Fire Detection under Suspicious Activity Recognition in Video Surveillance using Deep Learning Algorithm

Neha Gupta[1,2,a)] and Bharat Bhushan Agarwal[3,b)]

[1]*Computer Science and Engineering Department, IFTM University, Moradabad, India.*
[2]*Computer Science and Engineering Department, Moradabad Institute of Technology, Moradabad, India*

[3]*Computer Science and Engineering Department, School of Computer Science and Applications, IFTM University, Moradabad, India.*

Corresponding author: a) discoverneha@gmail.com
b) bharat_agarwal@iftmuniversity.ac.in

Abstract:

This study presents a deep learning model-based real-time fire detection system using YOLOv8 technology. The system uses a combination of deep learning, machine learning, and computer vision to detect fires in real-time video surveillance. The research focuses on the complex techniques used in deployment, validation, and data preparation. The system mimics the human fire detection process, handling video data generated by regular cameras. The results show the system is effective in spotting flames or fire, reducing false alarms and security threats. The system also improves assessments, reduces security threats, and continuously learns from previous incidents to adapt to new situations. This approach, which incorporates YOLOv8 which is a significant advancement in fire protection measures, improving public safety and safeguarding lives and property. When a fire is detected in the frame, an alarm is sent along with a confidence score and a time instance.

Keywords: Fire recognition,Fire object detection,YOLOv8, Roboflow, Ultralytics, Deep learning, cls loss, box loss, dfl loss, mAP

INTRODUCTION

Fire is an unusual occurrence that has the potential to seriously harm both people and property. This work describes the design and implementation of a YOLOv8 deep learning model-based real-time fire detection system. The study uses a mix of deep learning, machine learning, and computer vision approaches to accomplish reliable fire detection in real-time video surveillance [11]. It examines the complex techniques used at every turn, from deployment and validation to data preparation.Thus, in this research, we propose a video sequence-based deep learning-based fire detection approach that mimics the human fire detection process. This study presents a novel approach to handle video data generated by a regular camera watching a scene using the YOLOv8 model to identify fire and/or flames in real-time. The results of the experiments demonstrate how effective the suggested strategy is in spotting flames or fire. Furthermore, it significantly lowers the number of false alarms that are set off for common fire-coloured moving items. They improve their assessments, reduce security threats, and continuously absorb lessons from previous occurrences to adjust to novel situations by carefully pre- and post-processing. This strategy, which incorporates YOLOv8 and other state-of-the-art algorithms, is a major advancement in fire protection measures that will improve public safety and safeguard lives and property.

Unusual events like fire can swiftly result in serious injuries and property destruction.[1] The United States fire department reacted to an estimated 1,319,500 fires in 2017,[2] according to the National Fire Protection Association (NAPA). These fires caused an estimated $23 billion in direct property destruction, 3,400 civilian fire fatalities, and 14,670 civilian fire injuries. Early fire detection without a false alarm is essential to preventing such tragedies. As a result, numerous automatic fire detection technologies are under development and have broad practical use.

There are two main types of technologies: conventional fire alarms and computer vision-based fire detection. Smoke or heat sensors are the foundation of conventional fire alarm technology, and they need to be in close proximity to activate. In the event of an alarm from these sensors, human confirmation of a fire is required. Additionally, in order to provide information on the size, position, and degree of burning of the fire, these systems need a variety of equipment. Researchers have been looking into computer vision-based techniques in conjunction with different kinds of auxiliary sensors to get over these restrictions [3,4,].

Larger surveillance coverage and faster response times are provided by this category of technology, which also has the benefit of requiring less human intervention because a fire may be verified without a site visit and provides comprehensive fire information including location, size, and degree. Notwithstanding these benefits, there are still certain problems with system complexity and erroneous detection for various causes. Therefore, a lot of work has gone into developing computer vision technologies to overcome these problems.

Exploring the static and dynamic properties of flame to be used in a vision system is often a difficult undertaking since it calls for a lot of domain knowledge. However, in the deep learning technique, these processes of exploration and exploitation can be substituted by training a suitable neural network with enough data to prevent overfitting. Therefore, once a dataset including numerous photos or video clips of flames has been constructed, this strategy becomes practical.

Furthermore, numerous public spaces currently have closed-circuit television (CCTV) surveillance systems installed, which monitor both indoor and outdoor areas. By using fire detection software to process CCTV camera outputs in real time, such systems may be able to identify fires earlier.

International fire events are becoming more severe as a result of changing global climate conditions and increased human activity. These occurrences have had a significant negative impact on human communities, economy, and ecosystems. They have also resulted in significant property damage, biodiversity loss, and fatalities. For this reason, it is crucial to conduct research and develop new fire detection systems in order to identify fires early and put protective measures in place before they cause too much damage. Emergency situations such as fires necessitate quick thinking and efficient reaction plans. Effective emergency management and response plans can help lessen a fire's long-term effects in addition to its immediate damages.

Deep learning has proven to be a very effective and versatile machine learning framework for video fire detection. With an emphasis on current developments in deep learning techniques and widely used datasets for fire recognition, fire object detection, and fire segmentation, this paper provides an overview of deep learning-based video fire detection methods.Thus, making a significant investment in strong flame detection systems is crucial for securing industrial sites, defending the environment, and saving lives.

LITERATURE SURVEY FOR FLAME DETECTION TECHNOLOGIES

Conventional fire detection approaches mainly include methods based on visual image processing, temperature, flame, and smoke detection [5] .The first three of these, which rely on traditional fire alarms, have been applied extensively to stop large losses. According to Kanwal, Liaquat, Mughal, Abbasi, and Aamir (2017),[6] these solutions usually entail sensor-based temperature sampling, smoke analysis, and particulate sampling. However, the effectiveness of these detection techniques depends on the performance of the sensors, which must frequently be positioned close to the fire source for the best detection, which can be difficult in open spaces. Furthermore, not all fires happen when people are around, which is why manual monitoring is advised in order to confirm fires and assess their severity. At the same time, the low efficiency of obtaining discriminative feature information makes manual feature extraction from large volumes of data impractical.

Deep learning techniques have shown great promise and broad applicability in a variety of industries. They can extract more intricate and non-linear features from data. Various industries such as finance [7] ,medical care [8], monitoring [12], data image information extraction and monitoring systems are some examples. Due to deep learning's increasing popularity, scientists are experimenting with how well it can recognize and comprehend complicated patterns in flames. For automatic feature extraction and subsequent fire detection, a number of deep learning models have been used to date, including YOLO series model [13] (Wu & Zhang, 2018), Convolutional Neural Networks (CNN) [14] (Muhammad et al., 2018), AutoEncoders (AE) (Kim, Kim, Lee, & Kim, 2021), and Faster R-CNN [15](Zhang, Lin, Zhang, Xu, & Wang, 2018). Based on these foundations, researchers have presented a number of improved approaches. For example, Xu, Lin, Lu, Cao, and Liu (2021) [16] proposed an integrated method that uses EfficientNet, Yolov5, and EfficientDet to adaptively extract and detect features related to forest fires in various scenarios. To ensure the efficiency and reliability of EfficientNetB0 in fire detection by proposing a bespoke framework that uses attention processes and transfer learning [17]. In order to increase the accuracy of early smoke identification in forest fires, Hu et al. (2022) suggested a multidirectional smoke detection approach based on the value-switching attention mechanism and hybrid NMS [18]. This system combines weighting algorithms, soft-pool-space pyramid pooling, and multidirectional detection.

In order to efficiently identify and locate smoke and fire, Frizzi, Bouchouicha, Ginoux, Moreau, and Sayadi (2021) presented a novel approach based on convolutional neural networks for segmenting RGB images and creating fire and smoke masks [19]. These techniques have had a great deal of success in lowering the percentage of false positives and false negatives while also greatly enhancing the accuracy and resilience of fire detection. They are nonetheless constrained, nevertheless, by intricate environmental circumstances. For example, weather like rain and snow, situations with a lot of smoke, or bright lighting can all have an impact on the quality of photos or movies.

Furthermore, the size of the flame in the video or image to be detected may be quite small due to a variety of factors, including observing distance and angle. In these conditions, deep learning models would have trouble identifying these tiny flames, and the background features might mask their features. Additionally, certain natural things might resemble flames, and environmental variations such the dramatic variation in flame appearance and the presence of clouds, sunlight, and light reflection could make it more difficult to spot fires. As a result, these elements make fire detection extremely difficult.

INTEGRATION OF YOLOv8 in FIRE DETECTION SYSTEM

Yolov8s, a state-of-the-art object detection algorithm, offers a viable real-time fire detection solution.The Increasing Use of Deep Learning in Fire Safety has become a potent instrument for fire detection in recent years. Convolutional Neural Networks (CNNs) are capable of efficiently identifying fire patterns because they can learn intricate visual elements from large datasets. A novel object detection method called You Only Look Once version 8s (YOLOv8s) offers a potential real-time fire detection solution.The timeline of YOLO model from v1 to v8 is shown in fig.1.

Figure 1: Timeline of YOLO Model

Since its launch in 2015, the YOLO (You Only Look Once) algorithm has experienced a notable metamorphosis. YOLO was once praised for its lightning-fast real-time processing, but it has since improved to overcome accuracy issues and provide more versatility for a range of object recognition applications.

The most recent version of the YOLO algorithm, YOLOv8, represents a major breakthrough in real-time object detection. It adds new features to solve historical issues while enhancing the qualities of its predecessors. The major improvements that set YOLOv8 apart from earlier iterations are examined in this study. Flexible Backbone for Customized Object Recognition YOLOv8 carries over the modular design concept from YOLOv4, offering customers an assortment of backbones to meet their own requirements:
* CSPDarknet53 (default): This balanced backbone is appropriate for a variety of object detection jobs because it strikes a decent balance between speed and accuracy. * EfficientDet (optional): This lightweight backbone, which prioritizes speed, is perfect for contexts with limited resources and computing power.
* Custom Backbones: ResNet and EfficientNet are two examples of pre-trained backbones that may be experimented with using YOLOv8. By utilizing transfer learning approaches, these backbones—which were initially optimized for picture classification—can be further tailored for certain tasks, such as weapon detection, which could result in enhanced weapon-specific feature extraction.
*Enhanced Small Object Detection with Focus Scaling Module (FSM).

Ultralytics and Roboflow:

RoboFlow is an open-source computer vision framework that uses machine learning methods to fuel its continuous learning and adaption capabilities, along with Ultralytics as shown in fig.2.

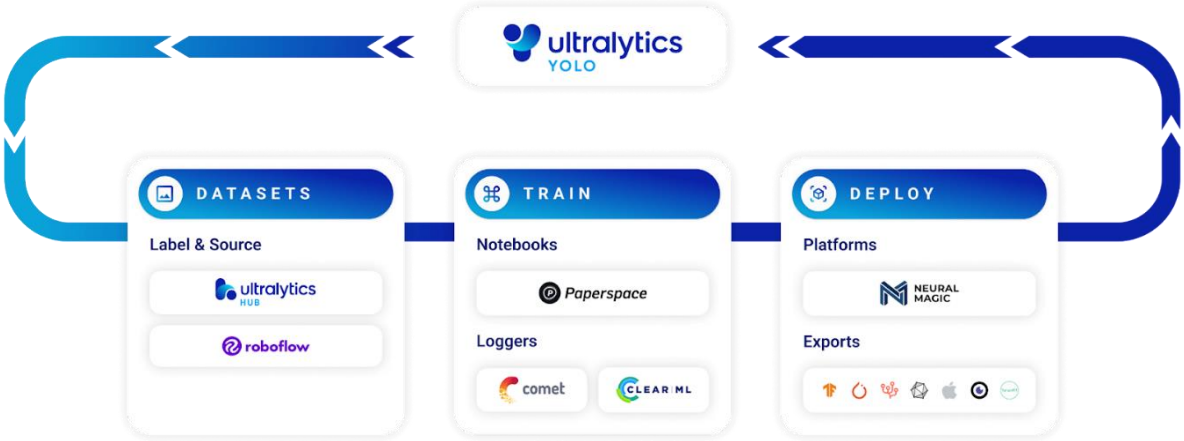

Figure 2:Ultralytics Interface

A powerful open-source library called Ultralytics focuses on computer vision tasks like segmentation, classification, and object identification. It provides an extensive range of features, such as:

Object detection: Identifying things in photos by utilizing cutting-edge deep learning models that have been trained on massive datasets.

Segmentation: Enables accurate object boundary delineation by supporting both semantic and instance segmentation.

Classification: Accurate picture classification into predetermined categories using strong neural networks.

With its modular architecture, Ultralytics offers pre-trained models that range in size from smaller, accuracy-optimized models to bigger, flexible models for edge devices.

A system called RoboFlow uses machine learning techniques to enable ongoing learning and adaptation. Large volumes of data can be analyzed by systems to gain insights that help them perform better, foresee problems, and make wise decisions quickly. With the help of this adaptive intelligence, firms can remain competitive in a market that is changing quickly by fostering innovation, fostering agility, and improving operational efficiency.

RoboFlow stresses the transformative influence of continuous learning and adaptation using machine learning

algorithms, boosting operational efficiency and stimulating creativity in multiple sectors, while Ultralytics focuses on computer vision tasks and offers a flexible toolbox for developers.

Using cutting-edge deep literacy models, the Ultralytics Python module provides an open-source toolset for posture estimation, object detection, and case segmentation. It uses the PyTorch framework as a foundation and provides a stoner-friendly API for planting and training these models. Important characteristics of the Python module Ultralytics include Model Assistance Pre-trained performances of well-known object detection models, such as YOLOv5, YOLOv6, YOLOv7, and YOLOv8, are included in the toolbox. These models exhibit great delicacy across colorful challenges, thanks to their extensive dataset training. API for stoners The API is designed to be user-friendly, even for those without strong background knowledge in deep literacy. It offers a simple-to-use interface for pre-processing images, importing models, and making predictions. Personalization Ability Drug users are able to modify the models to fit their own circumstances.

The toolkit provides customization options for both developing new models and fine-tuning pre-trained models on a bespoke dataset. GPU Intensification GPU acceleration support is built into the software, which greatly accelerates the training and conclusion operations. Development in Action The Ultralytics Python module is constantly being developed and preserved by a committed group of masterminds and experimenters. This guarantees frequent upgrades, bringing forth fresh functions and improvements.

DATASET PREPARATION

Roboflow is a platform designed to streamline the creation, management, and enhancement of datasets, particularly for computer vision tasks. It offers tools for uploading, annotating, preprocessing, and exporting image datasets, making it easier for developers and researchers to build and train computer vision models. Roboflow simplifies the process of collecting and annotating large volumes of data, enabling users to label objects within images with bounding boxes as shown in fig.3, polygons, or keypoints. Annotated datasets are crucial for supervised learning tasks in computer vision, and Roboflow provides preprocessing capabilities to increase variability and improve model generalization. Once annotated and preprocessed, datasets need to be exported in a format suitable for training machine learning models, such as TensorFlow or PyTorch. Roboflow is a tool that allows users to efficiently annotate objects within images. It involves uploading images, selecting an annotation tool, annotating objects, assigning labels, reviewing and editing annotations, saving annotations, and exporting the annotated dataset. Users can upload images individually or in bulk, depending on their dataset size and organization. Roboflow offers various annotation tools, such as bounding box, polygon, and keypoint annotation, to suit different annotation requirements. Once annotated, users can assign labels to each object, providing information about their category or class. Roboflow also allows users to review and edit annotations, adjust their position, size, and shape, save annotations, and export the annotated dataset in a format compatible with their machine learning framework or library. This user-friendly interface and powerful annotation tools make Roboflow an efficient tool for creating high-quality annotated datasets for computer vision model training.

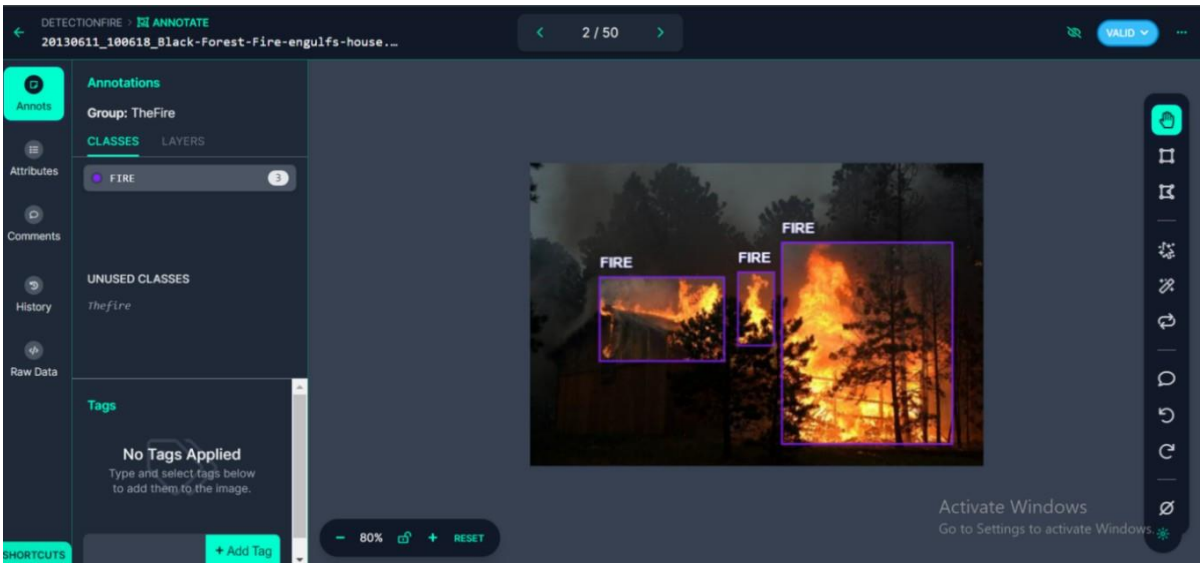

Figure 3: Annotating Input Image from Dataset

Train:

Roboflow is a machine learning tool that allows users to create custom models by uploading and annotating an annotated image dataset. Users can choose from a range of pre-trained model architectures for various computer vision tasks, such as object detection, image classification, and semantic segmentation. Training settings can be configured based on the dataset's size, complexity, and computational resources. The dataset can be split into training, validation, and test sets for accurate evaluation. Roboflow handles the training infrastructure, including provisioning GPU resources and managing training jobs. During training, users can monitor progress, track metrics, and visualize performance trends. After training, they can evaluate the model's performance using the validation or test set. Roboflow offers tools for visualizing predictions, calculating metrics, and analyzing model errors. Once satisfied with the model's performance, the model can be exported in formats like TensorFlow SavedModel, PyTorch TorchScript, etc making it easy to integrate into applications.

YOLOv8 MODELS and its METRICS

The incoming image is first preprocessed by the YOLOv8 architecture as shown in Fig.4, which involves resizing and priming it for feature extraction. Then, a feature extraction backbone is utilized, which may use an advanced convolutional neural network (CNN) architecture like EfficientNet or a modified ResNet. The detection process of YOLOv8 would be further refined, maybe using multi-scale feature fusion approaches such as Feature Pyramid Networks (FPN) or attention mechanisms. The goal of this improvement is to improve the model's capacity to identify items in the image that differ in size and complexity.

YOLOv8's prediction method is still based on dividing the input image into a grid of cells and assigning class probabilities, bounding boxes, and objectness scores to each cell. To achieve better accuracy and resilience, the model might use more complex techniques for bounding box prediction and categorization.YOLOv8 would then use post-processing techniques like non-maximum suppression (NMS) to weed out duplicate detections and improve the final collection of object detections, as predicted. Furthermore, the integration of optimization techniques such as quantization and pruning might improve the model's performance for real-world applications, especially on devices with limited resources.

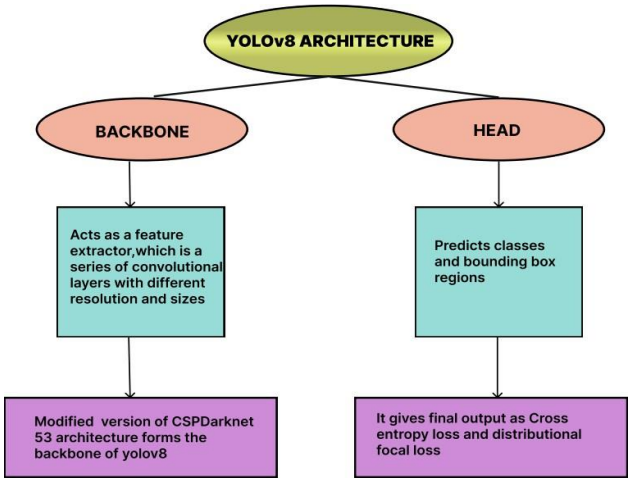

Figure 4: YOLOv8 Architecture

This is a summary of YOLOv8's operation, along with a flowchart as shown in Fig.5 and the factors that contribute to its widespread use in object detection:

1. Input Image: Any size of image can be used as an input for YOLOv8.
2. Preprocessing: In order to get the input image ready for the neural network, it is preprocessed. Resizing, normalization, and other preprocessing procedures could be required for this.
3. Neural Network Architecture: A deep convolutional neural network (CNN) architecture, comprising multiple convolutional layers and then fully linked layers, is commonly used by YOLOv8. YOLOv8 extracts hierarchical

representations of the input image through feature extraction layers.
4. Predictions: For each object in the input image, the network forecasts bounding boxes and class probabilities. The input image is divided into a grid of cells using YOLOv8, which then forecasts bounding boxes and class probabilities for each cell.
5. Anchor Boxes: To forecast bounding boxes of various sizes and forms, YOLOv8 uses anchor boxes. Anchor boxes are pre-established bounding boxes that are utilized in training to increase the precision of detection.
6. Non-Maximum Suppression (NMS): YOLOv8 uses non-maximum suppression (NMS) to eliminate redundant bounding box predictions once predictions are generated. NMS assists in removing overlapping boxes so that only the most certain predictions are kept.
7. Output: YOLOv8's end product is a collection of bounding boxes that show the presence of objects in the input image together with the matching class probabilities.

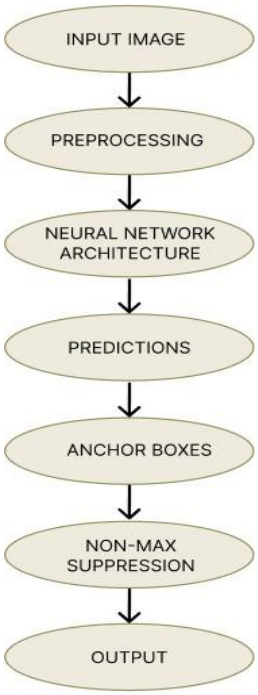

Figure 5: Flow Diagram

To sum up, labels are essential to YOLO object detection since they offer semantic information about items that are recognized, make model training and assessment easier, and allow for a deeper comprehension and interpretation of visual scenes.

● **Loss Function**: Minimizing the loss function, which is comprised of localization loss, confidence loss, and classification loss, is the aim of YOLOv8 training. The loss function can be expressed mathematically as shown in Eq.1.

$$\text{Loss} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 + \lambda_{obj} \sum_{i=0}^{S^2} 1_{ij}^{obj} (C_i - \hat{C}_i)^2$$

Equation-1

Where,

- N represents the quantity of grid cells in the picture
- S denotes the grid size
- B is the quantity of bounding boxes that each grid cell predicts.
- C is the quantity of classes.
- The localization loss coefficient is represented by λcoord.
- The coefficient for the confidence loss of no-objectness predictions is $\lambda noobj$
- The coefficient for the confidence loss of objectness predictions is λobj
- $(x_i, y_{i)}$ and $(x\%_i, y\%_{i)}$ are the bounding box center's ground truth and prediction coordinates for the i-th grid cell.
- $(w_i, h_{i)}$ and $(\hat{w}_i, \hat{h}_i)$ are the bounding box's anticipated and actual widths and heights for the i-th grid cell.
- $(C_i, C+_i)$ represent the objectness (predicted and ground truth) confidence scores for the i-th grid cell.
- 1_{ij}^{obj} is an indicator function that, if an object is present in the j-th bounding box in the i-th grid cell, equals 1; otherwise, it equals 0.
- 1_{ij}^{noobj} is an indicator function that, in the case that there is no object in the i-th grid cell's j-th bounding box, equals 1; otherwise, it equals 0.

The terms that penalize errors in localization (bounding box position and size), confidence predictions (based on whether an object is inside the bounding box), and classification predictions (based on object classification) make up the loss function. The coefficients "λcoord", "λnoobj", and "λobj" regulate the relative importance of each term in the overall loss.

● **Weight Update Equation**:
The following represents the weight update equation in Eq.2 for the Stochastic Gradient Descent (SGD) employed in YOLOv8:

$$W_{t+1} = W_t - \alpha \cdot \nabla L(W_t)$$ …….Equation-2

Where,

- At iteration t, W_t represents the current set of weights.
- The learning rate, represented by α, governs the number of steps that are executed during optimization.
- The loss function evaluated at the current set of weights W_t is denoted as $L(W_t)$
- The gradient of the loss function with respect to the weights W_t is represented by the symbol $\nabla L(W_t)$.
- The updated set of weights, W_{t+1} is calculated by deducting the scaled gradient from the existing weights.

- **Confidence**

In object detection models such as YOLOv8, confidence scores are important. They show how certain the model is that an object has been spotted in a picture or video frame. It helps in:
Certainty Assessment: The model's degree of confidence that a specific object is present in the picture or video frame is indicated by the confidence score. Greater certainty is indicated by a higher confidence level, whereas uncertainty is shown by a lower score.
Thresholding: Low-confidence detections are usually filtered away using confidence ratings. A threshold number, such as 0.5, can be set to help eliminate false positives and increase the detection system's precision. Detections with confidence ratings below this threshold are disregarded.
Decision Making: The system's decision-making procedures are influenced by confidence scores. High confidence detections, for instance, may initiate specific actions or alarms in applications like autonomous cars or surveillance systems, but low confidence detections can be disregarded or call for additional testing. **System Sensitivity:** The sensitivity of the detection system can be controlled by system developers by modifying the threshold for confidence scores. While a lower threshold yields more detections with possibly lower certainty, a higher threshold produces fewer but more confident detections.
Performance Evaluation: The effectiveness of object detection models is also assessed using confidence scores. In order to evaluate the total efficacy of the model, metrics like accuracy, recall, and F1 score frequently consider both the existence of correct detections and the confidence scores that go along with them.

To summarize, confidence scores play a crucial role in object detection models that evaluate performance, control

system sensitivity, influence decision-making processes, filter out low-confidence detections, and assess certainty. They support the detecting system's general resilience and dependability.

• LABELS:

The predefined categories or classes that the model has been taught to identify are referred to as "labels" in the context of YOLO object identification. Every object that YOLO detects has a label that indicates the class or category that it belongs to. Each object that is discovered during this labeling process is given a unique tag that indicates its classification in the dataset that was used to train the model.Its significance includes:

Categorization Identification: The labels denote the pre-established groups or classes that the YOLO model is trained to identify. Depending on the application domain and training dataset, these categories may contain items like automobiles, bicycles, pedestrians, traffic signs, and more.

Object Classification: A label is assigned to each detected object in an image or video frame based on its category. Higher-level comprehension of the picture is made possible by the classification process, which gives the model access to both the semantic meaning and the position of objects.

Training Data Annotation: Bounding boxes and labels are placed around objects of interest in pictures or video frames during the training phase. The YOLO model is trained using this annotated data as the ground truth in order to precisely forecast the bounding box coordinates and the labels that correspond with them.

Assessment of the Model: Labels are essential for assessing how well the YOLO model performs. Metrics like recall, accuracy, precision, and mean average precision, or mAP (mean average precision), are calculated depending on how well the model predicts the bounding boxes and labels of objects in test data.

Semantic comprehension: YOLO helps downstream applications make defensible decisions based on the categories of identified objects by providing labels to detected objects. This process promotes semantic comprehension of the scene.

User Interpretation: Labels give information about the things the model recognized that can be understood by humans. By looking at the labels attached to each object that the YOLO system detects, users may quickly comprehend the output produced by the system.

• PRECISION:

Precision is a key performance indicator for object detection models such as YOLOv8. Out of all the positive predictions the model makes, precision is the percentage of true positives (things that are accurately identified). The ratio of actual positives to all positive predictions is used to compute it. i.e.

$$\text{Precision} = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Interpretation: A greater precision number denotes a more accurate model that avoids false positives, i.e., a more exact object detection. On the other hand, a lower precision number indicates a larger rate of false positives, which may require additional development.

Metrics: YOLOv8 has metrics that summarize precision across various confidence levels and object types, such as Average Precision (AP) and Mean Average Precision (mAP). These metrics provide a thorough assessment of the model's effectiveness in tasks involving object detection.

Visualization: YOLOv8-generated bounding boxes and confidence scores can be seen using programs like TensorBoard. Analyzing false detections and improving the model's precision are made easier with the help of this display.

Hyperparameter fine-tuning: For particular dataset, fine-tuning training parameters like learning rate, epochs, and anchor boxes can assist increase accuracy. Furthermore, adding methods to the training data, such as scaling, flipping, or color jittering, diversifies the dataset and improves the generalization capacity of the model.

In conclusion, precision is an essential criterion for evaluating YOLOv8 and other object detection models' accuracy. Through comprehension of its calculation, analysis, measurements, tools for visualization, methods for comparison, and optimization strategies, programmers can assess and improve the accuracy of their models for a range of object recognition applications.

● BEST.PT:

In deep learning processes, "best.pt" is a standard procedure that allows training, performance tracking, and model deployment for inference activities to resume. These saved models offer flexibility and efficiency in the building and deployment of models, making them invaluable assets in machine learning pipelines.The saved model with the name "best.pt" usually has parameters linked to the best-performing version of the model, which is usually identified by validation metrics.

PyTorch and other deep learning frameworks offer the ability to load stored models from files called "best.pt." These features let you retrieve the model's parameters and apply them to different scenarios, including picking up where you left off with training or doing inference operations on fresh data. You can deploy the model for inference tasks in production environments or smoothly continue training from the best-performing checkpoint by loading "best.pt" utilizing PyTorch's model loading features.

● LAST.PT:

For saved model files, "last.pt" is another popular naming scheme, especially when deep learning and neural networks are involved.

Like "best.pt," "last.pt" is a stored model file that holds the weights or parameters of a model that has been trained. Usually, the most recent or last checkpoint saved throughout the training process is represented by the "last.pt" file. It accomplishes several goals, such as continuing training where it left off or applying the model to inference tasks on fresh data.

It is common practice when training deep learning models to store checkpoints on a regular basis. This allows you to monitor the model's progress and resume training if something goes wrong or further training is required. "last.pt" offers a practical means of storing the most recent checkpoint, making it simple for users to load the model's parameters and carry on with training where they left off.

It is typical practice for a machine learning model to periodically save its parameters, or weights, throughout training. These checkpoints are essential because they allow training to continue from locations or to deploy the model for inference tasks later on. If you come across a file called "last. pt," you can assume that it contains the model's most recent parameters. This file usually represents the model's state at the last checkpoint that was saved during training. This naming pattern ensures accessibility and user-friendliness for practitioners by helping to manage and organize the large number of checkpoints created during the training process.

Methodology Used:

The presented work uses an organized methodology that includes multiple crucial stages as shown in fig.6:

FIGURE 6: Flow of Stages as per methodology used

1. Gathering and preprocessing of data: First assemble a diversified dataset of fire images and videos that depicts a range of fire circumstances (e.g., different fire sizes, inside and outdoor flames). For efficient model training, make sure the dataset is of the highest caliber, thoroughly annotated, and contains well-labeled fire incidents. To improve the size of the dataset and the generalization of the model, preprocess the data by resizing, standardizing, and even supplementing the photos.
2. Training and Model Selection: use a pre-trained YOLOv8 model, such as "yolov8s-fire," which was trained on fire datasets. Proficiency in data preparation, annotation, and YOLOv8 training processes are necessary for this. The model picks up the ability to recognize objects and patterns of fire in the photos throughout training.
3. Model Integration for YOLOv8: To load the trained YOLOv8 model for real-time fire detection, use libraries such as Ultralytics. In order to facilitate frame-by-frame examination of the live video stream, integrate the model with the video processing pipeline.
4. Real-time detection of fire: For the same,Constantly record frames from the live video source (IP camera, webcam). Preprocess each gathered frame to ensure that it satisfies the input requirements of the model. Apply inference using the loaded YOLOv8 model to the preprocessed frame. Analyze the conclusions of the inference. For each identified fire incident, extract confidence scores and bounding boxes.Bounding boxes show the locations of fires, and confidence scores show how certain the model is that a fire has been detected. To lessen false Positives, you can optionally apply filtering based on confidence criteria.
5. Display and Results: By adding bounding boxes and confidence scores to the video frames, one can visually represent the identified fire zones. Create an output system by notifying recipients (by email) and documenting fire incidents for later examination
6. Implementation and Verification: Examine the system's functionality using live video broadcasts that feature fire incidents.To guarantee continued efficacy, keep an eye on system functioning and deal with any problems that crop up.

VALIDATION CHECKS

Deployment entails attaching the video stream device to the processing unit after the testing environment is set up and the model is verified. The report lists the essential deployment steps:

1. Putting Up the Ideal Weights (Best.Pt):

Make sure the model is utilizing the best training weights for the best fire detection before deploying the system. The model's optimal performance state is represented by the parameters that were learned during training and are contained in the "best.pt" file. Move this file to the processing unit housing the fire alarm system. By ensuring that the deployed model has the most recent and precise parameters, this phase ensures that the model can make accurate predictions in events that occur in real time.

2. Setting Up the Project Environment:

The project environment on the processing unit needs to be set up correctly for the fire detection system to be deployed effectively. Make sure that all required libraries and dependencies are installed and current, such as the YOLOv8 framework and related deep learning libraries. Furthermore, confirm that the processor unit satisfies the hardware specifications for effective inference, including enough RAM and, if relevant, GPU power. The foundation for the fire detection system's smooth deployment and dependable operation is a well-prepared surroundings.

3. Source of Video Stream:

If the video stream device is a webcam, IP camera, or some other type of source, include its IP address and any other pertinent information in the project code. Real-time fire detection is made possible by this arrangement, which gives the deployed model access to the live video feed. To prevent disruptions during operation, make sure the connection to the video stream device is steady and dependable. Precise setup of the video stream source guarantees prompt input for analysis to the model, improving the fire detection system's responsiveness.

4. Configuring a Fire Alert:

Configure the fire detection system to notify certain email addresses in the event of a fire. This arrangement facilitates rapid action and mitigation actions by enabling immediate notification to pertinent stakeholders. By enabling prompt reaction to possible fire occurrences, a strong alert system can improve the efficacy of the fire detection system. To allow the necessary action, configure the alarm system to deliver complete information, including the location and intensity of the detected fire.

5. Begin to Detect:

After everything is set up, start the fire detection procedure to examine the live feed and find any possible fire incidents. Activate the deployed model to begin processing the incoming video input, leveraging its

training parameters to identify fires in real time. Keep an eye on the system's performance to make sure it runs without hiccups or glitches. Assess the detection results on a regular basis to confirm the system's efficacy and resolve any problems that may occur while it's in use. The fire detection system becomes operational and prepared to efficiently protect against fire dangers upon detection.

- Image based detection: Create scripts to assess each image in the testing dataset for the accuracy of the model as shown in fig.7.

FIGURE 7: Image-based detection

- Video based detection: Develop features to test the model on video sequences as shown in fig.8. that have already been recorded and contain fire incidents. Examine the model's detection performance for various fire kinds in various video lengths as shown in fig.9.

FIGURE 8: Frames extracted from custom video for detection

FIGURE 9 : Fire detected with confidence while processing the video

- Detect live video streams: Configure the live video stream from the webcam or IP camera of your choice. Combine the model with the live feed to evaluate how well it performs in real life when it comes to real-time fire detection as shown in fig.10.

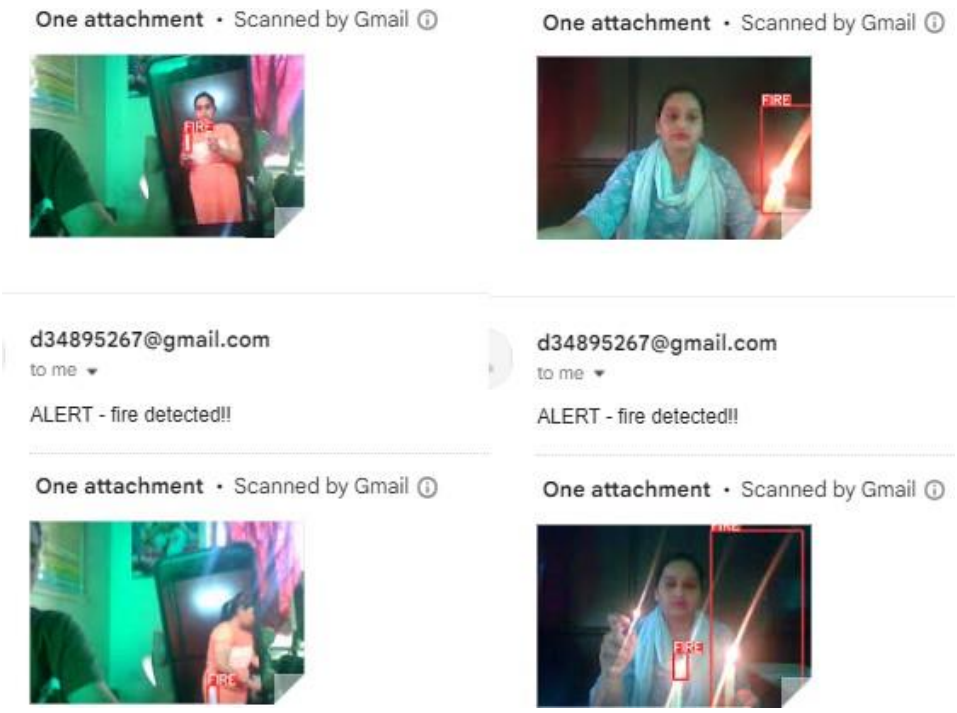

FIGURE 10: Fire detection on Live video stream and sending alert

RESULT

Using the custom image dataset of 2505 images, the YOLOv8 model was fine-tuned with changes made to the 200 number of epochs. A confusion matrix, box loss, cls loss, and dfl loss included in the evaluation metrics.
Accurately predicting bounding boxes is essential for object detection, the process of locating and classifying objects in images. In the regression job, the "box_loss" probably refers to the loss function that is used to forecast bounding box coordinates, which are usually expressed as (x, y, w, h). When calculating box loss, the model finds items in photos and attempts to predict the size and location of the bounding boxes that surround each object. The ground truth bounding box shown in the training data is then compared to the predicted bounding box. Greater precision in localizing objects within the image is shown by a lower box loss. It's crucial to remember that in the overall loss function of object detection models, box loss is frequently paired with additional components like objectness loss and classification loss. During training, minimizing this combined

loss improves the model's performance in tasks involving both object localization and classification. Classification loss is referred to as "cls loss" and is a part of the total loss function in the training of object detection models. Object detection entails classifying and using bounding boxes to localize items. The ability of the model to accurately predict the class label for each bounding box that has been discovered is measured by classification loss.
For class predictions, YOLOv8 usually uses the soft max activation function, which normalizes class scores into probabilities. YOLOv8's total loss function is made up of several parts, such as classification loss (cls loss), confidence loss (obj loss), and localization loss (coord loss). The model is trained to minimize this total loss by varying parameters, which enhances object localization and increases the accuracy of class prediction.
In object detection training, distributional focal loss, or dfl, attempts to alleviate class imbalance problems. The model redirects more attention to challenging, wrongly classified examples by adding a modifying component that downweights the loss associated with well-classified examples.
Model performance is often evaluated using additional assessment measures like recall, precision, and mean average precision, or mAP. Recall is the ratio of actual positive instances to all correctly predicted positive instances, whereas precision is the ratio of accurately predicted instances to all positively anticipated cases.
In order to assess model performance across multiple precision-recall trade-offs in object detection tasks, mAP takes into account precision and recall at varying confidence levels. A thorough evaluation of performance over a wide variety of IoU thresholds is provided by mAP 50-95, which sheds light on the model's efficacy at various bounding box overlap levels.

As we see the performance evaluation of a YOLOv8 fire detection system,we looked into how the model performed when the number of training epochs changed (15,30,60,120,200) as shown in table1. Analyzing the confusion matrix and performance graphs showed that when training epochs increased, there were noticeable gains as shown in fig. 10. These advancements showed up as: Increased frequencies of true positives More real fire incidents were accurately recognized by the model.
There was a discernible increase in confidence scores with longer training times as shown in fig. 11,12 and 13. The model generated fewer false alarms due to faulty fire detections. This shows how accurate the model is becoming and how confident it is in differentiating between things that are on fire and those that are not.

Table 1: Analyzing Loss by increasing number of epochs

No. of Epoch	**GPU_mem**	**box_loss**	**cls_loss**	**dfl_loss**
200	**4.29G**	**0.5329**	**0.3093**	**0.90**
120	**5.14G**	**0.8578**	**0.4909**	**0.9812**
60	**4.11G**	**0.762**	**0.4949**	**1.039**
30	**3.32G**	**0.635**	**0.4104**	**1.008**
15	**2.52G**	**1.629**	**2.324**	**1.924**

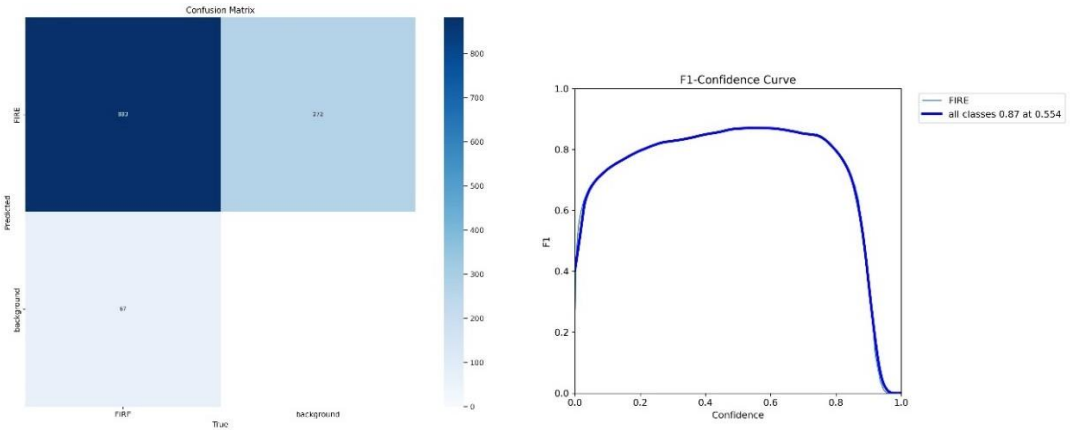

FIGURE 11: Confusion Matrix and F1-Confidence Curve

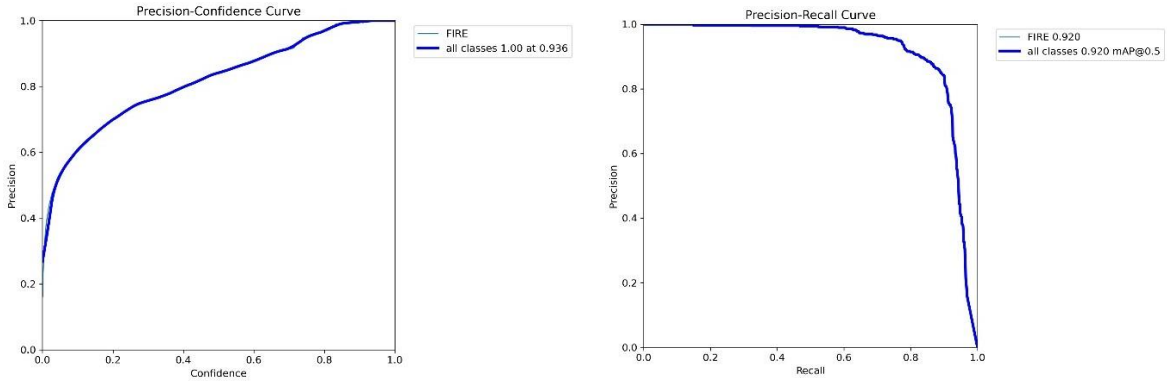

FIGURE 12:Precision-Confidence and Precision-Recall Curve

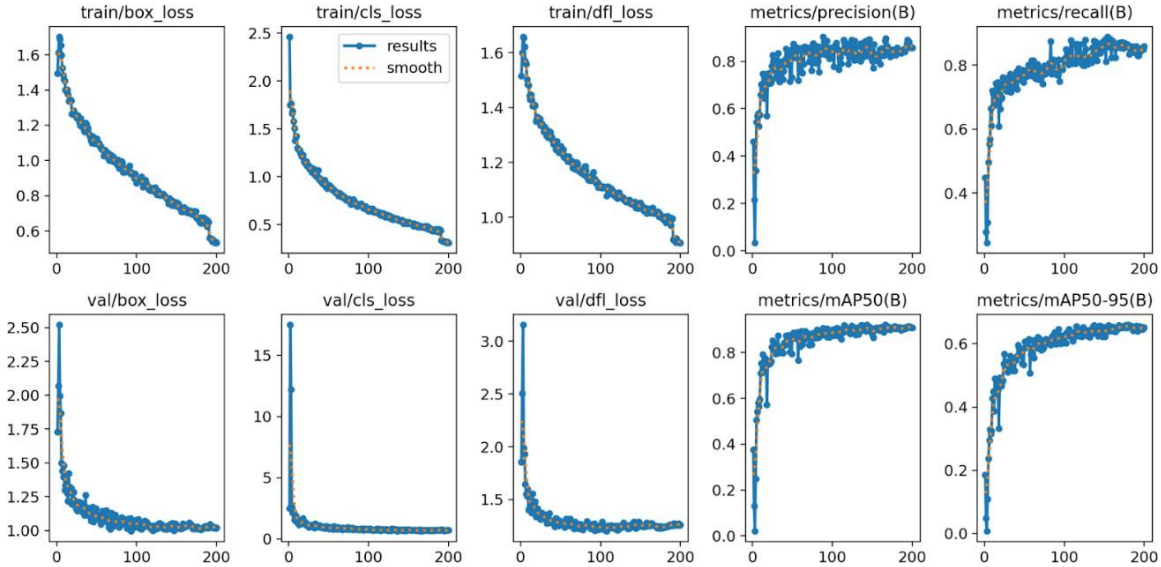

FIGURE 13: Results received at 200 Epochs

All things considered, the outcomes indicate how well our method works for real-time fire detection and how the model can adjust and get better over longer training periods.

CONCLUSION

The application of YOLOv8 for fire detection in video surveillance systems is a noteworthy development that contributes to the safety and security of a variety of settings. The potential for early fire event detection and mitigation has been considerably increased through the combination of video surveillance technologies with state-of-the-art deep learning algorithms like YOLOv8. This paper's research demonstrates how well YOLOv8 works to reliably identify smoke and flames in real-time video streams. YOLOv8 has demonstrated promising results in accurately and reliably identifying fire-related occurrences by utilizing its object detection and categorization skills. This makes it possible to respond and intervene in a timely manner to stop fire occurrences from getting worse and to lessen their effects on property and people. Researchers, developers, and practitioners in the field of fire safety and security can now easily apply YOLOv8 due to the utilization of pre-trained models and publicly available frameworks.

Future difficulties include resistance to climatic variability, detection performance enhancement, and integration with sophisticated fire suppression systems call for increased research and development. Furthermore, investigating multi-sensor fusion methods and adding contextual data from video streams may improve the performance of YOLOv8-based fire detection systems.

REFERENCES

1. Chi, R.; Lu, Z.M.; Ji, Q.G. Real-time multi-feature based fire flame detection in video. *IET Image Process.* **2016**, *11*, 31–37. [**Google Scholar**] [**CrossRef**]
2. Evarts, B. *Fire loss in the United States during 2017*; National Fire Protection Association, Fire Analysis and Research Division: Quincy, MA, USA, 2018. [**Google Scholar**]
3. Ko, B.C.; Ham, S.J.; Nam, J.Y. Modeling and formalization of fuzzy finite automata for detection of irregular fire flames. *IEEE Trans. Circuits Syst. Video Technol.* **2011**, *21*, 1903–1912. [**Google Scholar**] [**CrossRef**]
4. Qiu, T.; Yan, Y.; Lu, G. An autoadaptive edge-detection algorithm for flame and fire image processing. *IEEE Trans. Instrum. Meas.* **2012**, *61*, 1486–1493. [**Google Scholar**] [**CrossRef**]
5. T. -H. Chen, Y. -H. Yin, S. -F. Huang and Y. -T. Ye, "The smoke detection for early fire-alarming system base on video processing," *2006 International Conference on Intelligent Information Hiding and Multimedia*, Pasadena, CA, USA, 2006, pp. 427-430, doi: 10.1109/IIH-MSP.2006.265033.
6. Kehkashan Kanwal et al. Towards Development of a Low Cost Early Fire Detection System Using Wireless Sensor Network and Machine Vision, July 2017, Wireless Personal Communications , DOI: 10.1007/s11277-016-3904-6
7. M. N. Ashtiani and B. Raahemi, "Intelligent Fraud Detection in Financial Statements Using Machine Learning and Data Mining: A Systematic Literature Review," in *IEEE Access*, vol. 10, pp. 72504-72525, 2022, doi: 10.1109/ACCESS.2021.3096799.
8. Liu, S., Liu, D., Srivastava, G., Połap, D. and Woźniak, M., 2021. Overview and methods of correlation filter algorithms in object tracking. *Complex & Intelligent Systems*, *7*, pp.1895-1917.
9. universe.roboflow, Explore the Roboflow Universe, https://universe.roboflow.com/
10. github, ultralytics, https://github.com/ultralytics/ultralytics
11. Myagmar-Ochir, Y. and Kim, W., 2023. A survey of video surveillance systems in smart city. *Electronics*, *12*(17), p.3567.
12. Shaohua Wan, Songtao Ding, Chen Chen,,"Edge computing enabled video segmentation for real-time traffic monitoring in internet of vehicles",Pattern Recognition,Volume 121,2022,108146,ISSN 0031-3203,https://doi.org/10.1016/j.patcog.2021.108146.
13. S. Wu and L. Zhang, "Using Popular Object Detection Methods for Real Time Forest Fire Detection," *2018 11th International Symposium on Computational Intelligence and Design (ISCID)*, Hangzhou, China, 2018, pp. 280-284, doi: 10.1109/ISCID.2018.00070.
14. K. Muhammad, J. Ahmad, I. Mehmood, S. Rho and S. W. Baik, "Convolutional Neural Networks Based Fire Detection in Surveillance Videos," in *IEEE Access*, vol. 6, pp. 18174-18183, 2018, doi: 10.1109/ACCESS.2018.2812835.

15. Qi-xing Zhang, Gao-hua Lin, Yong-ming Zhang, Gao Xu, Jin-jun Wang,"Wildland Forest Fire Smoke Detection Based on Faster R-CNN using Synthetic Smoke Images",Procedia Engineering,Volume 211,2018,Pages 441-446,ISSN 1877-7058,https://doi.org/10.1016/j.proeng.2017.12.034.
16. Xu R, Lin H, Lu K, Cao L, Liu Y. A Forest Fire Detection System Based on Ensemble Learning. *Forests*. 2021; 12(2):217. https://doi.org/10.3390/f12020217
17. Hikmat Yar, Waseem Ullah, Zulfiqar Ahmad Khan, Sung Wook Baik,"An Effective Attention-based CNN Model for Fire Detection in Adverse Weather Conditions" ,ISPRS Journal of Photogrammetry and Remote Sensing,Volume 206,2023,Pages 335-346,ISSN 0924-2716,https://doi.org/10.1016/j.isprsjprs.2023.10.019.
18. Jialei Zhan, Yaowen Hu, Guoxiong Zhou, Yanfeng Wang, Weiwei Cai, Liujun Li,A high-precision forest fire smoke detection approach based on ARGNet,Computers and Electronics in Agriculture,Volume 196,2022,106874,ISSN 0168-1699,https://doi.org/10.1016/j.compag.2022.106874.
19. Frizzi, Bouchouicha, Ginoux, Moreau, and Sayadi," Convolutional neural network for smoke and fire semantic segmentation", https://doi.org/10.1049/ipr2.12046