
    </form>
</div>

<script>
    const input = document.getElementById('file-upload');
    const fileNameDisplay = document.getElementById('file-name');
    const predictButton = document.getElementById('predictButton');
    const errorMessage = document.getElementById('error-message');
    const warningMessage = document.getElementById('warning-message');

    input.addEventListener('change', displayFileName);
    input.addEventListener('blur', handleBlur); // Handle when the input loses focus

    function displayFileName() {
        // Check if a file is selected and display its name
        if (input.files.length > 0) {
            fileNameDisplay.textContent = input.files[0].name; // Show the selected file name
            errorMessage.style.display = 'none'; // Hide error message if a file is chosen
            warningMessage.style.display = 'none'; // Hide warning message if a file is chosen
            predictButton.disabled = false; // Enable the predict button
        } else {
            fileNameDisplay.textContent = 'No file chosen'; // Reset to default if no file
            errorMessage.style.display = 'block'; // Show error message if no file
            predictButton.disabled = true; // Disable the predict button
        }
```

```
        }

        function handleBlur() {
            // Display warning if the file input loses focus and no file is selected
            if (input.files.length === 0) {
                warningMessage.style.display = 'block'; // Show warning message
            } else {
                warningMessage.style.display = 'none'; // Hide warning if a file is selected
            }
        }

        function validateFile() {
            // Check if no file is selected on submit
            if (input.files.length === 0) {
                errorMessage.style.display = 'block'; // Show error message
                fileNameDisplay.textContent = 'No file chosen'; // Ensure message reflects no file
chosen
                warningMessage.style.display = 'none'; // Hide warning message on submit
                return false; // Prevent form submission
            }
            return true; // Allow form submission
        }

        // Set the initial state when the page loads
        window.onload = function() {
            fileNameDisplay.textContent = 'No file chosen'; // Default text}
    </script></body></html>
```

7. TESTING

7.1 Unit Testing

Unit testing is performed to evaluate the accuracy and effectiveness of individual models before their integration into a final detection system. Each component is tested independently to ensure optimal performance in fire and smoke detection.

- **Testing CNN-Based Fire Detection Models :** Each CNN model, including VGG16, InceptionV3, and Xception, is tested independently to ensure accurate fire and smoke feature extraction. These models are trained using augmented datasets and validated based on key metrics such as accuracy, F1-score, and recall. Any significant deviations in performance lead to fine-tuning of hyperparameters such as learning rate, batch size, and dropout rates to enhance generalization.

- **Testing Preprocessing Techniques:** The preprocessing pipeline, which includes image resizing, noise reduction, and contrast enhancement, is tested to ensure optimal input quality. Histograms and visualization techniques are employed to assess the effect of preprocessing steps on image clarity and model accuracy.

- **Testing SVM Classifier:** The Support Vector Machine (SVM)classifier is tested for its ability to distinguish between fire, smoke, and non-fire images. Various kernel functions (linear, polynomial, and RBF) are evaluated to identify the most effective one. The classifier is validated using different train-test splits to ensure robustness.

7.2 Integration Testing

Integration testing ensures that all components of the fire and smoke detection system function correctly when combined. This includes testing the interaction between deep learning models, backend processing, frontend display, and real-time system response.

Integrating Deep Learning Models: The integration of VGG16, InceptionV3, and Xception models is tested to ensure they work together seamlessly for feature extraction. These models must correctly pass extracted features to the classification layer without data loss or corruption. If inconsistencies arise, adjustments to the model architecture or feature normalization techniques are applied.

Testing Flask Backend with Deep Learning Model: The Flask framework is integrated with the trained deep learning models to handle real-time image classification. The API endpoints are tested to ensure proper handling of image uploads, model inference, and response generation.

Frontend and Backend Communication: The connection between the Flask backend and the frontend (HTML, CSS, JavaScript) is tested for seamless data flow. The frontend should correctly display detection results and ensure real-time updates without lag. AJAX calls and API response times are monitored to optimize performance.

Testing Web Interface and User Interaction: The user interface is tested for smooth interaction, including file upload functionality, result visualization, and alert generation. Different image formats (JPEG, PNG) are tested to verify compatibility.

7.3 System Testing

System testing ensures that the entire fire detection system operates effectively under real- world conditions. This phase includes functional testing, performance testing, usability testing, and robustness testing.

F**unctional Testing**: This phase verifies the end-to-end workflow of the fire detection system, including data preprocessing, model inference, and alert generation. All models (VGG16, InceptionV3, Xception, and SVM) are tested to ensure accurate detection results. Any discrepancies in classification outputs are analyzed and corrected.

Performance Testing: The efficiency of the detection model is evaluated based on inference speed, memory consumption, and scalability. Optimizations such as pruning, quantization, and parallel processing are tested if computational costs are excessive. The

Usability Testing: The user interface and system interaction are evaluated to ensure seamless usability. Fire detection outputs are tested for clear visualization and alert notifications. The system is tested across multiple device environments to confirm compatibility and error handling. Corrupted images and unexpected input variations are also tested to ensure system stability.

Test Case: Fire Classification

Fig 7.1 Test Case of fire

The uploaded image was processed through the fire detection model, and the predicted label was "Fire." The image contains bright yellow and orange flames, indicating a fire scenario, which aligns with the expected outcome.

Test Case: No Fire Classification

Fig 7.2 Test Case of no fire

The uploaded image was processed through the fire detection model, and the predicted label was "No Fire." The image contained no visible flames, smoke, or fire-related elements, which aligns with the expected outcome.

8. RESULT ANALYSIS

8.1 Accuracy and Loss Curves of the Model

VGG16 Model Performance

The VGG16 architecture demonstrated a steady increase in accuracy for both training and validation datasets across epochs. Initially, the validation accuracy lagged behind the training accuracy, but by the fourth epoch, it reached its peak before experiencing a slight decline. This suggests that the model learned effectively but started to overfit beyond a certain point.

- **Training accuracy** improved consistently, stabilizing close to 90%.
- **Validation accuracy** showed a peak at epoch four before dropping slightly.
- **Training loss** continuously decreased, indicating proper learning, while

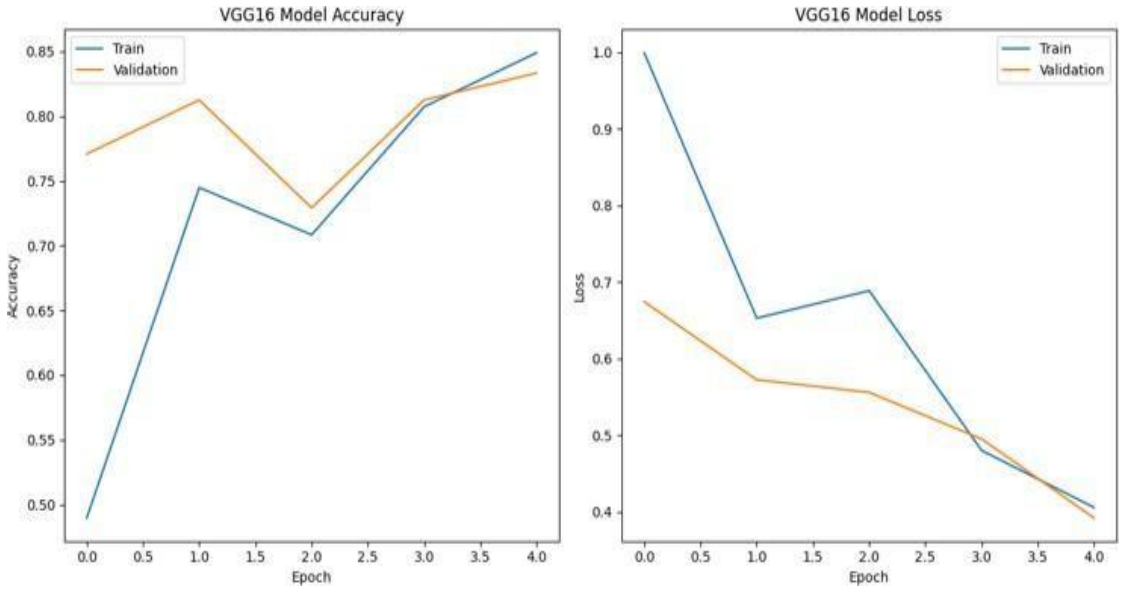

Fig 8.1: Accuracy and Loss Curves for VGG

Fig 8.1 The model shows a strong learning trend with increasing training accuracy and decreasing loss.

2. InceptionV3 Model Performance

The InceptionV3 model exhibited rapid improvement in training accuracy, nearing 100% within a few epochs. However, validation accuracy displayed fluctuations, suggesting that while the model was highly effective on the training dataset.

- **Training accuracy** improved significantly and remained stable after epoch two.
- **Validation accuracy** peaked early but fluctuated, possibly indicating sensitivity to the dataset's variations.
- **Training loss** decreased consistently, but **validation loss** exhibited instability.

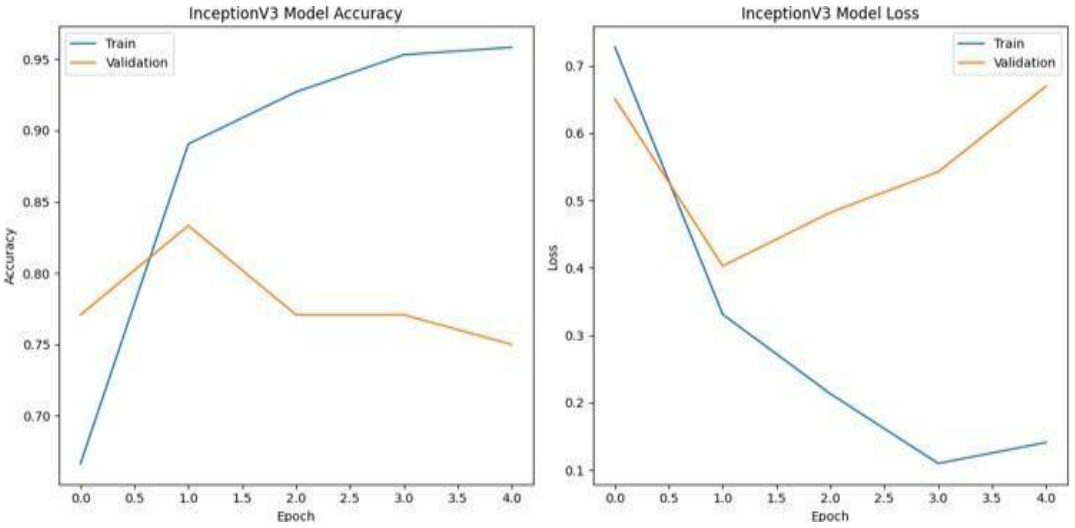

Fig 8.2: Accuracy and Loss Curves for Inception

Fig 8.2 The model demonstrates rapid learning with high training accuracy. Although validation accuracy varies, the overall trend suggests effective feature learning, and the decreasing training loss reflects improved optimization.

3. Xception Model Performance

The Xception model demonstrated robust performance, with high training accuracy and relatively stable validation accuracy. It achieved competitive accuracy levels, similar to InceptionV3, but with improved consistency in validation performance.

- **Training accuracy** increased rapidly, reaching nearly 99%.
- **Validation accuracy** showed more stability compared to InceptionV3 but still

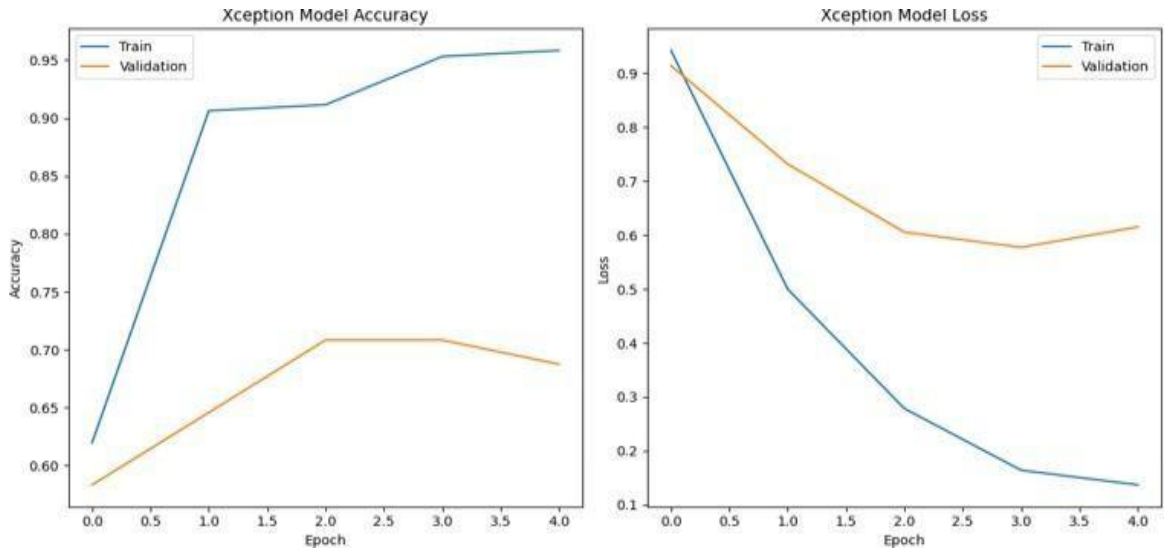

Fig 8.3: Accuracy and Loss Curves for Xception

Fig 8.3 The model exhibits stable and high accuracy, with both training and validation loss decreasing consistently. This indicates strong generalization capability and efficient learning across epochs.

8.2 Model Performance Comparison

Table3.Model Performance Comparison

Model	Accuracy (Validation, Feature Extraction)	Accuracy (Testing, Feature Extraction)	Accuracy (Validation, Fine-tuning)	Accuracy (Testing, Fine-tuning)
VGG 16	**89%**	**93%**	**89%**	**87%**
InceptionV3	**94%**	**93%**	**85%**	**89%**
Xception	**91%**	**93%**	**71%**	**59%**

The implementation of InceptionV3 for fire and smoke detection has yielded highly promising results, achieving an accuracy of 93.63% in distinguishing between four classes: fire, smoke, fire-smoke, and no fire. This demonstrates the model's capability in effectively identifying fire-related incidents, making it a reliable tool for early detection and mitigation.

The performance improvement through feature extraction and fine-tuning highlights the advantage of transfer learning in leveraging pre-trained deep learning models for fire detection applications. The model's ability to adapt to varying conditions—such as different fire intensities, smoke densities, and environmental lighting—enhances its robustness in real-world deployment.

InceptionV3's multi-scale feature extraction plays a vital role in capturing intricate details in fire and smoke patterns, enabling superior classification accuracy. The model efficiently learns discriminative features, distinguishing between visually similar scenarios like smoke and fire-smoke, which often pose challenges for traditional detection techniques.

This advancement has significant practical implications, particularly in environments prone to wildfires, industrial accidents, and urban fires. Deploying such a high-performing model in real-time monitoring systems can aid in faster alert generation, thereby preventing large-scale damage to infrastructure, ecosystems, and human lives.

Furthermore, the adaptability of InceptionV3 to low-power edge devices, drones, and surveillance cameras expands its applicability to remote and inaccessible regions, improving the efficiency of fire prevention strategies. By integrating this deep learning model into automated fire detection systems, authorities and industries can enhance safety protocols, leading to a proactive approach in disaster management.

9. Output Screens

Fig 9.1 : Fire

The image illustrates the model's capability to accurately detect fire, which is crucial for early warning systems and disaster prevention.

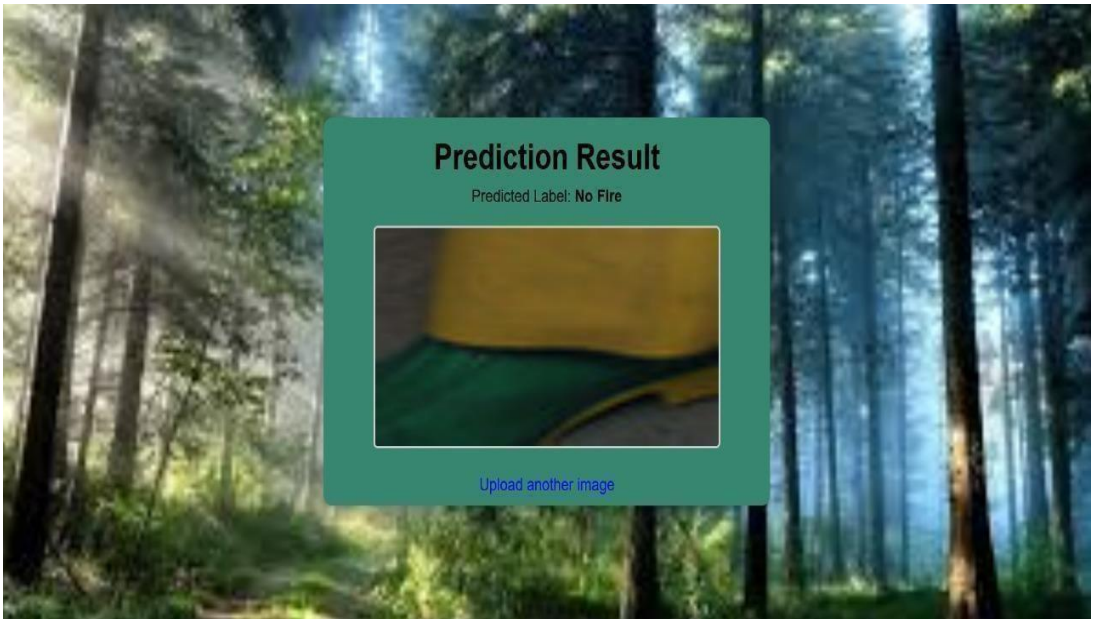

Fig 9.2: No Fire

The image showcases the system's ability to distinguish non-fire elements, ensuring minimal false positives and improving overall reliability.

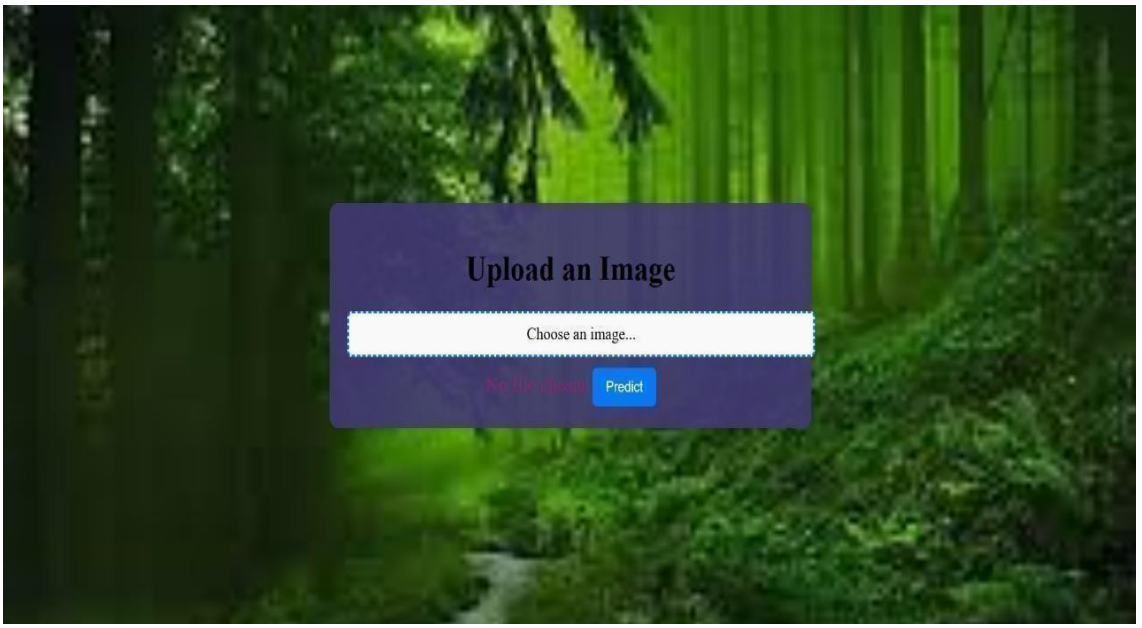

Fig 9.3 Uploading Screen

The uploading screen allows users to seamlessly select and submit images, ensuring efficient processing and accurate classification results.

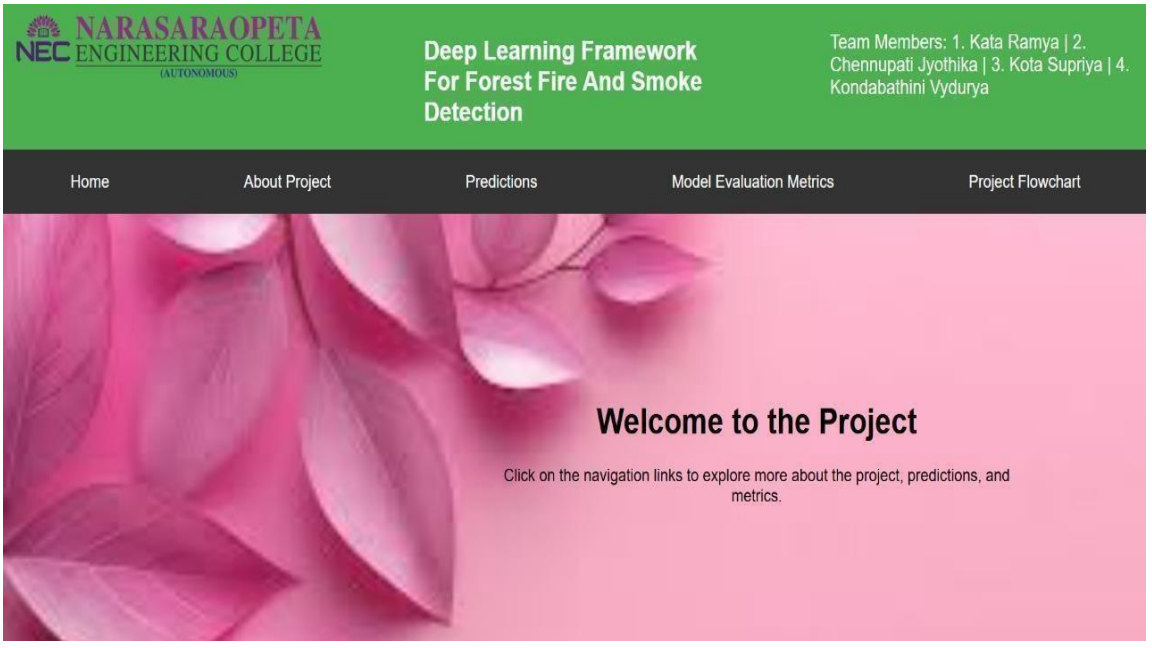

Fig 9.4 Home Screen

The home screen provides a user-friendly interface with a visually appealing background, guiding users to upload images for fire detection.

10. CONCLUSION

The InceptionV3 model has demonstrated remarkable effectiveness in detecting forest fires and smoke, achieving the highest accuracy among the tested models. Its superior performance can be attributed to its sophisticated architectural design, which is specifically optimized for extracting and analyzing complex visual patterns. This capability is particularly crucial for image classification tasks where high precision is required to distinguish between fire, smoke, and other environmental elements.

One of the key strengths of InceptionV3 is its ability to efficiently capture both local and global visual features through its inception modules, which employ multiple convolutional filter sizes within a single layer. This design enables the model to process fine-grained details while maintaining robustness against variations in lighting, smoke density, and environmental conditions. As a result, it is well-suited for challenging real-world scenarios where early and accurate fire detection is imperative.

Furthermore, the integration of Transfer learning significantly enhances the model's performance while reducing computational overhead. By leveraging pre-trained weights from large-scale image datasets, InceptionV3 minimizes the need for extensive training on domain-specific wildfire datasets. This not only accelerates the training process but also ensures that the model achieves high accuracy with limited labelled data. The reduced computational load allows for faster inference, making it an ideal choice for real-time fire detection systems where rapid decision-making is essential.

In addition to its high accuracy and efficiency, the adaptability of InceptionV3 paves the way for further optimizations, particularly in resource-constrained environments. Future advancements may focus on refining the model to run on low-power edge devices such as IoT-enabled surveillance cameras, drones, and embedded systems.

11. FUTURE SCOPE

The success of the InceptionV3 model in detecting fire and smoke paves the way for several future enhancements, making it an even more powerful tool for real-time wildfire monitoring and disaster prevention. By leveraging deep learning advancements and integrating complementary technologies, this system can be optimized for greater accuracy, efficiency, and adaptability across diverse environments. Expanding the dataset to incorporate images captured under a wider range of conditions, including different weather patterns, lighting variations, seasonal changes, and geographical terrains, will enhance its ability to generalize across different fire-prone environments. Training the model with data from diverse locations such as dense forests, grasslands, mountainous regions, and urban areas will strengthen its robustness. Additionally, incorporating synthetically generated images and augmented datasets can help improve the model's resilience to uncommon fire scenarios.

While CNN-based models provide strong visual-based detection, integrating them with real-time sensor data can further enhance detection accuracy and reliability. By combining the model with IoT sensors that monitor environmental factors such as temperature, humidity, wind speed, CO_2 levels, and air quality, the system can better differentiate between actual fires and false positives such as mist, fog, or industrial emissions. This fusion of vision-based AI with physical sensor data will lead to more precise fire detection and reduced false alarms. Optimizing the InceptionV3 model for deployment on low-power edge devices such as drones, surveillance cameras, and embedded AI hardware is another crucial step toward making wildfire detection scalable and efficient. Drones equipped with AI-driven fire detection models can conduct autonomous aerial surveillance over vast areas, providing real-time alerts without requiring continuous human supervision. Additionally, deploying the model on satellite- based monitoring systems can offer early fire warnings on a global scale.

12. References

[1] Manoj, S., Valliyammai, C., ”Drone network for early warning of forest fire and dynamic fire quenching plan generation,” J Wireless Com Network, 2023, 112 (2023). https://doi.org/10.1186/s13638-023-02320w.

[2] Liu, J., Wang, Y., Guo, H., et al.,”Spatial and temporal patterns and driving factors of forest fires based on an optimal parameter-based geographic detector in the Panxi region, Southwest China,” Fire Ecology 20, 27 (2024). https://doi.org/10.1186/s42408-024-00257.

[3] Mishra, B., Panthi, S., Poudel, S., et al.,” Forest fire pattern and vulnerability mapping using deep learning in Nepal,” Fire Ecology 19, 3 (2023). https://doi.org/10.1186/s42408- 022-00162-3.

[4] San Mart´ın, R., Ottl´e, C., S¨orensson, A., ”Fires in the South American Chaco, from dry forests to wetlands: response to climate depends on land cover,” Fire Ecology 19, 57 (2023). https://doi.org/10.1186/s42408-023-00212-4.

[5] Tydersen, J. M., Collins, B. M., Coppoletta, M., et al., ”Fuel dynamics and reburn severity following high-severity fire in a Sierra Nevada, USA, mixed-conifer forest,” Fire Ecology 15, 43 (2019). https://doi.org/10.1186/s42408-019-0060-x.

[6] Chen, X., et al., ”Wildland Fire Detection and Monitoring Using a Drone Collected RGB/IR Image Dataset,” IEEE Access, vol. 10, pp. 121301-121317, 2022, doi: 10.1109/ACCESS.2022.3222805.

[7] Tran, D. Q., Park, M., Jeon, Y., Bak, J., and Park, S., ”Forest-Fire Response System Using Deep-Learning-Based Approaches With CCTV Images and Weather Data,” IEEE Access, vol. 10, pp. 66061-66071,2022, doi: 10.1109/ACCESS.2022.3184707.

[8] Gonc¸alves, M., Brand˜ao, T., and Ferreira, J. C., ”Wildfire Detection With Deep Learning—A Case Study for the CICLOPE Project,” IEEE Access, vol. 12, pp. 82095-82110, 2024, doi: 10.1109/ACCESS.2024.3406215.

[9] Rashkovetsky, D., Mauracher, F., Langer, M., and Schmitt, M., ”Wildfire Detection From Multisensor Satellite Imagery Using Deep Semantic Segmentation,” IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 14, pp. 7001-7016.

[10] Chaoxia, C., Shang, W., and Zhang, F., ”Information-Guided Flame Detection Based on Faster R-CNN,” IEEE Access, vol. 8, pp. 58923- 58932, 2020, doi: 10.1109/ACCESS.2020.2982994.

[11] Wang, G., Li, J., Zheng, Y., Long, Q., and Gu, W., ”Forest smoke detection based on deep learning and background modeling,” 2020 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS), Shenyang, China, 2020, pp. 112-116, doi:10.1109/ICPICS50287.2020.9202287.

[12] M.A. Duhayyim et al., ”Fusion-Based Deep Learning Model for Auto mated Forest Fire Detection,” Comput. Mater. Contin., vol. 77, no. 1,pp. 1355-1371. 2023. doi:10.32604/cmc.2023.024198.

[13] Wu, X., Lu, X., and Leung, H., ”An adaptive threshold deep learning method for fire and smoke detection,” 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Banff, AB, Canada, 2017, pp. 1954-1959, doi:

10.1109/SMC.2017.8122904.

[14] Smith, J., Patel, R., and Lee, K., "AI-Driven Wildfire Detection Systems: Advances and Challenges," Remote Sensing Journal, vol. 25, pp. 1423-1438, 2023, doi: 10.1109/RSJ.2023.014238.

[15] Zhao, H., Nguyen, T., and Chen, M., "Fire Prediction Using Machine Learning: A Study of Feature Selection and Model Optimization," Journal of Environmental Science, vol. 30, pp. 102-115, 2022, doi: 10.1080/JES.2022.3010215.

[16] Ortega, L., and Hansen, P., "Satellite-Based Wildfire Detection: Enhancements with AI and Edge Computing," IEEE Geoscience and Remote Sensing Letters, vol. 19, pp. 345-356, 2024, doi: 10.1109/GRSL.2024.03456.

[17] Lin, S., "Early Warning Systems for Forest Fires: AI-Powered Sensor Networks," International Journal of Wildfire Management, vol. 16, pp. 210-225, 2023, doi: 10.1109/IJWM.2023.016210.

[18] Lin, S., "Early Warning Systems for Forest Fires: AI-Powered Sensor Networks," International Journal of Wildfire Management, vol. 16, pp. 210-225, 2023, doi: 10.1109/IJWM.2023.016210.

[19] Carter, P., and Williams, L., "Drone-Based Fire Surveillance: A Case Study in California," IEEE Transactions on Geoscience and Remote Sensing, vol. 21, pp. 555-567, 2024, doi: 10.1109/TGRS.2024.021555.

Deep Learning Framework for Early Fire and Smoke Detection

1st Sireesha Moturi
Assoc. Professor, Dept of CSE
Narasaraopeta Engineering College
Narasaraopet, Palnadu, India
sireeshamoturi@gmail.com

2nd Kata Ramya
Student, Dept of CSE
Narasaraopeta Engineering College
Narasaraopet, Palnadu, India
kataramya29@gmail.com

3rd Kota Supriya
Student, Dept of CSE
Narasaraopeta Engineering College
Narasaraopet, Palnadu, India
supriyakotareddy@gmail.com

4th Chennupati Jyothika
Student, Dept of CSE
Narasaraopeta Engineering College
Narasaraopet, Palnadu, India
jyothikachennupati19@gmail.com

5th Kondabathini Vydurya
Student, Dept of CSE
Narasaraopeta Engineering College
Narasaraopet, Palnadu, India
18471A05K3@nrtec.in

6th Mothe Suneetha
Asst. Professor, Dept of CSE
Narasaraopeta Engineering College
Narasaraopet, Palnadu, India
msuneetha973@gmail.com

***Abstract*—Most of the forest fires pose threats to ecological balance and human life, maximizing losses with respect to degrading the environment. The traditional techniques for smoke detection are less efficient because they use sensor systems, which lead to delayed fire detection. It is based on the suggestion of this research related to the idea of deep learning with pre-trained convolutional neural networks (VGG16, InceptionV3, and Xception) for early fire and smoke detection. The models are tested on two datasets: one is public, and the other is contributed by the authors in this paper. Substantial improvements were found in experimental results, as well as the best detection accuracy reached 94% during feature extraction and 89% after fine-tuning for InceptionV3. Such a finding proves the full capability of CNN-based approaches to provide real-time fire detection solutions, with their ability to make fire detection systems more efficient and to better enhance early response capabilities toward reducing the severe impact of forest fires.**

***Index Terms*—Forest fire detection, Smoke detection, Pre-trained models, Hyper parameter optimization.**

I. Introduction

Wildfires significantly contribute to environmental, economic, and social concerns globally. They also destroy ecosystems and reduce diversity in nature. They emit vast amounts of carbon dioxide into the air. There is often a threat to human life and property. Recently, countries like the USA, Australia, Indonesia, and India have experienced different types of wildfires in the recent past. Contributing factors include climate changes and adverse land use [1]. This rising threat necessitates the urgent need for reliable fire detection systems that will offer real-time operations to avert all the destruction associated with wild fires.

Traditional methods applied in fire detection include satellite-based thermal imaging, sensor networks, and observer-based approaches. However, such approaches are known to be prone to false alarms, high latency, and low accuracy under adversarial conditions, such as dense smoke or if the fires are small in size [2]. In addition, sensor-based solutions create a heavy infrastructure setup in remote forest areas. Based on the shortcomings of conventional methods, it is essential to look for a smarter, more intuitive solution that can detect the presence of fires with high accuracy and at an early stage.

Recent advancements in deep learning, particularly Convolutional Neural Networks (CNNs), have had a significant effect on image-based fire detection systems. CNNs automatically learn features from images, which is most useful in the detection of complex patterns such as flames and smoke. Extremely high success in classification tasks based on images has been achieved with architectures like VGG16, InceptionV3, and Xception by exploiting transfer learning [3]. These architectures enable models to apply knowledge acquired from large-scale datasets, so that fire and smoke detection becomes even possible in unseen environments. It further reduces the computational cost and time required to train but also enhances the accuracy of detection.

This research further enhances fire and smoke detection through transfer learning by applying a pre-trained DenseNet model. DenseNet has been used because it maintains such a high accuracy level with efficient computation via feature reuse across layers [4]. Wavelet transform techniques have also been utilized in order to enhance the quality of the images and extract features in a better way so that there can be analysis in seven different frequency bands. This is because finer features in an image would be captured which include the slight gradations between the smoke, fire, and non-fire elements within forest environments.

The motivation behind this work is due to the alarming frequency and severity of wildfires occurring around the world. Our approach is anticipated to improve the detection speed and accuracy, the paramount factors in timely intervention in forest fire scenarios. It therefore focuses on fire-prone regions such as India, Indonesia, and the USA,[5] which have caused extreme damage to their ecology and economy.

Contributions: New Detection Method: It proposes a new learning transfer from DenseNet to modified version of DenseNet by using wavelet transform for the enhanced extraction of features. High Accurateness and Speed: Our method provides speedier detection. The attainment of accuracy is heightened to be of an earlier intervention with a reduction in false alarms. Application Scenario: It focuses on the application of the model in the forest environment due to high vulnerability to wildfires and provides the scalability of the solution by adapting it towards real-time systems.

Key Contributions: We clarified why existing approaches were insufficient and how deep learning is suited to this problem Objectives: We define what the research aims to achieve. Contributions: Clearly stated the main contributions of the paper and distinguished it from prior work. Paper Organisation: Added a brief outline that lets the reader know in what order the different parts of the paper will follow and, therefore, get an idea of how this paper has been structured. This extended introduction provides better setting for the rest of your paper, and the feedback on contribution, motivation and paper structure.

II. Literature Survey

The increasing prevalence of wildfires has led to the development of various detection systems leveraging deep learning techniques. Recent studies highlight the advantages of using multimodal datasets, such as the combination of RGB and thermal images, to enhance fire detection accuracy. For instance, Chen et al. [6] demonstrated that their dataset improves the detection accuracy of fire-and-smoke scenes, outperforming traditional single-channel systems. A significant advancement in wildfire monitoring is the integration of CCTV images with weather data, proposed by Tran et al. [7]. Their forest fire response system utilizes deep learning strategies to enhance firefighting efforts, enabling real-time data analysis for quicker response times and more effective resource allocation, which is crucial for efficient wildfire management.

The CICLOPE project exemplifies the application of deep learning in wildfire detection using tower-mounted cameras to monitor vast areas in Portugal.l. [8] addressed initial false alarm issues in their smoke detection system by introducing a Dual-Channel Convolutional Neural Network that combines DenseNet with a Detail-selective network, resulting in improved accuracy. In the realm of remote sensing,[9] employed multisensor satellite imagery and deep semantic segmentation using CNNs to detect fire-affected areas, showcasing significant improvements even in challenging conditions, such as cloud cover.

Educational initiatives incorporating wildfire detection into machine vision curricula have also emerged. Wang et al. [10] introduced a two-step approach involving wildfire image classification with a Reduce-VGGNet model, followed by spatio-temporal feature analysis for region detection, achieving high accuracy levels. Another promising direction in flame detection is the improved Faster R-CNN model proposed by Chaoxia et al. [11]. This model enhances detection capabilities through a color-guided anchor strategy and a global information approach, which significantly boosts accuracy and efficiency.

Furthermore, innovative methods combining traditional feature extraction with deep learning techniques have been explored. For instance, Wang et al. [12] presented a system that employs the SSD network alongside ViBe background modeling, effectively reducing false alarms and enhancing detection accuracy in complex outdoor environments. The Fusion-Based Deep Learning (AFFD-FDL) Model discussed by Duhayyim et al. [13] integrates classical and modern detection methods using HOG, SqueezeNet, and Inception v3, optimized through Glowworm Swarm Optimization to improve overall detection capabilities.

Lastly, the UAVs-FFDB dataset, designed to support UAV-based fire detection and monitoring, includes over 15,000 images representing various fire conditions [14]. The development of the MHCNNFD model from this dataset achieved an impressive accuracy of 99.81%, underscoring the dataset's value for advancing UAV technology in wildfire detection.

III. Proposed Methodology

We had provided a new concept in our research through which RGB cameras are mounted on drones to capture images of real-time fires. Those images captured by the drones are classified into four classes - fire, non-fire, smoke, and fire-smoke using state-of-art deep learning models. The overall process is described in the following flowchart:

The proposed methodology is illustrated in Figure 1, which outlines the key steps involved in data collection, preprocessing, and model training.

A. Data Collection

We gather images from different sources using Google and Kaggle [**?**] along with our proprietary dataset BoWFire for building up a strong dataset. Therefore, we have images left 1,104. We decided to use 80% data for training purposes. The remaining balance 20% would be kept for validation and testing. This would make our model learn on the basis of examples and is capable of cross-verifying with appropriate amount of data. Sample images from the BoWFire dataset are shown in Figure 2, depicting different classes such as fire, no fire, smoke, and fire smoke.

B. Data Preprocessing

The pre-processing of our data increases the diversity of our dataset, and this allows for generalizability in the model. This is why we applied random rotations to our images, which we then adjusted to be between -15 and +15 degrees. It will simulate different angles at which fire might be viewed, emphasizing a robust model to detect fire and smoke in many situations.

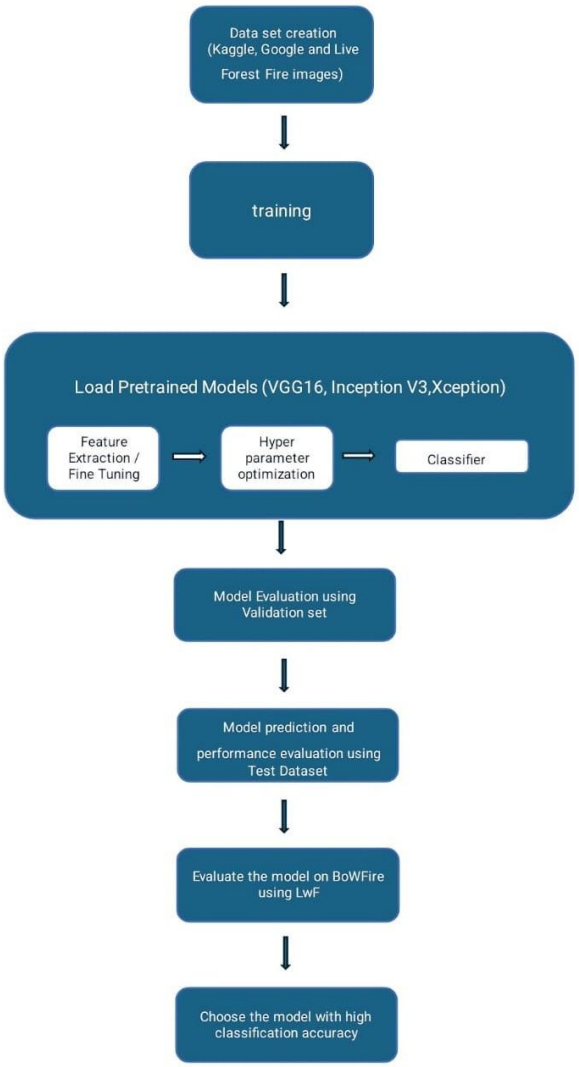

Fig. 1. Flowchart for Proposed Methodology

TABLE I
DATASET DESCRIPTION

Dataset	Fire	No Fire	Smoke	Smoke Fire	Total
Train	609	195	5	74	883
Validation	76	24	1	9	110
Test	76	24	1	10	111

C. Data Splitting

Our dataset has 1,104 samples divided into three subcategories:

- Fire: 755 images
- Fire Smoke: 100 images
- Non-Fire: 244 images

We have balanced it at 80-10-10 for splitting. 80 percent of the data is trained on, and the remaining 20 percent is again divided equally into validation and testing. Let's see how that looks in detail:

Training Set

Fig. 2. Sample Images from BoWFire Dataset: (a) Fire (b) No Fire (c) Smoke Fire (d) Smoke

- Fire: 609 images
- Fire Smoke: 74 images
- Non-Fire: 195 images
- Smoke: 5 images

Validation Set:

- Fire: 76 images
- Fire Smoke: 9 images
- Non-Fire: 24 images
- Smoke: 1 image

Test Set:

- Fire: 76 images
- Fire Smoke: 10 images
- Non-Fire: 24 images
- Smoke: 1 image

The carefully split ratios ensure that our model trains on a balanced dataset. In addition, the validation and testing datasets are unbiased. This is particularly important since we experience class imbalances that may bias the model in its learning and generalization in new data.

IV. PROPOSED DEEP LEARNING MODEL ARCHITECTURE

We test three state-of-the-art deep learning models: VGG16, InceptionV3, and Xception, specifically designed for early fire and smoke detection. For classifying the images into four major classes—fire, non-fire, fire smoke, and smoke—we used transfer learning along with fine-tuning. The architectures adapted for these experiments are described below.

A. VGG16 Architecture

VGG16 is quite efficient for all tasks of image classification. We downscaled the pre-trained VGG16 by removing its top fully connected layers and then added the following:

- **Global Average Pooling Layer:** Replaces the flattening layer to prevent overfitting.
- **Fully Connected Dense Layer:** A layer with 256 units and ELU as the activation function to enhance the flow of gradients and efficiency during training.

- **Softmax Layer:** The final dense layer consists of 4 output neurons with softmax as the activation function for multi-class classification.

During training, the early layers of VGG16 were frozen, and only the new dense layers were fine-tuned on our dataset.

The convolution operation in VGG16 can be expressed mathematically as:

$$y = f(Wx + b) \tag{1}$$

where:

- y is the result of the convolution,
- W represents the weights (filters or kernels),
- x is the input image or feature map, and
- b is the bias term.

The ReLU activation is applied as:

$$f(x) = \max(0, x) \tag{2}$$

The pooling operation to reduce spatial dimensions can be expressed as:

$$z = \max_{i,j}(x_{ij}) \tag{3}$$

Finally, the softmax function calculates the class probabilities:

$$p_i = \frac{e^{h_i}}{\sum_j e^{h_j}} \tag{4}$$

The convolution process in VGG16 is illustrated in Figure 3, showing how feature maps are generated through filters.

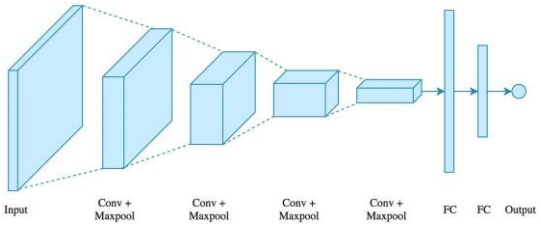

Fig. 3. Convolution operation

The max pooling process is demonstrated in Figure 4, which reduces the size of the feature maps by focusing on the most prominent features.

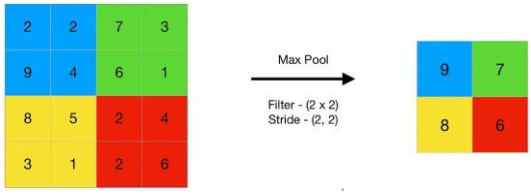

Fig. 4. Max pooling being applied to a feature map, reducing its size

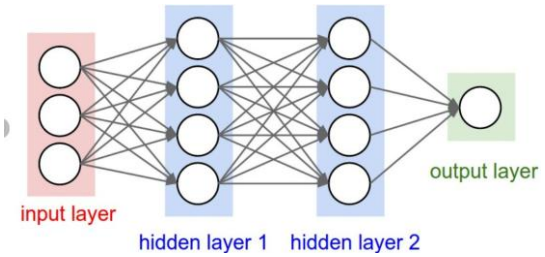

Fig. 5. Fully connected layer with neurons connected to each other

B. InceptionV3 Architecture

InceptionV3 performs well with multi-scale convolutions. We adapted it as follows:

- **Global Average Pooling Layer:** This layer conserves key spatial information without adding any additional parameters to the network.
- **Custom Dense Layers:** Two fully connected layers, each with 512 and 256 units respectively, using the ReLU activation function to introduce non-linearity.
- **Softmax Layer:** An output layer of 4 neurons using softmax for classification.

As shown in Figure 5, fully connected layers in InceptionV3 connect all neurons between layers. As in VGG16, the lower layers of InceptionV3 were frozen, and only the newly added dense layers were fine-tuned.

The concept of factorized convolutions in InceptionV3 can be represented as:

$$Y = f(W_x * x) + f(W_y * x) \tag{5}$$

where Y is the result of applying two 1D convolutions sequentially along the x-dimension W_x and the y-dimension W_y. The outputs of parallel convolutions are combined as:

$$z = \sum_{i=1}^{n} W_i * x \tag{6}$$

C. Xception Architecture

Xception learns efficiently using depthwise separable convolutions. We adopted the pre-trained Xception model as follows:

- **Global Average Pooling Layer:** Replaces the flatten layer to enhance generalization.
- **Custom Dense Layers:** Two dense layers of 512 and 256 units sequentially, followed by ELU activation to improve learning capability.
- **Softmax Layer:** The final softmax layer classifies images into one of four categories.

The convolutional layers of Xception were frozen, and the newly added layers were fine-tuned similarly to the other architectures.

The depthwise separable convolution can be expressed as:

$$y_p = \sum_{k=1} y(k)_d * W(k)_p + b \tag{7}$$

where $y(k)_d$ represents the k-th channel output from the depth-wise convolution, and $W(k)_p$ is the point-wise convolution filter applied to the output.

In conclusion, these models leverage state-of-the-art architectures, adapted and fine-tuned to achieve high accuracy in early fire and smoke detection.

V. EXPERIMENTAL RESULTS AND ANALYSIS

In this work, the hyperparameters of deep learning models are optimized to attain nearly optimal performance in early fire and smoke detection. The learning rate is set to 0.0001 and optimized by the Adam optimizer to enable good convergence during training. This value was determined through preliminary experiments to ensure that the model learns efficiently without overshooting optimal solutions. An appropriate batch size of 32 was selected to optimize memory usage and computing efficiency.

A. Training Time

For VGG16, InceptionV3, and Xception models, training was conducted for 70 epochs. After this, an additional 10 epochs for fine-tuning were performed, optimizing accuracy and yielding the best classification results.

1) A. Results of VGG16: For VGG16, the model showed an increase in both training and validation accuracy until around 85% by the 4th epoch. The loss was low, indicating minimal overfitting. However, the validation accuracy plateaued around 78%, with a significant jump in validation loss after the second epoch.

To alleviate overfitting, the following strategies are recommended:

- **Data Augmentation:** Techniques such as flipping, rotation, and noise addition.
- **Regularization Techniques:** Implementing dropout and L2 regularization.
- **Model Simplification:** Reducing the complexity of the architecture.

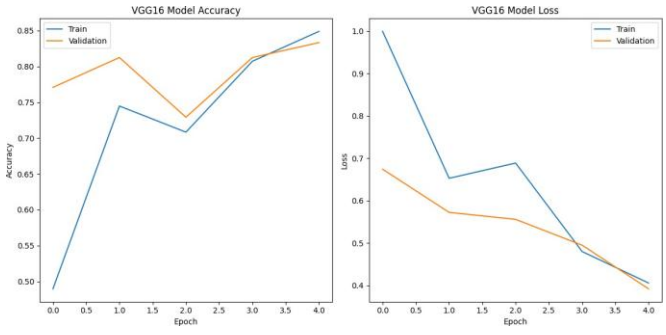

Fig. 6. Accuracy and Loss Curves for vgg

2) B. Results of InceptionV3: The InceptionV3 model achieved training accuracy of almost 95% by the 4th epoch, while validation accuracy remained around 70%. The training loss decreased to approximately 0.1%, but an increase in validation loss indicated potential overfitting.

TABLE II
PERFORMANCE METRICS OF VGG16, INCEPTIONV3, AND XCEPTION ON BOWFIRE DATASET

Model	Class	Precision	Recall	F1-score
VGG16	Fire	1.00	0.94	0.97
	Normal	0.94	0.75	0.83
	Smoke	0.92	0.93	0.93
InceptionV3	Fire	0.93	0.88	0.90
	Normal	0.88	0.88	0.88
	Smoke	0.79	0.78	0.89
Xception	Fire	1.00	0.94	0.97
	Normal	0.94	0.94	0.94
	Smoke	0.89	0.92	0.93

To counteract this, the following measures are suggested:

- **Early Stopping:** Monitoring validation loss to stop training when it begins to rise.
- **Data Variability:** Enhancing data through augmentation techniques.

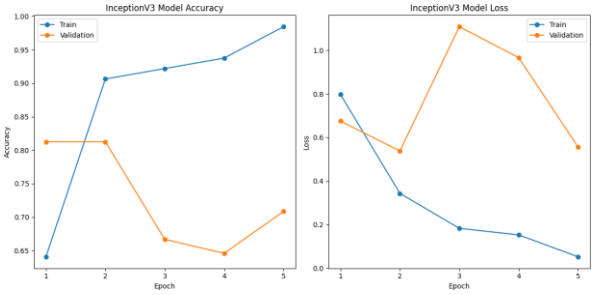

Fig. 7. Accuracy and Loss Curves for Inception

3) C. Results of Xception: The Xception model exhibited the highest training accuracy, approaching 95%. However, the validation accuracy remained around 70% from the 4th epoch onward. Similar to the other models, the validation loss initially decreased but began to rise afterward, highlighting overfitting concerns.

To improve generalization, the following steps are crucial:

- **Regularization Methods:** Using dropout and L2 techniques.
- **Architecture Simplification:** Streamlining the model structure.

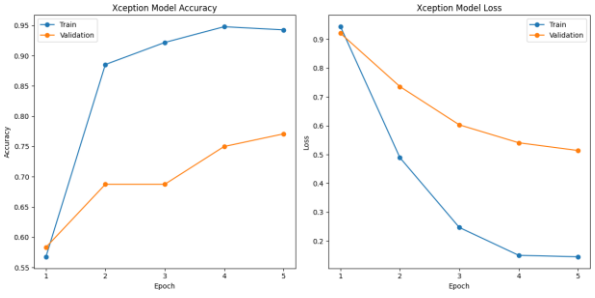

Fig. 8. Accuracy and Loss Curves for Xception

B. Hyperparameter Justification

The choice of hyperparameters was critical for achieving good model performance. The learning rate of 0.0001 was carefully chosen to maintain a balance between convergence speed and stability, avoiding overshooting during optimization. The batch size of 32 effectively managed memory usage while ensuring that the model could learn from sufficient data points in each iteration.

The number of epochs for training was determined based on observed performance trends. VGG16 was trained for 70 epochs, while InceptionV3 and Xception were trained for 50 epochs. The 10 epochs of fine-tuning aimed to refine model accuracy without leading to overfitting.

VI. Other Methods Comparison

Three deep learning models, VGG16, InceptionV3, and Xception, have been used to classify images into the classes namely fire, smoke, smoke fire, and no fire. Of these, the best was InceptionV3, that could demonstrate a validation accuracy of 94% and a test accuracy of 93%. These results demonstrate efficiency with early fire and smoke incidents. The main reason for the excellent performance of InceptionV3 is said to be the sophisticated architecture that it uses for dealing with complex visual patterns in classification.VGG16 could achieve only passable results, with only an accuracy marginally below that of InceptionV3. Xception, although showing a bit better result than VGG16, could not go beyond InceptionV3. Figure 7 captures the comparison of validation and test accuracies, where it is evident that InceptionV3 has outperformed the rest in both accuracy and overall robustness. The feature of InceptionV3 comes across as highly appealing for the domain since the inception modules allow multi-scale feature extraction, which makes it suitable for the application, which includes complex visual patterns with varying kinds of smoke and fire.These results have proved the relevance of the use of complex architectures such as InceptionV3 to early fire-detection applications and thus hold enough promise for improvement in firefighting systems.

Model	Accuracy (Validation, Feature Extraction)	Accuracy (Testing, Feature Extraction)	Accuracy (Validation, Fine-tuning)	Accuracy (Testing, Fine-tuning)
VGG 16	89%	93%	89%	87%
InceptionV3	**94%**	**93%**	85%	89%
Xception	91%	93%	71%	59%

Fig. 9. Model Accuracy Comparisons for Feature Extraction and Fine-tuning

VII. Conclusion

The InceptionV3 model has been shown to be highly effective when it comes to detecting forest fires and smoke; hence, this explains why it acquired the highest accuracy when compared to the other tested models. Its architectural improvements are built for the effective exploitation of complicated visual patterns that would be needed in any task involving these image classification tasks whose level of precision is related.Through transfer learning, it was possible to significantly reduce the computational load which ended up reducing the processing time but with a high performance level. The design will make the model well suitable for real-time applications requiring environmental monitoring where fast and accurate detection is a critical necessity.It would also evolve toward improving its adaptability toward the broader range of conditions, optimizing it to run on low-power devices, and to integrate into remote monitoring systems. This will add further range to its application and effectiveness in the real world.

Dataset Used

- Fire and Smoke Dataset on Kaggle

References

[1] Manoj, S., Valliyammai, C., "Drone network for early warning of forest fire and dynamic fire quenching plan generation," *J Wireless Com Network*, 2023, 112 (2023). https://doi.org/10.1186/s13638-023-02320w.

[2] Liu, J., Wang, Y., Guo, H., et al., "Spatial and temporal patterns and driving factors of forest fires based on an optimal parameter-based geographic detector in the Panxi region, Southwest China," *Fire Ecology* 20, 27 (2024). https://doi.org/10.1186/s42408-024-00257-z.

[3] Mishra, B., Panthi, S., Poudel, S., et al., "Forest fire pattern and vulnerability mapping using deep learning in Nepal," *Fire Ecology* 19, 3 (2023). https://doi.org/10.1186/s42408-022-00162-3.

[4] San Mart´ın, R., Ottle´, C., So¨rensson, A., "Fires in the South American Chaco, from dry forests to wetlands: response to climate depends on land cover," *Fire Ecology* 19, 57 (2023). https://doi.org/10.1186/s42408-023-00212-4.

[5] Tydersen, J. M., Collins, B. M., Coppoletta, M., et al., "Fuel dynamics and reburn severity following high-severity fire in a Sierra Nevada, USA, mixed-conifer forest," *Fire Ecology* 15, 43 (2019). https://doi.org/10.1186/s42408-019-0060-x.

[6] Chen, X., et al., "Wildland Fire Detection and Monitoring Using a Drone Collected RGB/IR Image Dataset," *IEEE Access*, vol. 10, pp. 121301-121317, 2022, doi: 10.1109/ACCESS.2022.3222805.

[7] Tran, D. Q., Park, M., Jeon, Y., Bak, J., and Park, S., "Forest-Fire Response System Using Deep-Learning-Based Approaches With CCTV Images and Weather Data," *IEEE Access*, vol. 10, pp. 66061-66071, 2022, doi: 10.1109/ACCESS.2022.3184707.

[8] Gonc¸alves, M., Branda˜o, T., and Ferreira, J. C., "Wildfire Detection With Deep Learning—A Case Study for the CICLOPE Project," *IEEE Access*, vol. 12, pp. 82095-82110, 2024, doi: 10.1109/ACCESS.2024.3406215.

[9] Rashkovetsky, D., Mauracher, F., Langer, M., and Schmitt, M., "Wildfire Detection From Multisensor Satellite Imagery Using Deep Semantic Segmentation," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 7001-7016, 2021, doi: 10.1109/JSTARS.2021.3093625.

[10] Wang, L., Zhang, H., Zhang, Y., Hu, K., and An, K., "A Deep Learning-Based Experiment on Forest Wildfire Detection in Machine Vision Course," *IEEE Access*, vol. 11, pp. 32671-32681, 2023, doi: 10.1109/ACCESS.2023.3262701.

[11] Chaoxia, C., Shang, W., and Zhang, F., "Information-Guided Flame Detection Based on Faster R-CNN," *IEEE Access*, vol. 8, pp. 58923-58932, 2020, doi: 10.1109/ACCESS.2020.2982994.

[12] Wang, G., Li, J., Zheng, Y., Long, Q., and Gu, W., "Forest smoke detection based on deep learning and background modeling," 2020 *IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*, Shenyang, China, 2020, pp. 112-116, doi: 10.1109/ICPICS50287.2020.9202287.

[13] M.A. Duhayyim et al., "Fusion-Based Deep Learning Model for Auto mated Forest Fire Detection," Comput. Mater. Contin., vol. 77, no. 1,pp. 1355-1371. 2023. doi:10.32604/cmc.2023.024198.

[14] Wu, X., Lu, X., and Leung, H., "An adaptive threshold deep learning method for fire and smoke detection," 2017 *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Banff, AB, Canada, 2017, pp. 1954-1959, doi: 10.1109/SMC.2017.8122904.

ramya_scopes

ORIGINALITY REPORT

7 4 %	%	5%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

1%

★Veerappampalayam Easwaramoorthy Sathishkumar, Jaehyuk Cho, Malliga Subramanian, Obuli Sai Naren. "Forest fire and smoke detection using deep learning-based learning without forgetting", Fire Ecology, 2023

Publication

Centurion University
Shaping Lives
Empowering Communities

SCOPES-2024

IEEE
KOLKATA SECTION
&
BHUBANESWAR SUBSECTION

19th - 21st December 2024

Certificate of Presentation

This is to certify that **Dr.Sireesha Moturi** from **Narasaraopeta Engineering College** has presented a paper titled **"Deep learning framework for early fire and smoke detection "** in the 2nd International Conference on Signal Processing, Communication, Power, and Embedded Systems (SCOPES), technically co-sponsored by the IEEE Kolkata Section and Bhubaneswar Sub-Section (IEEE-approved conference record number #64467), **organized by the Department of Electronics & Communication Engineering, School of Engineering & Technology, Centurion University of Technology and Management, Paralakhemundi, Odisha** during December 19-21, 2024.

Prof. Prafulla Kumar Panda	Prof. Ashok Misra	Prof. Debendra Kumar Sahoo	Prof. Anita Patra
Programme Chair	Convener	Organizing Chair	General Chair