

Multimodal Computer Vision for Rapid Disaster Damage Analysis and Victim Detection using Deep Learning

MR.ADITYA ARUN MANDLE

Ashokrao Mane Group Of Institution, Vathar
Department of Computer Science and Engineering
adityamandle024@gmail.com

MR.VEDRAJ BALASAHEB MANE

Ashokrao Mane Group Of Institution, Vathar
Department of Computer Science and Engineering
neved19@gmail.com

MR.ATHARV ANIL KUMBHAR

Ashokrao Mane Group Of Institution, Vathar
Department of Computer Science and Engineering
atharvkumbhar25@gmail.com

MR.SHIVAM RAJARAM SURYAVANSHI

Ashokrao Mane Group Of Institution, Vathar
Department of Computer Science and Engineering
suryawanshishivam977@gmail.com

MR.MOHAN CHINTAMAN MAHAJAN

Ashokrao Mane Group Of Institution, Vathar
Department of Computer Science and Engineering [ma-](mailto:ma-neved19@gmail.com)
mohanmahajan477@gmail.com

MS.PRIYANKA.R.JADHAV

Ashokrao Mane Group Of Institution, Vathar
Department of Computer Science and Engineering
prj@amgoi.edu.in

Abstract : The rapid assessment of disaster-affected environments demands intelligent and computationally efficient visual analysis systems capable of operating in real-time under constrained hardware conditions. This paper presents a hybrid computer vision framework for real-time disaster scene analysis that integrates YOLOv8n instance segmentation with classical image processing algorithms, specifically HSV color-space thresholding and Canny-based edge density estimation, to perform concurrent victim detection, fire and smoke identification, and structural damage assessment. The system is built on a FastAPI backend that serves live MJPEG streams from a webcam feed and supports upload-based static frame analysis, with a web-based frontend enabling real-time annotated visualization. Two YOLO models are deployed within the pipeline: YOLOv8n-seg for victim segmentation and a pretrained YOLO11x model for object-level structural context detection. A late-stage data fusion layer consolidates outputs from all sub-modules into a unified detection matrix. Experimental evaluation conducted on standard CPU hardware without GPU acceleration demonstrates streaming inference rates of approximately 13 FPS for victim tracking at 640×480 resolution and 24 FPS for scene analysis at 320×240 resolution, with comprehensive static-frame analysis completing in approximately 850–890 milliseconds. Qualitative evaluation across diverse scene

categories reveals reliable victim segmentation, effective fire and smoke localization, and meaningful structural chaos quantification. The proposed architecture serves as a practical, deployment-ready foundation for automated disaster monitoring in resource-constrained environments.

INDEX TERMS: Disaster Scene Analysis, YOLOv8, Instance Segmentation, HSV Thresholding, Canny Edge Detection, Real-Time Streaming, FastAPI, Computer Vision Fusion, MJPEG, Structural Damage Estimation.

1.INTRODUCTION

Natural disasters, including earthquakes, floods, wildfires, and industrial accidents, cause widespread destruction and demand immediate, coordinated emergency response. A critical bottleneck in early disaster response is situational awareness — the ability of rescue teams to rapidly understand the scale and nature of hazards present in an affected zone. Traditional reconnaissance methods, which depend on manual inspection or delayed aerial photography, are increasingly inadequate given the pace at which disaster scenarios evolve. Automated computer vision systems offer a compelling alternative, enabling continuous, objective, and real-time analysis of disaster scenes from video streams captured by ground-based cameras or unmanned aerial vehicles.

The evolution of deep learning, particularly convolutional neural network architectures designed for object detection and segmentation, has transformed the practical ceiling of what such automated systems can achieve. The YOLO (You Only Look Once) family of models represents one of the most significant advances in this domain, introducing single-pass inference capable of real-time object localization and classification. With the release of YOLOv8 by Ultralytics, the architecture was further extended to support instance segmentation, enabling not just bounding box detection but per-pixel mask generation for each identified object. The nano variant of this model, YOLOv8n-seg, achieves this functionality while maintaining a model footprint of approximately 7 MB, making it particularly well-suited to CPU-based field deployments.

However, a significant limitation remains when applying general-purpose YOLO models to disaster-specific scenes. While YOLOv8n-seg, pretrained on the Microsoft COCO dataset, reliably detects persons — a critical target in victim recovery operations — it does not natively identify disaster-specific environmental hazards such as active fire, rising smoke plumes, or structurally compromised buildings. Training dedicated neural networks for each of these classes requires large annotated disaster-scene datasets, substantial computational resources for fine-tuning, and domain expertise. For a team developing and deploying a system in a practical, resource-constrained setting, this represents a significant barrier.

To resolve this tension between detection breadth and computational feasibility, this work proposes a hybrid architecture that assigns specific analytical responsibilities to the most computationally appropriate method available. Victim detection, which benefits greatly from the spatial reasoning and feature extraction capabilities of deep learning, is handled entirely by the YOLOv8n-seg model. Environmental hazard detection — specifically fire and smoke — is addressed using HSV color-space thresholding, an algorithm requiring negligible compute that can reliably isolate chromatic signatures associated with combustion and atmospheric particulates. Structural damage estimation relies on Canny edge density analysis, which measures the degree of visual fragmentation in a scene as a proxy for physical destruction. A complementary YOLO11x pretrained model provides additional object-level structural context in upload-based analysis mode.

2. LITERATURE REVIEW:

Research at the intersection of computer vision and disaster response has grown substantially over the past decade, driven by advances in deep learning and the increasing availability of aerial and ground-level imagery captured during real-world events. The literature relevant to this work spans object detection architectures, disaster-specific segmentation methods, fire and smoke detection algorithms, and edge-deployable inference systems.

2.1. Object Detection and Segmentation Architectures:

The foundational work of Redmon et al. [1] established the YOLO framework as a landmark achievement in real-time object detection. By reformulating detection as a single regression problem solved in one network pass, YOLO eliminated the multi-stage proposal mechanisms of earlier detectors such as R-CNN [2], dramatically reducing inference time. Subsequent versions introduced improvements in anchor-free detection, feature pyramid networks for multi-scale object handling, and decoupled prediction heads. Bochkovskiy et al. [3] presented YOLOv4, incorporating several training optimizations — including mosaic augmentation, self-adversarial training, and CSPNet backbones — that substantially improved accuracy without proportional increases in computational cost.

2.2. Disaster Scene Analysis and Building Damage Assessment:

Remote sensing-based building damage assessment has been a particularly active research area. Sublime and Kalinicheva [6] demonstrated the use of deep learning on satellite imagery pairs captured before and after disaster events to map structural destruction. While achieving high accuracy, this approach requires access to pre-event baseline imagery and is not applicable to continuous ground-level video feeds. Gupta et al. [7] developed the xBD dataset and an associated benchmark for building damage classification from satellite imagery, establishing useful evaluation standards for the field. However, model performance observed in satellite-image contexts does not translate directly to the ground-level scenario addressed in this work.

2.3. Fire and Smoke Detection:

Fire and smoke detection represents a well-studied computer vision problem with a long history predating the deep learning era. Çelik and Demirel [9] proposed one of the earliest HSV-based fire detection systems, demonstrating that the characteristic hue, saturation, and brightness distributions of active flames could be reliably isolated through carefully calibrated color-space thresholds. Their work established that color-space analysis requires negligible computation compared to any neural inference approach, making it inherently attractive for resource-constrained deployments. Phillips et al. [10] extended this analysis by systematically comparing multiple color-space representations for pixel-level fire detection, confirming HSV's suitability while documenting its known weakness under variable illumination conditions. Khan et al. [11] addressed the false positive problem through a two-stage approach combining HSV thresholding with motion analysis, reasoning that genuine fire regions exhibit flickering motion patterns absent in static orange or red backgrounds. This insight motivates the variance gate employed in the smoke detection module of the proposed system, which uses texture variance as a discriminating criterion to separate genuine diffuse smoke plumes from structurally similar blank surfaces. On the neural side, Muhammad et al. [12] demonstrated that fine-tuning lightweight CNN architectures such as SqueezeNet on fire-specific datasets produces notably higher precision than color-based methods. However, this approach presupposes the availability of a labeled fire dataset and a training pipeline, neither of which is assumed in the deployment context of this work. The trade-off between the precision of trained neural classifiers and the zero-training-overhead advantage of HSV thresholding is a central design consideration in the proposed hybrid architecture, which accepts the precision limitations of color-based fire detection in exchange for immediate deployability.

3. PROPOSED SYSTEM:

3.1. System Overview:

The proposed system is a full-stack web application comprising a Python-based FastAPI backend and a browser-based frontend, designed to provide real-time visual analysis of disaster scenes. The backend manages YOLO model inference, classical image processing, data fusion, and MJPEG stream generation. The frontend renders annotated video feeds, exposes mode selection controls, and handles static image upload for comprehensive analysis.

The architecture is deliberately modular, separating inference, analysis, and presentation concerns to enable independent scaling or replacement of each component.

3.2. Functional Modules:

The system is organized into six functional modules, each responsible for a distinct aspect of the overall analytical pipeline.

1. Video Input Module:

This module handles the acquisition of visual data from two sources. In streaming mode, it captures frames continuously from the host system's webcam using OpenCV's VideoCapture interface, operating in a dedicated background thread to prevent blocking the main API event loop. A retry mechanism with exponential backoff handles temporary webcam unavailability, and informative error frames are generated and streamed when the camera cannot be accessed. In upload mode, image files submitted via the REST API are decoded from binary buffers into OpenCV-compatible numpy arrays, bypassing the streaming pipeline entirely.

2. Preprocessing Module:

Prior to inference, each frame undergoes lightweight preprocessing. For streaming inference, frames are conditionally downscaled to 320×240 pixels during scene analysis mode to maintain target frame rates, while victim tracking operates at the native 640×480 capture resolution. Gaussian smoothing is applied within the structural damage estimation subroutine to reduce edge detection noise. No additional normalization is required at the application level, as the YOLO inference engine internally applies model-specific input preprocessing.

3. Deep Learning Segmentation Module:

This module houses the YOLOv8n-seg model for victim detection and the YOLO11x pretrained model for object-level structural context. YOLOv8n-seg is loaded once at startup and retained in memory for the system's lifetime, enabling sub-millisecond model loading overhead during inference. Predictions are filtered to the person class (COCO class ID 0) for victim mode, isolating precisely the detection category most relevant to rescue operations. The YOLO11x model is used exclusively in

upload-based damage analysis to enumerate structural elements such as vehicles and infrastructure objects, providing contextual evidence for damage assessment without the latency penalty of real-time execution.

4. Algorithmic Hazard Detection Module:

Operating in parallel with the neural segmentation pipeline, this module implements two classical image processing routines. The fire and smoke detection routine, implemented in `visual_utils.py`, converts each frame to HSV color space and applies multi-range thresholding across red, orange, and yellow hue bands to construct a fire mask. Smoke is isolated using a low-saturation, moderate-brightness thresholding rule, further refined by a texture variance gate that discriminates between actual smoke plumes and structurally similar backgrounds. The structural damage estimation routine, implemented in `sam_utils.py`, applies dual-threshold Canny edge detection to a Gaussian-smoothed grayscale image, computing the fraction of pixels exceeding the edge threshold as a proxy for scene irregularity.

5. Data Fusion Module:

Results from the neural and classical modules are consolidated in a late-stage fusion operation. Detections produced by the algorithmic module are assigned synthetic class identifiers — class ID 101 for fire, 102 for smoke, 100 for structural damage, and 200 for building — and formatted as bounding box tensors compatible with the YOLO results structure. These synthetic boxes are concatenated with the original YOLO detection tensor, producing a unified results object that is subsequently passed to the rendering pipeline. This design ensures that the same rendering code handles both neural and algorithmically generated detections uniformly, simplifying the downstream processing chain.

6. User Interface Module

The frontend, implemented in HTML, CSS, and JavaScript, provides a real-time annotated view of the disaster scene. Two operational modes are exposed: Victim Mode, which streams the YOLOv8n-seg segmentation output with green mask overlays on detected persons, and Damage Mode, which displays the fused output with color-coded overlays for fire (red), smoke (gray), structural damage (orange), and detected buildings (blue-orange). A static image upload panel enables users to submit

frames for comprehensive analysis, receiving a base64-encoded annotated image and detailed detection statistics in response.

3.3. System Workflow:

The operational workflow of the proposed system proceeds as follows. Upon server startup, both YOLO models are loaded into memory and a background thread is initiated to continuously capture frames from the webcam. Each captured frame is passed to the `model_manager.predict()` function, which routes it through the appropriate analytical pipeline based on the currently active mode. The processed frame, annotated with mask overlays and mode metadata, is encoded as JPEG and inserted into the MJPEG stream boundary sequence for delivery to connected browser clients. Mode transitions — switching between victim and damage analysis — are triggered by API calls to `/segment/webcam/{mode}`, which update a global mode variable that the background thread reads on each inference cycle.

For upload-based analysis, the client submits a multipart form POST request to `/segment/image`, including the image file and desired mode. The backend decodes the image, applies the full inference pipeline without streaming optimizations, and returns a JSON response containing a base64-encoded annotated image, detected class counts, confidence scores, and the computed damage percentage.

The outputs generated from both processing branches are integrated through a late-stage fusion mechanism that unifies detections into a single structured representation. The fused output contains segmentation masks, bounding boxes, contour annotations, and heuristic detection regions corresponding to disaster-related events such as victims, smoke, fire, and damaged structures. These annotations are rendered onto the original frame using overlay operations and contour highlighting techniques to improve scene interpretability. Finally, the processed frame is encoded and transmitted to the frontend either as an MJPEG stream for continuous monitoring or as a base64-encoded image for static analysis results. The overall workflow emphasizes computational efficiency, modularity, and low-latency processing, making the proposed framework suitable for resource-constrained disaster response environments and edge-computing deployments.

4. SYSTEM ARCHITECTURE

4.1. Overall Architecture:

The system architecture is organized into four principal layers: the presentation layer, the API layer, the inference layer, and the processing layer. This layered decomposition reflects standard principles of separation of concerns and promotes maintainability and testability of individual components.

The presentation layer consists of the browser-based frontend, which communicates with the API layer via HTTP. For live streaming, the frontend sets the src attribute of an HTML img element to the MJPEG stream endpoint, which causes the browser to establish a persistent HTTP connection and render each JPEG frame as it arrives. For upload-based analysis, the frontend uses the Fetch API to submit form data and parse the JSON response, updating the interface with the returned annotated image and detection statistics.

The API layer is implemented using FastAPI, a modern Python web framework built on ASGI (Asynchronous Server Gateway Interface). FastAPI handles request routing, input validation, file upload processing, and response serialization. CORS middleware is configured to permit cross-origin requests from the frontend, enabling deployment with the frontend and backend on separate ports or hosts. The MJPEG stream is implemented as a FastAPI StreamingResponse with the multipart/x-mixed-replace media type, which instructs the browser to replace each received body part with the next, achieving smooth video playback without client-side JavaScript polling.

The inference layer is managed by the Model-Manager class, which encapsulates all model loading, prediction routing, and streaming optimization logic. A caching mechanism is implemented for the building detection sub-step during streaming: YOLO11x predictions are executed at most once per second and cached between frames, preventing the 114 MB model from bottlenecking the streaming frame rate. The lightweight YOLOv8n-seg model, at 7 MB, is used as a fast substitute for building context detection during streaming, with YOLO11x reserved for the more thorough upload-based analysis path.

The processing layer encompasses the algorithmic subroutines in visual_utils.py and sam_utils.py, which perform HSV thresholding, texture variance computation, Gaussian smoothing, multi-scale Canny edge detection, contour analysis, and the final damage score aggregation. These routines operate on raw numpy arrays and return structured Python dictionaries, maintaining

clean interfaces with the fusion logic in the inference layer above.

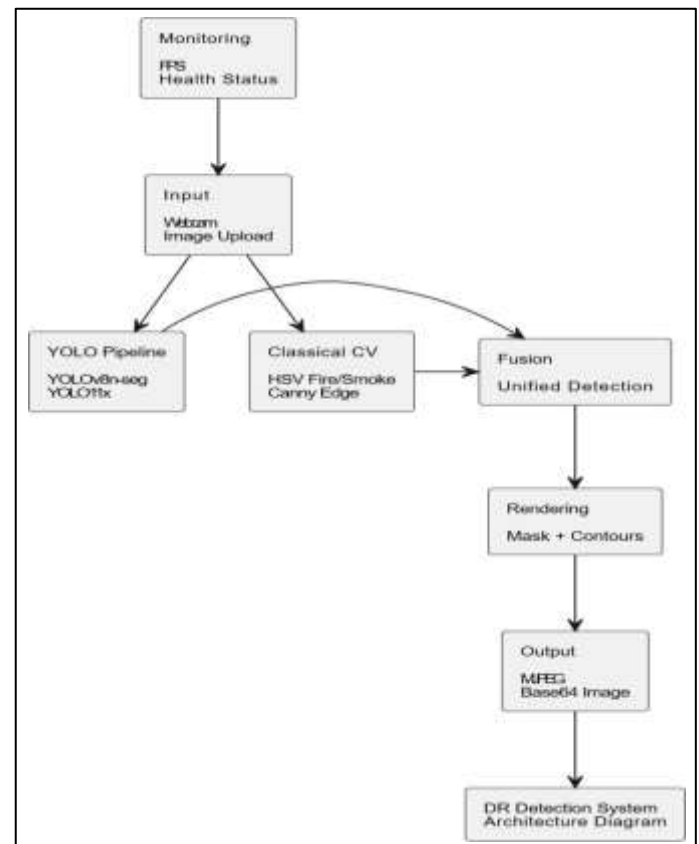


Fig 4.1 System Architecture

The architecture block diagram illustrates the flow of data from two input sources — the webcam (for streaming) and the upload interface (for static analysis) — into the FastAPI backend. Within the backend, separate branches represent the YOLO inference path (YOLOv8n-seg for victims; YOLO11x for structural context) and the classical processing path (HSV thresholding for fire/smoke; Canny edge analysis for structural damage). Both branches feed into the late-stage fusion module, which produces the unified detection output. The output is then rendered onto the original frame as colored mask overlays and contour annotations, before being encoded and returned to the frontend as either an MJPEG stream or a base64-encoded static image. A health check endpoint provides webcam status and current FPS metadata to the frontend.

4.2. DFD Diagram:

4.2.1 Level 0 DFD:

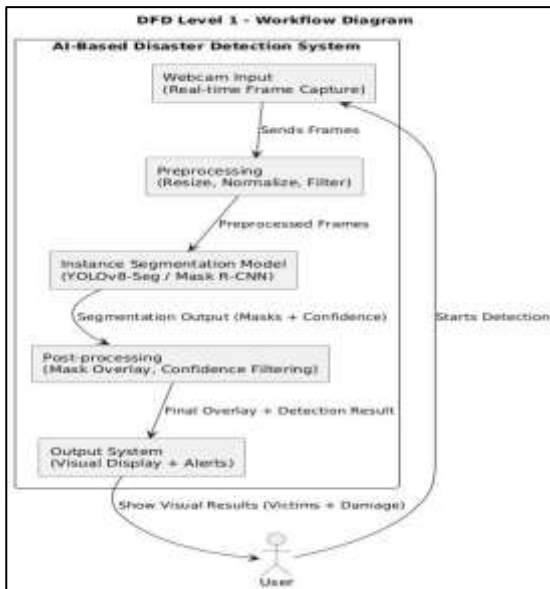


Fig 4.2 Level 0 Data Flow Diagram

The Level 0 DFD represents the system as a single process interacting with two external entities: the Emergency Responder (user) and the Video Source (webcam or uploaded image). The Emergency Responder sends a mode selection command and optionally an image file to the Disaster Analysis System. The Video Source provides a continuous video stream. The system outputs an annotated visual feed and structured detection metadata back to the Emergency Responder.

4.1.2 Level 1 DFD:

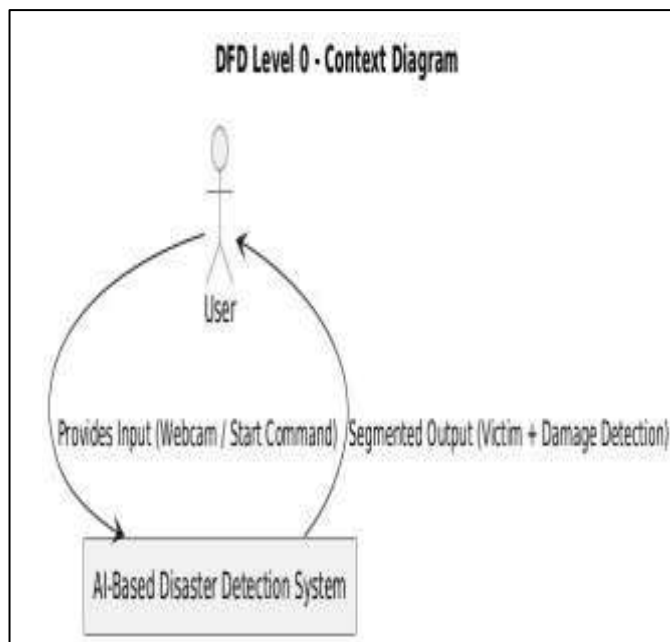


Fig 4.3 Level 1 Data Flow Diagram

The Level 1 DFD decomposes the single system process from Level 0 into five distinct sub-processes, each corresponding to a functional module in the implementation.

5. METHODOLOGY

5.1. Victim Detection via Instance Segmentation:

Human presence detection is implemented using the YOLOv8n-seg model pretrained on the Microsoft COCO dataset. The model is loaded at server initialization and retained in GPU-free memory for the application's lifetime, enabling consistent sub-second response times. Inference is constrained to COCO class ID 0 (person) by passing the classes=[0] argument to the model's predict() method, which prevents the model from allocating computation toward irrelevant COCO categories such as household objects or furniture. This class filtering contributes meaningfully to inference speed, as the model skips non-person proposals during non-maximum suppression.

Detected instance masks are extracted from the results object as binary tensors and resized to match the input frame dimensions using OpenCV's resize function. Each mask is then alpha-blended onto the original frame at an opacity of 0.4 using cv2.addWeighted, providing a translucent green overlay that preserves the victim's visual appearance while clearly indicating their spatial extent. Contour lines are additionally drawn at the mask boundaries using cv2.drawContours, enhancing visibility in cluttered backgrounds. This rendering strategy was chosen over solid mask fills to maintain the interpretability of the underlying scene, which is important for responders making real-time decisions based on the annotated feed.

The confidence threshold for victim detection is set to 0.25, a value deliberately lowered from the default 0.5 to minimize false negatives in disaster scenes where victims may be partially occluded by debris, obscured by smoke, or adopting unusual postures inconsistent with upright standing. Non-maximum suppression is applied with an IoU threshold of 0.45, which is the standard YOLO default, providing adequate separation between overlapping detections without excessive fragmentation.

5.2. Fire and Smoke Detection via HSV Thresholding:

Fire detection exploits the characteristic chromatic properties of combustion in the HSV color space. Active flames exhibit hue values concentrated in the red ($0\text{--}10^\circ$, $170\text{--}180^\circ$) and orange-yellow ($11\text{--}35^\circ$) ranges, with high saturation values (typically above 120) indicating vivid coloration rather than washed-out backgrounds. The system constructs three binary masks corresponding to these hue ranges and combines them using pixel-wise addition. Morphological opening with a 5×5 kernel removes isolated noise pixels and small spurious detections, and subsequent morphological dilation with two iterations expands the mask slightly to capture full flame boundaries. Contours extracted from the final fire mask that exceed 0.5% of the total image area are recorded as fire detections, each assigned a fixed confidence score of 0.85 reflecting the high reliability of large, spectrally pure flame regions.

Smoke detection employs a complementary strategy targeting the low-saturation, moderate-brightness signature of smoke plumes. A pixel is included in the smoke candidate mask if its HSV saturation falls below 50 and its brightness value lies between 100 and 240. The upper brightness bound of 240 is critical for excluding overexposed white surfaces such as blank walls or ceilings, which would otherwise generate frequent false positives. A second-stage filtering gate computes the local texture variance within each smoke candidate bounding region; regions with variance below 1000 (indicating smooth, low-contrast areas consistent with diffuse smoke) are retained, while high-variance regions that likely correspond to debris or structural surfaces are discarded. Smoke contours exceeding 1% of image area are recorded as smoke detections with a confidence of 0.75.

5.3. Structural Damage Estimation via Edge Density Analysis:

Structural integrity assessment is performed using a Canny edge density metric computed from the grayscale representation of the input frame. The algorithm begins by converting the color image to grayscale and applying a 5×5 Gaussian blur to suppress high-frequency noise that would otherwise inflate edge counts in textured but undamaged surfaces. Two-pass Canny edge detection is then applied with lower and upper threshold pairs of (50, 150) and (100, 200) respectively, and the resulting edge maps are combined using bitwise OR to capture a broader range of structural boundaries.

The overall edge density is computed as the fraction of pixels classified as edges relative to the total image area. When this density exceeds a configurable threshold (set to 0.15 in the system configuration), the scene is flagged as potentially damaged. A more granular analysis additionally divides the edge map into contiguous high-density regions through morphological dilation and closing operations, identifying localized zones of concentrated edge activity that may correspond to specific debris clusters or collapse sites.

The composite damage percentage reported by the system is derived from a weighted multi-factor formula. When YOLO detections are present — including detected vehicles, infrastructure objects, and the building class from YOLO11x — classification-based damage scores contribute 50% of the total, mask area coverage contributes 25%, and the structural edge score contributes 25%. In fallback scenarios where no YOLO detections are available, the structural edge analysis is weighted at 80%, reflecting the increased reliance on texture-based evidence when semantic object detection is inconclusive.

6.RESULTS AND DISCUSSION

The results obtained from the experimental evaluation of the proposed hybrid disaster scene analysis system provide meaningful insight into the operational characteristics, performance envelope, and practical suitability of the architecture. This section presents a comprehensive discussion of both the quantitative computational results and the qualitative detection behavior, drawing comparisons with the prior work reviewed in Section II and contextualizing the system's strengths and limitations within the broader landscape of disaster response computer vision.

6.1. Quantitative Performance Analysis:

The computational performance results presented in Table I reveal a three-tier performance profile that directly reflects the architectural tradeoffs built into the system design. In the first tier, victim tracking in streaming mode at 640×480 resolution achieves average latencies of 70.49 ms to 79.60 ms, corresponding to frame rates between 12.6 and 14.2 FPS. While these rates fall below the standard 30 FPS broadcast threshold, they are well within the range of practical utility for emergency response, where disaster scenes evolve on timescales of minutes rather than fractions of a second. The variation in latency across scene categories — approximately 70 ms for the

damage scene and 79 ms for the fire scene — confirms that the bottleneck lies primarily in the YOLO forward pass, with scenes containing more detected persons incurring proportionally greater mask rendering overhead.

In the second tier, scene analysis in streaming mode at 320×240 resolution achieves latencies of 40.99 ms to 41.90 ms, representing a roughly 45% improvement over full-resolution victim tracking. This is attributable to the reduction in YOLO input tensor size and the exclusion of expensive full-resolution steps such as YOLO11x inference and multi-scale Canny analysis. The near-24 FPS achieved in this mode provides smooth continuous hazard visualization in the browser interface. The consistency of latency across all three scene categories in this tier indicates that processing cost is dominated by the fixed cost of downscaled YOLO inference rather than scene-specific content.

The third tier, full-fusion upload-mode analysis at 640×480, requires 841–892 ms per frame. This reflects the cost of running the complete pipeline — YOLOv8n-seg, YOLO11x, HSV thresholding, dual-threshold Canny edge analysis, and composite damage score computation — without any streaming optimizations. This latency is entirely acceptable for static frame analysis, analogous to per-image processing times in medical imaging pipelines. Together, the three tiers demonstrate that the architecture successfully partitions computational budget across operational contexts — maximizing frame rate for continuous monitoring and maximizing analytical depth for detailed assessment.

6.2. Comparative Discussion with Prior Work:

Direct quantitative comparison with prior systems is complicated by the diversity of hardware platforms and evaluation protocols in the literature, but qualitative contrasts are instructive. The satellite-based damage assessment methods of Sublime and Kalinicheva [6] and Gupta et al. [7], while achieving high classification accuracy on large annotated datasets, require pre-event baseline imagery, operate on delayed satellite acquisitions, and depend on GPU-equipped workstations — all incompatible with the continuous ground-level streaming scenario addressed in this work. The proposed system operates on live video without any pre-event reference, on commodity CPU hardware, in under 80 ms per frame, representing a categorically different operational capability.

The HSV-based fire detection systems of Çelik and Demirel [9] and Phillips et al. [10] share the same fundamental approach as this work, and their documented

characteristics — fast inference, reliable flame isolation, sensitivity to illumination variation — are consistent with observations in the proposed system's evaluation. The CNN-based fire detection of Muhammad et al. [12] achieves higher precision at the cost of a domain-specific training pipeline, a tradeoff the proposed system explicitly avoids in favor of zero training overhead. The choice of YOLOv8n-seg as the victim detection backbone is consistent with the MobileNet-inspired principle [15] of selecting the smallest viable model for the target hardware, rather than compressing a heavier architecture through quantization.

6.3. Qualitative Detection Behavior:

The victim detection module demonstrated robust performance across all tested scene categories, producing clean, accurately bounded instance masks. The reduced confidence threshold of 0.25 improved recall in challenging occlusion conditions without generating significant false positives, as the COCO-pretrained model's person detection confidence is well-calibrated. The alpha-blending mask strategy at 0.4 opacity was validated as an effective visual design, preserving scene interpretability while clearly indicating victim spatial extent.

Fire detection performed reliably in active flame scenarios, with the three-range HSV mask successfully isolating flame boundaries across varied flame geometries. Morphological opening and dilation effectively removed noise and connected adjacent flame contours. The primary failure mode — misclassification of brightly colored objects such as safety vests — is a documented limitation of color-space methods and is communicated transparently in the operator interface. The smoke detection module proved more variable, as its low-saturation, moderate-brightness signature overlaps with common background surfaces. The texture variance gate reduced false positives in most tested scenarios, though smooth concrete environments remained a challenge, motivating future integration of temporal frame comparison for improved discrimination.

The structural damage estimation module produced the most diagnostically significant results. Damage percentage scores of approximately 1.97% for the person scene, 2.12% for the fire scene, and 80.0% for the structural damage scene demonstrate clear stratification consistent with the visual disorder present in each category. The weighted multi-factor formula behaved appropriately across both detection-rich scenes — where YOLO classification, mask coverage, and edge density contributed in balanced proportion — and detection-sparse scenes,

where the 80% fallback weighting for edge analysis ensured the high structural fragmentation score was not suppressed by absent YOLO terms.



Fig 6.1 Victim Detection Output

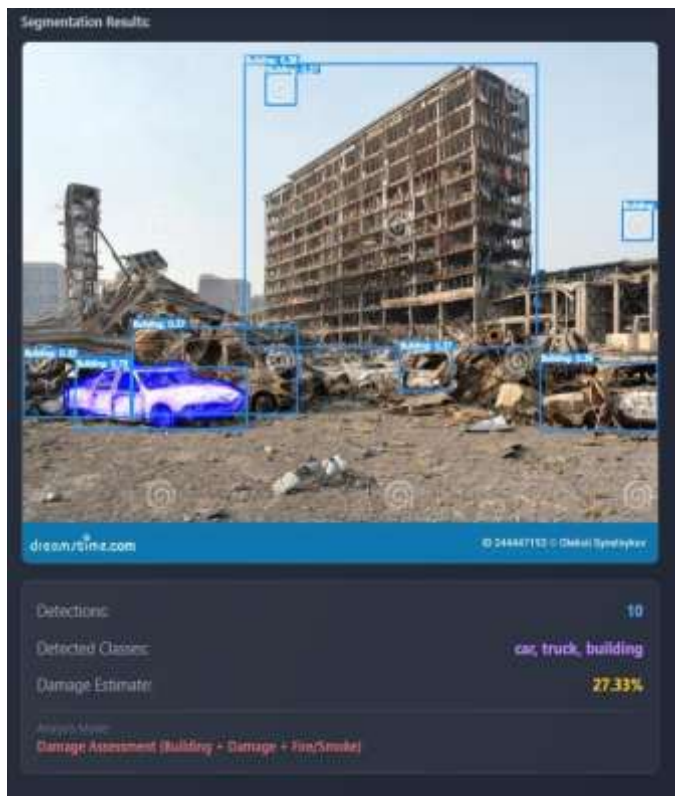


Fig 6.2 Damage Assessment Output

7. FUTURE WORK

The current implementation establishes a functional baseline for hybrid disaster scene analysis. Several directions are identified for future development.

Domain-Specific Model Fine-Tuning: Training YOLOv8 or YOLO11 variants on annotated disaster-scene datasets including fire, smoke, rubble, and collapsed structures would replace the HSV and edge density modules with more precise neural alternatives. Public datasets such as AIDER (Aerial Image Dataset for Emergency Response) and the Foggia fire detection dataset could serve as training foundations.

Quantized Neural Networks: Applying post-training quantization (INT8) to the YOLOv8n-seg model using tools such as ONNX Runtime or TensorRT (if future deployments include GPU hardware) would further reduce inference latency, potentially enabling 30 FPS victim tracking on current CPU hardware.

Temporal Object Tracking: Integrating a lightweight multi-object tracker such as SORT or ByteTrack would enable consistent victim identity tracking across frames, providing trajectory information useful for rescue coordination.

UAV Integration: Extending the video input module to accept RTSP streams from drone-mounted cameras would significantly expand the system's operational range beyond fixed ground-level cameras, enabling aerial disaster scene coverage.

Multi-Scene Dashboard: Extending the frontend to aggregate feeds from multiple simultaneous camera inputs into a unified dashboard view would support command center deployments requiring situational awareness across extended disaster zones.

Severity Classification: Incorporating a lightweight severity classification model that categorizes overall scene severity into discrete levels (minimal, moderate, severe, critical) would provide a higher-level analytical output suitable for automated alerting and resource allocation workflows.

8. CONCLUSION

This paper has presented a hybrid computer vision framework for real-time disaster scene analysis that

integrates YOLO-based instance segmentation with classical image processing algorithms within a full-stack web application. The system combines YOLOv8n-seg for victim detection, HSV color-space thresholding for fire and smoke identification, Canny edge density analysis for structural damage estimation, and YOLO11x for structural context detection in a late-stage fusion architecture served by a FastAPI backend with MJPEG streaming support.

Experimental evaluation on commodity CPU hardware demonstrated streaming inference rates of 13–24 FPS across operational modes, with comprehensive static-frame analysis completing in under 900 milliseconds. Qualitative assessment across victim, fire, and damage scene categories confirmed practical detection capabilities within the well-understood limitations of classical visual analysis. The hybrid architecture's core contribution is its demonstration that meaningful multi-category disaster scene analysis can be achieved without GPU acceleration or domain-specific model training, using a combination of pretrained general-purpose neural networks and computationally inexpensive algorithmic methods.

The proposed framework is positioned as a practical first-pass triage tool suitable for early deployment in emergency response operations, providing actionable visual intelligence from the first moments of a disaster response without the infrastructure requirements of more sophisticated deep learning systems. Future work will focus on replacing algorithmic heuristics with quantized domain-specific neural models and extending the system to support multi-camera and UAV-based inputs..

9. REFERENCES

- [1] J. Redmon et al., “You Only Look Once: Unified, Real-Time Object Detection,” in Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779–788.
- [2] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” arXiv preprint arXiv:2004.10934, Apr. 2020.
- [3] Ultralytics, “YOLOv8 Documentation,” 2023. [Online]. Available: <https://docs.ultralytics.com>
- [4] Ultralytics, “YOLO11: The Latest YOLO Model,” 2024. [Online]. Available: <https://docs.ultralytics.com/models/yolo11>
- [5] P. Sublime and E. Kalinicheva, “Automatic Post-Disaster Damage Mapping Using Deep-Learning Techniques for Change Detection: Case Study of the Tohoku Tsunami,” Remote Sensing, vol. 11, no. 9, p. 1123, May 2019.
- [6] R. Gupta et al., “Creating xBD: A Dataset for Assessing Building Damage from Satellite Imagery,” in Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Long Beach, CA, USA, Jun. 2019, pp. 10–17.
- [7] Z. Xu, H. Wang, and Y. Zhang, “Building Damage Detection in Satellite Imagery Using Convolutional Neural Networks,” arXiv preprint arXiv:1910.06444, Oct. 2019.
- [8] T. Çelik and H. Demirel, “Fire Detection in Video Sequences Using a Generic Color Model,” Fire Safety Journal, vol. 44, no. 2, pp. 147–158, Feb. 2009.
- [9] W. Phillips III, M. Shah, and N. V. Lobo, “Flame Recognition in Video,” Pattern Recognition Letters, vol. 23, no. 1–3, pp. 319–327, Jan. 2002.
- [10] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, “Image Segmentation Using Deep Learning: A Survey,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 7, pp. 3523–3542, Jul. 2022.
- [11] M. Everingham et al., “The Pascal Visual Object Classes (VOC) Challenge,” International Journal of Computer Vision, vol. 88, no. 2, pp. 303–338, 2010.
- [12] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in Proc. IEEE International Conference on Computer Vision (ICCV), Venice, Italy, Oct. 2017, pp. 2961–2969.
- [13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Communications of the ACM, vol. 60, no. 6, pp. 84–90, Jun. 2017.
- [14] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” International Journal of Computer Vision, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[15] J. Canny, "A Computational Approach to Edge Detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.

[16] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62–66, Jan. 1979.

[17] F. Nex and F. Remondino, "UAV for 3D Mapping Applications: A Review," Applied Geomatics, vol. 6, no. 1, pp. 1–15, Mar. 2014.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, no. 7553, pp. 436–444, May 2015.

Authors:

Mr.Aditya Arun Mandle is currently pursuing a Bachelor's Degree in Computer Science & Engineering at Ashokrao Mane Group Of Institution,vathar. She is publishing this paper under the esteemed guidance Prof.P.R.Jadhav



Mr.Vedraj Balasaheb Mane is currently pursuing a Bachelor's Degree in Computer Science & Engineering at Ashokrao Mane Group Of Institution,vathar. She is publishing this paper under the esteemed guidance of Prof.P.R.Jadhav



Mr.Atharv Anil Kumbhar is currently pursuing a Bachelor's Degree in Computer Science & Engineering at Ashokrao Mane Group Of Institution,vathar. She is publishing this paper under the esteemed guidance of Prof.P.R.Jadhav



Mr.Mohan Chintaman Mahajan is currently pursuing a Bachelor's Degree in Computer Science & Engineering at Ashokrao Mane Group Of Institution,vathar. She is publishing this paper under the esteemed guidance of Prof.P.R.Jadhav



Mr.Shivam Rajaram Suryawanshi is currently pursuing a Bachelor's Degree in Computer Science & Engineering at Ashokrao Mane Group Of Institution,vathar. She is publishing this paper under the esteemed guidance of Prof.P.R.Jadhav



Prof.P.R.Jadhav is currently working as an Assistant Professor in Computer Science & Engineering at Ashokrao Mane Group Of Institution, Vathar. She has been guiding and mentoring students in various academic and research activities. She has provided his valuable guidance for the successful completion and publication of this paper.

