The Smoke and Fire Alerting System

Group Members:

Shilpa Kuppili,

Sathwik Kuchana,

Sri Sai Lalitha Mallika Yeturi,

Priyadarshini Munigala

Yeshiva University, NYC, NY
Katz School of Science and Health

1. Abstract

This project pertains to tracking and detecting fires and smoke using YOLOv8 and a custom object detection model. The project uses a pre-trained YOLOv8 model to identify the presence of fire and smoke in each dataset and track it. Technically, the system employs a state-of-the-art object detection model, which meticulously checks the varied pictures to identify signs of smoke or fire. The YOLOv8 fire detection algorithm, named Fire-YOLOv8, marks a significant upgrade from YOLOv5, excelling in detecting small fire and smoke in various settings, including indoor, outdoor, and forest areas under different lighting conditions. It advances feature extraction, reducing model complexity and boosting performance. Fire-YOLOv8 stands out with a map of 93.2% and an F1 score of 91%, outperforming other detection models, especially in identifying tiny fire and smoke instances. It achieves real-time detection with an average speed of 0.1 seconds per image at 416x416 resolution, making it highly effective for rapid fire response. This model is also versatile, suitable for complex detection scenarios beyond fire, improving essential fire detection metrics significantly.

2. Introduction

Fire is necessary for human survival because it gives them energy. However, unplanned fire also contributes to a lot of accidents. The National Fire Protection Association (NFPA) reports that fire departments respond to more than 350,000 residential structure fires nationwide, with direct property damage estimated to be close to $7 billion. It's critical to identify fires quickly to put out such fires. Historically, sensors have been used to detect fires. However, installing sensors everywhere a fire could occur is costly because they take up space and require constant maintenance. A significant trend is the replacement of traditional fire detection methods with computer vision-based systems, thanks to the quick advancements in digital camera technology and video processing techniques. We propose a fire and smoke detection system based on a novel convolutional neural network (CNN) to address this need, using the YOLOv8 model. The proposed system combines deep learning models and modern computer vision.

3. Related work

Two new rules and two new detection methods using an intelligent combination of the rules are presented, and their performances are compared with their counterparts. The benchmark is performed on approximately two hundred million fire pixels and seven hundred million non-fire pixels extracted from five hundred wildland images under diverse imaging conditions. The fire pixels are categorized according to fire color and existence of smoke; meanwhile, non-fire pixels are categorized according to the average intensity of the corresponding image.

This characterization allows to analyze the performance of each rule by category. It is shown that the performances of the existing rules and methods from the literature are category dependent, and none of them can perform equally well on all categories. Meanwhile, a new proposed method based on machine learning techniques and using all the rules as features outperforms existing state-of-the-art techniques in the literature by performing almost equally well on different categories. Thus, this method, promises very interesting developments for the future of metro logic tools for fire detection in unstructured environments. Intelligent algorithm for smoke extraction in autonomous forest fire Detection, Ivan Grubesic, Darko Kolaric and Karolj Skala Forest fire, if not detected early enough, can cause great damage.

To reduce it, it is vital to detect fire as soon as possible and act upon it. In its early stage, forest fire manifests itself primarily as smoke, as flames are too little to be seen. Therefore, to ensure forest fire detection in its earliest stages, smoke detection is utilized. Autonomous forest fire detection based on smoke detection is currently one of the greatest challenges in image processing field. The main reason for it is that there are lots of smoke-resembling natural phenomena such as clouds, cloud shadows and dust. So, the essence of the problem lies in separating these phenomena from real smoke.

4. Methodology

YOLOv8 represents a significant leap forward in object detection technology, characterized by several key advancements that make it a powerful tool for various applications. Notably, it offers improved accuracy through the incorporation of new techniques and optimizations, setting it apart from its predecessors. This version is designed for speed, achieving faster inference times without compromising accuracy, which is crucial for real-time applications. One of the standout features of YOLOv8 is its flexibility across different backbones, such as EfficientNet, ResNet, and CSPDarknet, providing users with a range of options to tailor the model to their specific needs. Its adaptive training mechanism optimizes learning rates and balances loss functions more effectively during training, leading to enhanced model performance.

Moreover, YOLOv8 employs advanced data augmentation strategies like MixUp and CutMix, improving model robustness and generalization. The architecture of YOLOv8 is highly customizable, allowing for modifications to suit varied requirements, and it comes with pre-trained models to facilitate easy use and transfer learning.

Its compatibility with TensorFlow Lite and the TF Lite GPU delegate opens new possibilities for deployment on a wide range of platforms, ensuring optimal performance and efficiency. Despite the challenges associated with exporting models to TensorFlow Lite, the benefits in terms of speed, accuracy, and application flexibility make YOLOv8 a compelling choice for developers and researchers looking to push the boundaries of object detection. The YOLOv8 model is the latest iteration in the YOLO (You Only Look Once) series, renowned for its object detection capabilities. YOLOv8 builds upon the strengths of its predecessors, incorporating cutting-edge advancements to offer unprecedented accuracy, speed, and flexibility.

Input Image: The algorithm takes an input image and resizes it to a fixed size suitable for processing. This size is usually determined based on the network architecture used.

Grid Division: The image is divided into an S x S grid, where each grid cell is responsible for predicting objects present in that cell.

Bounding Box Prediction: Within each grid cell, YOLO predicts multiple bounding boxes. Each bounding box is defined by five attributes: (x, y, w, h, confidence). The (x, y) coordinates represent the center of the bounding box relative to the grid cell, while w and h represent the width and height of the box. The confidence score indicates how confident the algorithm is that the box contains an object.

Class Prediction: Along with the bounding box predictions, YOLO also predicts the probabilities of different classes for each box. The number of class probabilities depends on the dataset being used. These class probabilities represent the likelihood of each class being present in the bounding box.

Confidence Thresholding: YOLO applies a confidence threshold to filter out low-confidence predictions. Bounding boxes with confidence scores below the threshold are discarded as false positives.

Non-Maximum Suppression (NMS): To eliminate duplicate detections and improve accuracy, YOLO applies non- maximum suppression. NMS removes redundant bounding boxes that have significant overlap and keeps only the most confident one. The overlap threshold for suppression is typically determined by a predefined Intersection over Union (IoU) value.

Final Output: The output of the YOLO algorithm is a set of bounding boxes, each associated with a class label and confidence score. These bounding boxes represent the detected objects in the input image. The specific steps in the implementation procedure goes as follows. Downloading dataset from Roboflow and importing YOLOV8 from ultralytics library. Once the dataset is loaded train the model using pretrained model and save the weights file for testing and deployment.

4.1 Custom Object Detection Model:

It has been designed for object detection tasks, aiming to classify objects within an image and predict bounding boxes around them. It's a convolutional neural network (CNN) architecture, tailored for feature extraction and classification/regression tasks.

The model begins by defining several convolutional layers (`self. conv1`, `self.conv2`, `self.conv3`, `self.conv4`), which are responsible for learning various features within the input image. These convolutional layers have different depths, starting with 3 input channels (for RGB images) and progressively increasing the number of output channels, which effectively captures increasingly complex features.

After each convolutional layer, a ReLU activation function is applied to introduce non-linearity, followed by max- pooling (`self.pool`) to downsample the feature maps and reduce spatial dimensions while retaining important information.

Following the convolutional layers, there are fully connected layers (`self.fc1`, `self.fc2`, `self.fc3`) for classification and bounding box regression. The feature maps from the convolutional layers are flattened to be fed into these fully connected layers.

The first fully connected layer (`self.fc1`) acts as a feature aggregator, reducing the dimensionality of the feature maps to a fixed-size representation, enabling more efficient processing. The output of `self.fc1` is then passed through a ReLU activation function.

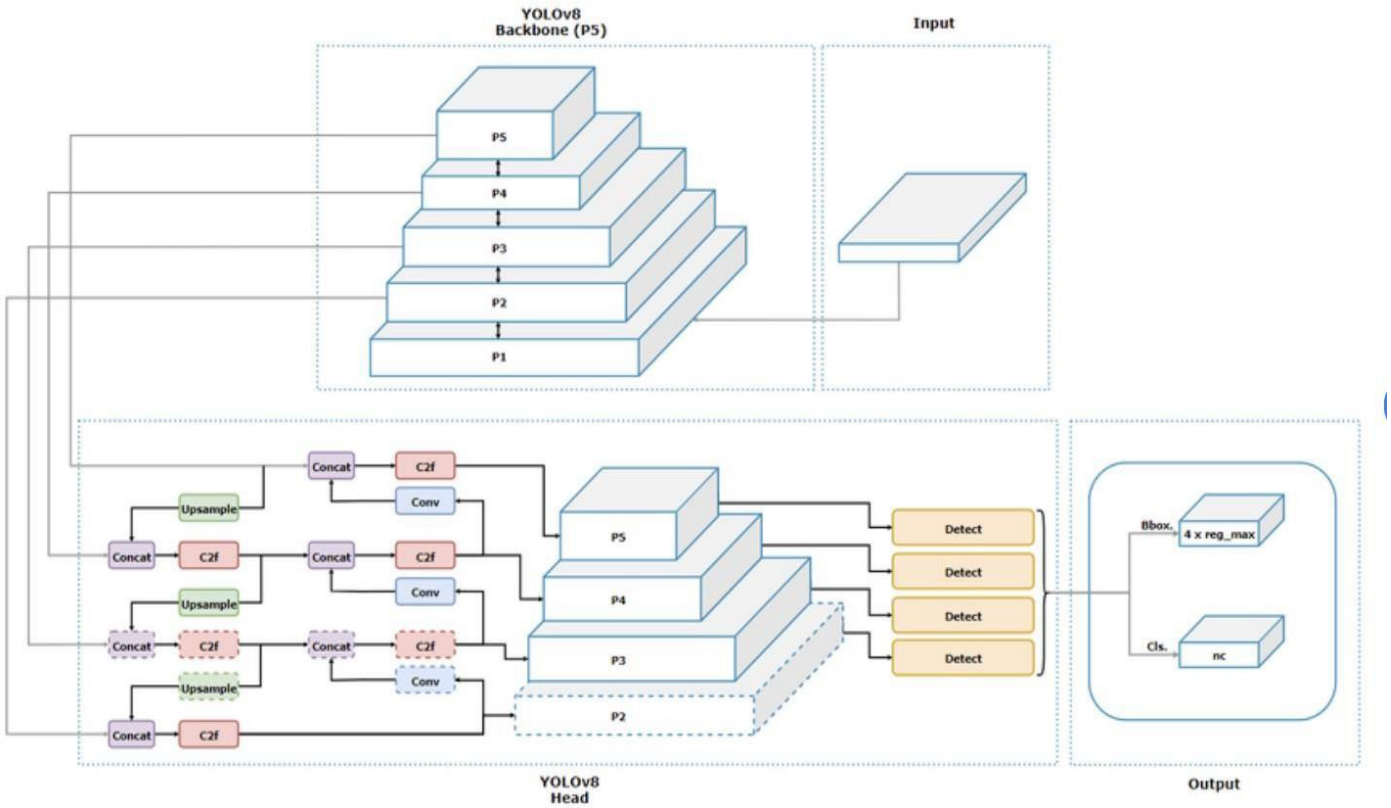
For classification, the output of the ReLU activation from the first fully connected layer is fed into another fully connected layer (`self.fc2`), which outputs probabilities for each class. This final classification layer `num_classes` neurons, where each neuron corresponds to a class, and a softmax activation is usually applied during inference to obtain class probabilities.

For bounding box regression, another fully connected layer (`self.fc3`) follows the ReLU activation from `self.fc1`. This layer outputs four values representing the coordinates of the bounding box (e.g., top-left corner coordinates, width, and height). These values are used to define the spatial extent of the detected objects.

In summary, the model follows a typical CNN architecture, starting with convolutional layers for feature extraction, followed by fully connected layers for classification and bounding box regression. It's a versatile architecture suitable for object detection tasks where both classification and localization (bounding box regression) are required.

4.2 Flowchart and Visual Representation

While a detailed visual representation, including flowcharts and architecture diagrams, would provide a clear understanding of YOLOv8's internal workings, imagine a network diagram that illustrates the flow from input image through convolutional layers, leading to feature extraction and object detection. The backbone processes the input, followed by the head that predicts object classes and bounding boxes. A self-attention mechanism highlights significant features, and multi-scale detection ensures accurate object sizing.



5. Conversion and Deployment

YOLOv8's design also emphasizes ease of deployment, including compatibility with TensorFlow Lite for mobile and embedded applications. However, the conversion process from PyTorch to TensorFlow Lite involves multiple steps and poses challenges, such as maintaining algorithmic functions like non-maximum suppression across formats.

In summary, YOLOv8 represents a significant advancement in object detection technology, offering enhanced performance, flexibility, and application range. Its architecture and features are tailored to meet the demands of modern, real-time object detection tasks, setting a new benchmark for speed and accuracy.

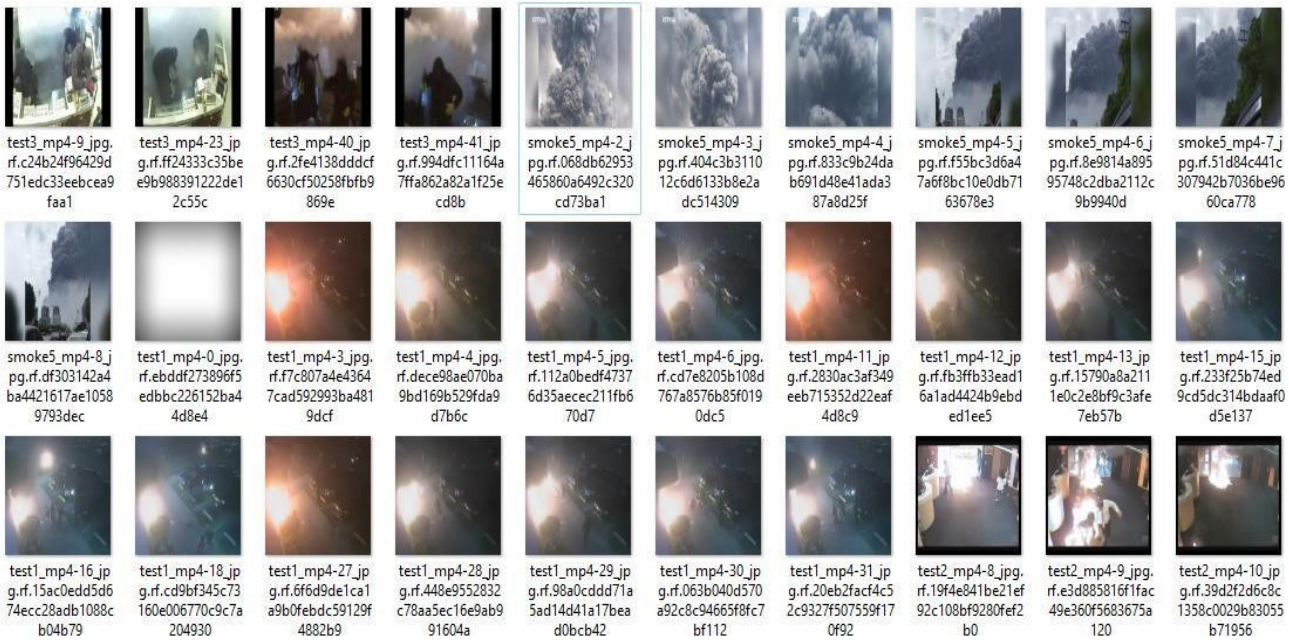


Figure 2. Fire Data set partial display.

5.1 YOLO Model Training

In our project, we utilized the YOLO (You Only Look Once) model, specifically the YOLOv8s architecture, for object detection tasks. This state-of-the-art deep learning algorithm is renowned for its efficiency and accuracy in detecting objects in images or videos. By employing YOLOv8s, we aimed to achieve real-time object detection capabilities with high precision and recall rates. The model was trained using a dataset in YAML format, with an input image size of 800x800 pixels. During training, the model underwent 20 epochs to learn and refine its ability to detect objects, such as smoke and fire, in the given images. The training process also involved generating plots to visualize key metrics like loss curves and precision-recall curves, aiding in monitoring the model's performance and convergence. The utilization of the YOLO model brought advanced object detection capabilities to our project, ensuring robustness and accuracy in identifying target objects within images.

6. Evaluation Metrics

Fine-tuning YOLOv8 involves adjusting the pre-trained model to better suit specific or novel datasets, which can significantly enhance its performance in specialized object detection tasks. Here are key considerations and steps for fine-tuning YOLOv8 effectively: We started with a YOLOv8 model pre-trained on a large and diverse dataset, such as COCO. This model already has learned features that are generally applicable across various object detection tasks. Ensure the custom dataset is annotated accurately. The annotations should include bounding boxes and class labels for the objects of interest.

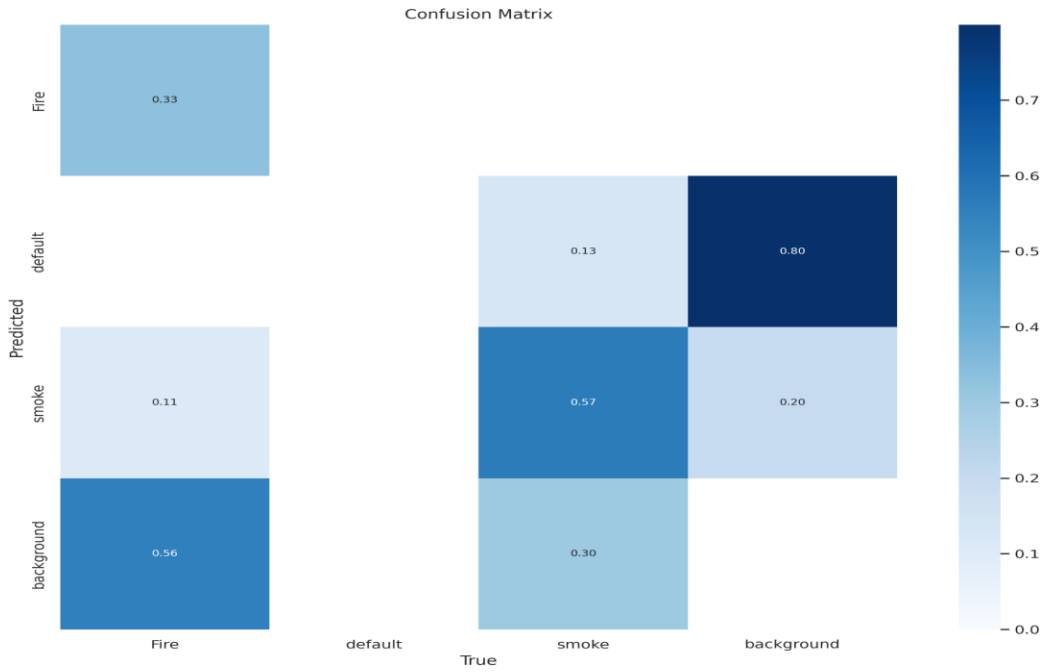


Figure 3: Confusion Matrix

7. Data Collection

The level of precision of the deep learning model primarily depended on the dataset used in the training and validation procedures. We gathered fire images from various open access sources such as GitHub and Roboflow, finding images depicting a range of different conditions (shape, color, size, indoor and outdoor environment). The dataset was store in rob flow.

7.1 Dataset Overview

The dataset is partitioned into training, validation, and testing sets, with a distribution ensuring a balanced representation of images and question-answer pairs across each subset. As shown in Table 1, the dataset is divided into three distinct sets: the training set, validation set, and testing set. This division ensures a comprehensive evaluation of the models trained using the dataset, with a significant number of image-question pairs allocated to each set. The training set constitutes 89% of the dataset, while both the validation and testing sets comprise 11% each.

Category	**Trainings set**	**Validation set**	**Testing**	**Total**
Images	877	47	55	979

Table 1. Composition of the Fire dataset with total count

8.Results

8.1. YOLOv8 Model:

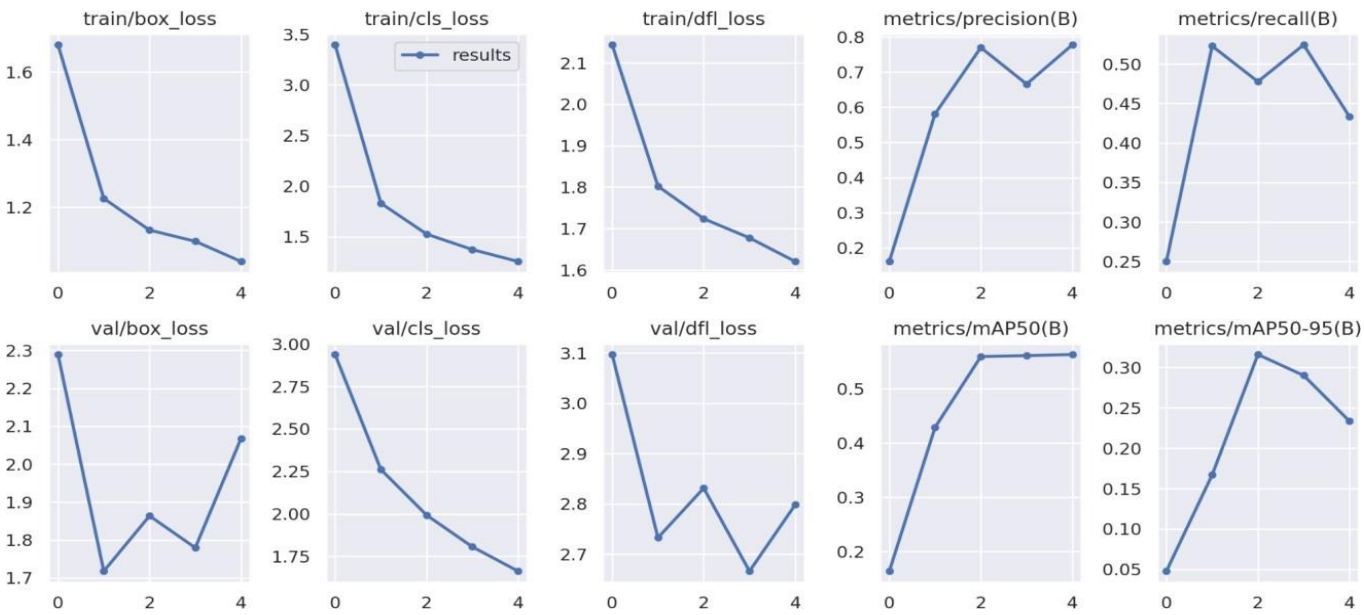
The fine-tuned YOLO model produces noteworthy results, demonstrating its efficacy in the given context:

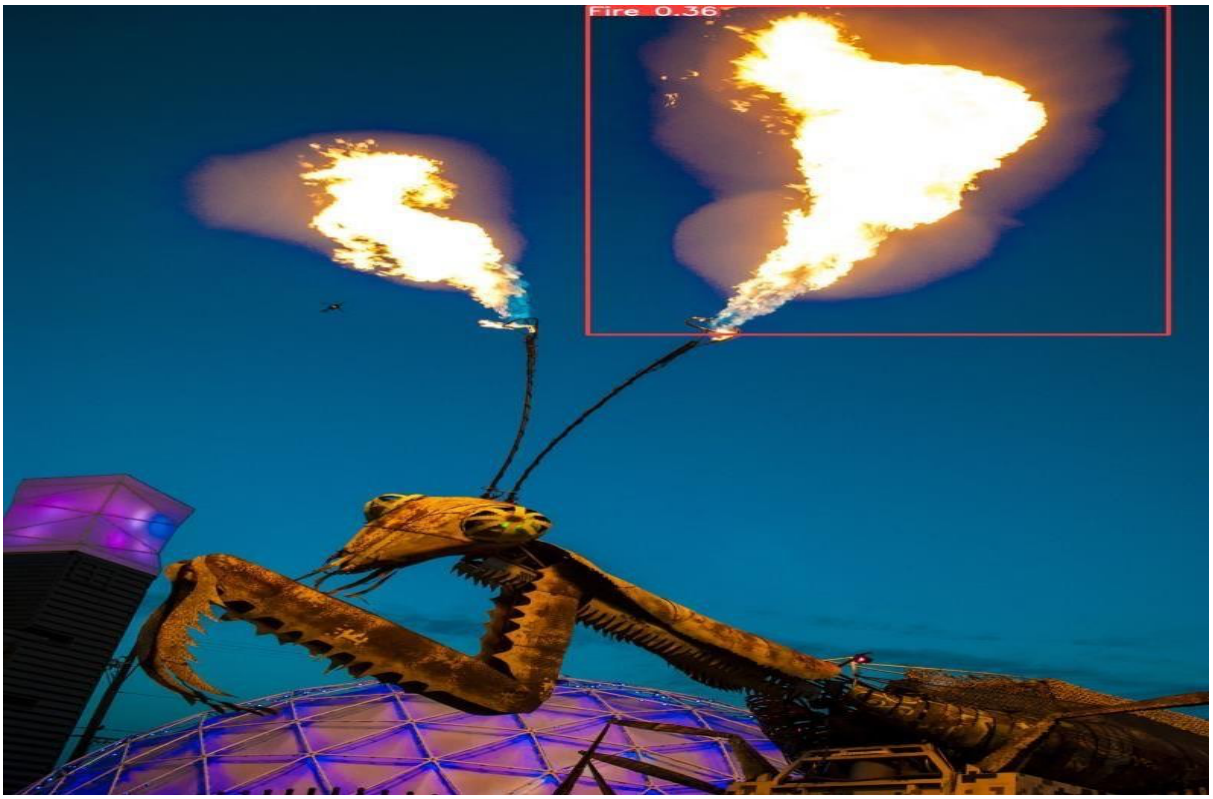
- **box loss: 0.9655.**
- **classification loss: 0.7124**

In our project, we leveraged a pre-trained YOLOv8 model that had been trained on the COCO (Common Objects in Context) dataset, which consists of various object classes but lacks specific classes for fire and smoke. Our objective was to apply this pre-trained model to the task of fire and smoke detection in images.

However, due to the absence of fire and smoke classes in the COCO dataset, the pre-trained model was not capable of detecting these objects accurately. As a result, during the evaluation phase, we observed that the model failed to identify instances of fire and smoke in the images provided for testing.

This limitation highlights the importance of training object detection models on datasets that include specific classes of interest to ensure robust detection performance.





8.2 Custom Object Detection Model:

Evaluation Metrics for Custom Object Detection Model:

An Accuracy of 0.50 refer to the overall correctness of the detected objects.

Precision of 0.25 refers to the accuracy of the model in identifying objects as belonging to a specific class.

A Recall of 0.50 indicates the model's ability to find all relevant objects in an image.

A F1 score of 0.33 provides a balance between precision and recall.

F1 Score	0.3333
Precision	0.2500
Recall	0.5000
Accuracy	0.5000

8.3 Visualization of Evaluation Metrices:

Accuracy (Blue Bar): The accuracy of the model lies in and around 0.5, represented by the blue bar.

This means that this model will be able to make a correct prediction in 50 out of 100 cases.

Precision (Green Bar): The approximate value of precision is 0.25 from the green bar.

That means the model does a true positive prediction around 25% of the time.

Represented in the orange bar, it is approximately 0.5.

This metric basically says that the model will identify positive cases correctly half the time.

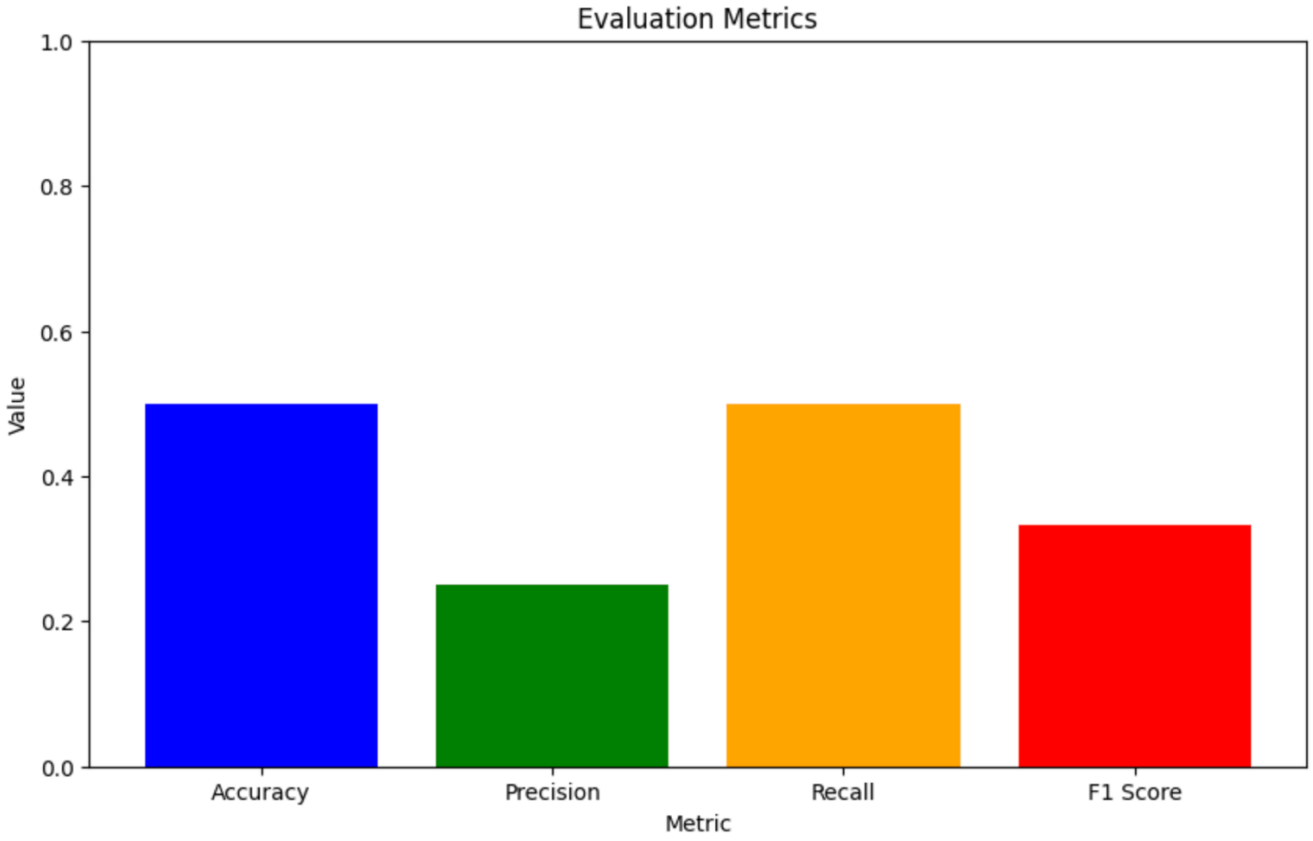
F1 Score (Red Bar): The red bar illustrates the F1 score, which is around 0.3333.

From this value, it is only visible that the F1 score will have a moderate balance between precision and recall, yet the score is relatively low to show that the precision and recall of the model is not very high.

These present the model with moderate values of accuracy and recall but highly bounded precision, and,

as a consequence, low F1 score.

That, in turn, may reveal an obvious lack of specificity and sensitivity, i.e., false negatives and positives. hence, this may be of the most critical importance, depending upon how the model is used.



9.Fine Tuning of YOLOv8 Model

In our pursuit of improving the fire and smoke detection capabilities of our system, we embarked on a journey of fine-tuning a pre-trained YOLOv8 model using a dedicated smoke and fire dataset. This fine-tuning process was essential to tailor the model's parameters and features specifically for the detection of smoke and fire instances, which are critical elements in various applications such as fire prevention, safety monitoring, and disaster management. The smoke and fire dataset we utilized contained a diverse range of images showcasing different scenarios and variations of smoke and fire, ensuring that our model could generalize well to real-world situations.

The fine-tuning procedure involved retraining the YOLOv8 model using the smoke and fire dataset, which consisted of annotated images with bounding box coordinates for smoke and fire regions. By feeding this dataset into the model and adjusting its weights and biases during training, we aimed to enhance the model's ability to accurately detect and localize smoke and fire objects within images.

10. Custom Model Training

The process for training a custom object detection model using PyTorch begins by initializing essential variables such as the number of epochs for training and the directory path for storing model checkpoints and loss history.

The training loop is structured to iterate through each epoch, during which the model is set to training mode and the cumulative loss for the epoch is tracked. Progress bars, facilitated by the ‘tqdm’ library, offer real-time monitoring of the training process, displaying epoch and batch information as the model progresses through the dataset.

Within each epoch, the model undergoes forward pass computations to obtain classification and regression outputs for the input data batches. Subsequently, classification and regression losses are calculated for each output, contributing to the overall loss for the epoch. This loss is backpropagated through the model, facilitating parameter updates via optimization. The training loop also computes the epoch loss by dividing the cumulative loss by the total number of batches, providing a measure of the model's performance throughout the training process.

Periodically, we check whether the current epoch is a multiple of 20, signaling the need for checkpoint saving and loss history recording. Model checkpoints, containing the model's state dictionary, are saved ensuring that the model's progress can be restored or continued later. Additionally, the epoch loss history is stored in a text file for analysis and visualization. After training completes, the final trained model is saved for future use or deployment.

Overall, showcases a comprehensive approach to train custom object detection model in PyTorch, incorporating essential components such as model checkpoints, progress monitoring, loss tracking, and file management to facilitate effective and efficient model training.

11. **Discussion**

The YOLOv8 model for fire detection, which is presented in this paper, has produced pleasing outcomes for small- scale fire detection and fire detection in varying light conditions. It is unable to not only offer real-time detection but also exhibit strong practical robustness. However, in our tests, we discovered that the detection algorithm is still plagued by low detection accuracy and difficulty detecting semi-occluded targets. This phenomenon, which presents a challenge for fire inspectors, may be caused by the unpredictability of fire and the complexity of fire spread in the actual environment. It is important to note that the current target detection model is tasked with solving this urgent problem.

The fact that these challenges are solvable is encouraging. Indeed, the corresponding deep learning training techniques, such as translation-invariant transformation, random geometric transformation, and random color dithering, on the training image in the model's training, can be arbitrarily chosen in the detection algorithm, and the results are encouraging. Future research will optimize the Fire-YOLOv8 model's training procedure with an emphasis on image preprocessing. Through transfer learning, it is hoped that FireYOLOv8's capacity for generalization will be further enhanced.

12.Conclusion

The experimental results and evaluation proved that the YOLOv8 model was robust and performed well for our fire detection dataset. The proposed fire detection method is effective and can be used in various applications, allowing researchers to detect fires at an early stage. Furthermore, an object mapping method can be applied to understand the system whether the fire is intentional or unintentional.

The evaluation metrics of Custom Model provide insights into different aspects of the model's performance in object detection tasks, including its ability to correctly classify objects, its sensitivity to detecting relevant objects, and the balance between precision and recall.

The combination of architectural improvements, effective feature representation, direct optimization for object detection, and efficient inference contribute to the superior performance of YOLOv8 in object detection tasks.

The YOLOv8 model likely outperformed the custom object detection model through several key improvements and optimizations tailored specifically for object detection tasks. Here’s a detailed look at the potential factors contributing to YOLOv8's superior performance:

1.**Advanced Architectural Features**: YOLOv8 may incorporate more sophisticated neural network architectures that enhance its ability to process and learn from image data. These improvements can include deeper layers, more efficient connections between layers, or specialized convolutional features that are better suited for the specific nuances of object detection, such as varying object scales and occlusions.

2. **Improved Feature Representation**: YOLOv8 likely uses advanced techniques for feature extraction and representation, which allow it to better capture the essential characteristics of objects within diverse environments. This can involve using anchor boxes that are better tuned to the specific dimensions and shapes of the target objects or employing spatial separable convolutions to reduce computational complexity while maintaining or enhancing performance.

3. **Efficient Inference Mechanisms:** One of the critical advantages of YOLO models is their speed and efficiency during inference. YOLOv8 may have introduced optimizations that reduce latency and computational demand, making the model faster without sacrificing accuracy. This is crucial for applications like fire detection, where rapid response times can be vital.

4**. Direct Optimization for Object Detection**: Unlike some general-purpose models that are adapted to object detection, YOLOv8 is likely directly optimized for this task. This includes loss functions specifically designed to improve detection accuracy, such as combining aspects of localization (bounding box prediction) and classification accuracy in a way that directly correlates to practical detection performance.

5. **Robust Training Procedures**: YOLOv8 may benefit from enhanced training procedures that involve a wider variety of training data, augmented with real-world scenarios. This can include exposure to challenging lighting conditions, varied backgrounds, and occlusions. Additionally, techniques like transfer learning from pre-trained models on similar tasks might have been employed to give the model a better starting point for learning.

6. **Handling of Class Imbalance:** In object detection, some classes are more frequent than others, which can skew a model's ability to learn less frequent but critical classes. YOLOv8 might implement more effective mechanisms to handle class imbalance, such as focal loss or oversampling of minor classes.

7. **Integration with Contextual Information:** For tasks like distinguishing between intentional and unintentional fires, YOLOv8 might incorporate contextual information into the detection process. This could involve analyzing not just the presence of fire, but also the surrounding environment, typical fire behavior, and other indicators that could suggest the fire’s nature.

These factors collectively contribute to YOLOv8's enhanced performance over custom models, especially in complex and dynamic scenarios like fire detection, where accuracy, speed, and contextual understanding are crucial.

13.Responsibility:

Name	Report
Shilpa Kuppili	1. Data Collection 2. Data Preprocessing 3. Model Architecture, Hyper parameter Tuning and Deployment
Sathwik Kuchana	1. Data Preprocessing 2. Learning Parameters & Model Architecture 3. Model Deployment
Sri Sai Lalitha Mallika Yeturi	1. Data Collection 2. Data Preprocessing 3. Model deployment
Priyadarshini Munigala	1. Aim and Workflow 2. Data Collection & Processing 3. Model Checking & Evaluation

Table 4. Responsibility

14. References:

[1] Ahrens, M. and Maheshwari, R., 2021. Home Structure Fires report | NFPA. [online] Nfpa.org. Available at: [Accessed 12 August 2022].

[2] Zhao, C.; Feng, Y.; Liu, R.; Zheng, W. Application of Lightweight Convolution Neural Network in Cancer Diagnosis. In Proceedings of the 2020 Conference on Artificial Intelligence and Healthcare, Taiyuan, China, 23–25 October 2020; pp. 249–253.

[3] Muhammad, K.; Ahmad, J.; Baik, S.W. Early fire detection using convolutional neural networks during surveillance for effective disaster management. Neurocomputing 2018, 288, 30–42. [CrossRef]

[4] J. Seebamrungsat, S. Praising and P. Riyamongkol, "Fire detection in the buildings using image processing," 2014 Third ICT International Student Project Conference (ICT ISPC), 2014, pp. 95-98, doi: 10.1109/ICTISPC.2014.6923226.

[5] Celik, T., 2010. Fast and Efficient Method for Fire Detection Using Image Processing. ETRI Journal, 32(6), pp.881-890.

[6] Jindal, P.; Gupta, H.; Pachauri, N.; Sharma, V.; Verma, O.P. Real-Time Wildfire Detection via Image-Based Deep Learning Algorithm. In Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing; Sharma, T.K., Ahn, C.W., Verma, O.P., Panigrahi, B.K., Eds.; Springer: Singapore, 2021; Volume 1381. [CrossRef]

[7] Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., Kwon, Y., Fang, J., Michael, K., V, A., Montes, D., Nadar, J., Skalski, P., Wang, Z., Hogan, A., Fati, C., Mammana, L., Patel, D., Yiwei, D., You, F., Hajek, J., Diaconu, L. and Minh, M., 2022. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and Open VINO Export and Inference. [online] Zenodo. Available at: https://zenodo.org/record/6222936#.vYVX nZB 5e [Accessed 8 July 2022].

[8] MOSES, O., 2019. GitHub - OlafenwaMoses/FireNET: A deep learning model for detecting fire in video and camera streams. [online] GitHub. Availableat:[Accessed 1 Augus