Fire Image Classification with a Modified YOLOv11 Backbone

Haoyu Wang Yifu Hou Haitian Wang
College of Computer Science and Artificial Intelligence, Fudan University
22307130168@m.fudan.edu.cn yfhou22@m.fudan.edu.cn 22307140113@m.fudan.edu.cn

Abstract

Fire recognition from images is important for early warning and safety monitoring. In this course project, we study a three-class fire image classification task on a Kaggle dataset with labels fire*,* start_fire*, and* no_fire*. We adapt a YOLOv11-style backbone, originally designed for detection, into an image-level classifier by redesigning its classification head and applying task-appropriate data augmentation. Our pipeline is implemented in PyTorch with a reproducible training procedure (AdamW optimization, checkpointing by minimum validation loss, and gradient clipping). The model achieves a training accuracy of 96.30% and a Kaggle public score of 0.907, while exhibiting a notable generalization gap on a small local validation set (72.57%). We report class-wise results and analyze why early-stage fire is harder and why local validation can be worse than the Kaggle test score under distribution and quality differences.*

1. Introduction

Fire detection is a practical computer vision problem with applications in surveillance, intelligent transportation, and emergency response. In many real-world scenes, fire-related evidence (flames, smoke, abnormal illumination) appears as *global* and *multi-scale* visual cues. Compared to tasks that emphasize precise boundaries, fire recognition often benefits from robust scene-level representation learning.

Task. We focus on an image classification setting derived from a Kaggle fire dataset. Each image is categorized into one of three classes: (i) **fire**: clearly visible flames; (ii) **start_fire**: early-stage fire with subtle or ambiguous evidence; (iii) **no_fire**: no fire evidence. The goal is to learn a model that predicts the correct class for an input image. This task definition follows our course project specification and presentation material.

Motivation. While standard CNN classifiers (e.g., ResNet) are natural baselines for image classification, we explore whether a YOLO-style architecture can serve as an effective backbone for fire classification. YOLO models are originally designed for object detection and emphasize efficient multi-scale feature extraction and context aggregation, which may better match fire recognition where large-area scene cues dominate over fine local details [1, 5].

Approach overview. We adopt a YOLOv11-style feature extractor and redesign its head for image-level classification. The final architecture consists of a convolutional backbone with C2f/C3k2/SPPF building blocks and a lightweight MLP-style classification head (Conv $\to$ GAP $\to$ FC layers with dropout). The training pipeline includes resizing to 640×640, geometric data augmentation, AdamW optimization, gradient clipping, and checkpointing based on validation loss.

Key findings. Our model fits the training set well and achieves a Kaggle public score of 0.907, but local validation accuracy is lower due to limited validation size, class ambiguity, and potential distribution differences. The `start_fire` class is consistently the hardest, matching the intuition that early fire signals are subtle and visually confusing.

Paper organization. Section 2 introduces the model, data processing, and training procedure. Section 3 reports quantitative results, comparison notes, and analyses (class-wise behavior, training dynamics, and error patterns). Section 4 concludes and discusses limitations.

2. Method

2.1. Problem Formulation

Given an RGB image $x \in \mathbb{R}^{H \times W \times 3}$, the task is to predict a label $y \in \{\texttt{fire}, \texttt{start\_fire}, \texttt{no\_fire}\}$. We train a multi-class classifier by minimizing cross-entropy:

$$\mathcal{L} = -\sum_{c=1}^{3} y_c \log p_c, \quad (1)$$

where p_c is the softmax probability of class c.

2.2. YOLOv11-style Backbone for Classification

Although YOLO architectures are originally designed for object detection, their backbones provide strong hierarchical representations with progressive downsampling and feature fusion. This aligns well with fire recognition, where cues are often scene-level patterns such as flames, smoke, and illumination changes rather than precise object boundaries [1, 5].

Backbone structure. Our backbone is implemented as a sequential stack of convolutional downsampling stages followed by C2f/C3k2/SPPF blocks. Specifically, the network progressively downsamples from 3 channels to 512 channels, then applies SPPF for multi-scale context aggregation, and finally produces a compact feature map for classification.

C2f blocks. C2f uses partial residual connections and multi-branch concatenation, reducing information loss in deep networks and keeping low-level texture cues accessible at later stages. This is useful for fire scenes where both textures (flame edges) and context (background illumination) contribute to decisions.

C3k2 blocks. C3k2 strengthens cross-channel interactions through multiple bottleneck layers and flexible kernel compositions, enabling higher-order structural pattern extraction beyond local edges. This can help distinguish true fire from visually similar non-fire lighting patterns.

SPPF module. The Spatial Pyramid Pooling–Fast (SPPF) module aggregates contextual information at multiple receptive fields by repeated max pooling on the same feature map. This design is closely related to the spatial pyramid pooling idea for multi-scale context aggregation [3], and it provides a strong global summary before classification, which is helpful because scene-level cues are important in fire recognition.

2.3. Classification Head

The original YOLO detection head is not used. Instead, we redesign a classification head: a 1×1 convolution for channel expansion, global average pooling (GAP) for translation invariance, followed by fully connected layers with SiLU activation and dropout, and a final linear layer for 3-class logits.

Classifier head (implemented).
1×1 Conv($512 \rightarrow 1024$) $\rightarrow$ GAP $\rightarrow$ Flatten
$\rightarrow$ FC(1024)+SiLU+Dropout(0.25)
$\rightarrow$ FC(512)+SiLU+Dropout(0.20)
$\rightarrow$ FC(3)

Figure 1. Schematic of the redesigned YOLOv11-style classification head.

Table 1. Data preprocessing and augmentation used in our implementation.

Stage	Operations
Train	Resize(640,640), HFlip, VFlip, Rotation(15°), ToTensor, Normalize
Val/Test	Resize(640,640), ToTensor, Normalize

2.4. Data Preprocessing and Augmentation

All images are resized to 640×640 to match the YOLOv11-style input setting. For training, we apply geometric augmentation: random horizontal flip, random vertical flip, and random rotation within $\pm 15^\circ$. For validation and inference, only resizing and normalization are used. Normalization uses ImageNet mean/std. We intentionally keep color augmentation conservative because the task is sensitive to global color and illumination.

2.5. Training Procedure and Reproducibility

We implement the training loop in PyTorch with the following design choices.

Optimizer and schedule. We use AdamW with learning rate 1×10^{-3}, weight decay 1×10^{-2}, batch size 32, and train for 25 epochs. A fixed random seed is set for reproducibility.

Checkpointing. We save checkpoints when validation loss decreases ("best-by-val-loss"). This prevents overfitting to training accuracy and provides a stable selection rule when validation accuracy fluctuates.

Stability tricks. We apply gradient clipping with maximum norm 1.0 to stabilize training. We also store loss/accuracy histories and optionally plot them after training for debugging and reporting.

Training algorithm (high-level). We summarize the implementation-level logic: (1) load train/val loaders; (2) iterate epochs; (3) per batch: forward, compute CE loss, backprop, clip gradients, AdamW step; (4) evaluate on validation; (5) save checkpoint if val loss improves; (6) log curves.

Table 2. Key training hyperparameters (from our training script).

Hyperparameter	Value
Epochs	25
Batch size	32
Optimizer	AdamW
Learning rate	0.001
Weight decay	0.01
Gradient clipping	$\|\nabla\| \leq 1.0$
Checkpoint rule	Save when val loss decreases
Input size	640×640

Table 3. Local split statistics (train/val).

Class	#Train	#Val
fire	1342	18
no_fire	970	126
start_fire	986	82
Total	3298	226

Table 4. Accuracy (%) on local split and train–val gap.

Class	Train Acc.	Val Acc.	Gap
fire	98.96	77.78	21.18
no_fire	99.38	76.98	22.40
start_fire	89.66	64.63	25.03
Overall	96.30	72.57	23.73

3. Experiments

3.1. Dataset and Splits

We use a Kaggle fire dataset with three categories: `fire`, `start_fire`, and `no_fire`. For local evaluation, we construct a train/validation split with 3298 training images and 226 validation images. The validation set is small and class-imbalanced, which can lead to high-variance estimates. Table 3 lists the split statistics.

3.2. Evaluation Metrics

We report classification accuracy on the local train and validation sets. In addition, we report the Kaggle public score on the held-out test set as provided by the platform.

3.3. Main Results

Table 4 summarizes class-wise accuracy on train and validation. Our model achieves 96.30% training accuracy and 72.57% validation accuracy, indicating a noticeable train–validation gap. On Kaggle, the public score reaches 0.907.

3.4. Comparison Notes

We also include a ResNet-18 classifier as a reference baseline for this three-class classification task [4]. In addition, prior work has studied fire detection in surveillance scenarios using both classical cues and learning-based methods [2, 6]. Since fire recognition is highly scene-dependent, we note that large-scale scene context is often important for robust classification [7].

Table 5. Reported Kaggle public scores for different backbones.

Model	Kaggle public score
ResNet-18 (baseline)	0.833
YOLOv8 (variant)	0.863
YOLOv13 (variant)	0.819
YOLOv11 (ours)	0.907

3.5. Training Dynamics

Training accuracy increases rapidly and exceeds 95% within a few epochs, reflecting the strong fitting ability of the backbone. Validation accuracy, however, fluctuates across epochs due to the small validation set and the presence of hard boundary samples. We therefore select the best checkpoint by minimum validation loss rather than peak validation accuracy, which provides a more stable criterion under high variance.

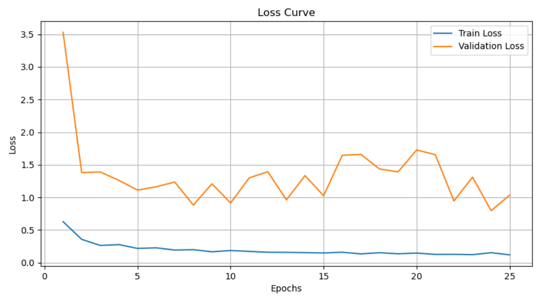
The curves in Figure 2 show the training process behavior. The training loss consistently decreases and the training accuracy rapidly increases within the early epochs, indicating fast convergence. Meanwhile, validation curves fluctuate more due to the small size of the validation set and the presence of ambiguous samples, highlighting challenges with generalization under data limitations.

3.6. Per-class and Error Analysis

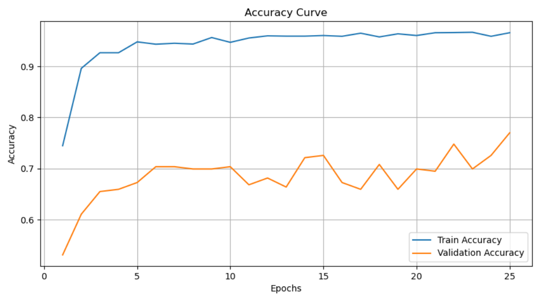
To better understand model behavior, we evaluate per-class accuracy and inspect error patterns. The `start_fire` class is consistently the hardest and shows the largest generalization gap. This aligns with the nature of the category: early-stage fire signals are subtle, visually ambiguous, and more sensitive to labeling noise.

Typical failure modes. Most errors occur when `start_fire` images resemble non-fire scenes (warm indoor lighting, reflections, haze) or when weak smoke-like textures appear without fire. In contrast, confusion between clear `fire` and `no_fire` samples is relatively rare, suggesting that the model learns strong discriminative patterns for obvious cases.

Why validation can be worse than Kaggle. A notable observation is that local validation accuracy (72.57%) is lower than the Kaggle public score (0.907). We attribute this to: (1) the validation set is small and imbalanced; (2)



(a) Training and validation loss across epochs.



(b) Training and validation accuracy across epochs.

Figure 2. Training dynamics showing loss and accuracy trends on both training and validation splits.

validation images may be blurrier or contain more boundary cases; and (3) the Kaggle test distribution may be closer to the training distribution, yielding a higher score.

4. Discussion and Limitations

Our results highlight both the strengths and limitations of YOLO-style classification. The backbone's strong multi-scale representation enables high performance on clear cases and shows good transfer to the Kaggle test set. However, early-stage fire recognition remains challenging due to intrinsic ambiguity. Additionally, dataset quality issues (low resolution, noisy labels) and limited validation size restrict the reliability of local evaluation.

Data augmentation is also constrained: fire recognition is sensitive to global color and spatial context, so aggressive color jitter or heavy geometric transforms may distort the semantics. Future work may include stronger validation protocols (e.g., cross-validation), label cleaning, and uncertainty-aware learning to better handle borderline `start_fire` samples.

5. Conclusion

We presented a YOLOv11-style approach for three-class fire image classification by redesigning the image-level classification head and adopting a reproducible training pipeline. The model achieves a Kaggle public score of 0.907 and strong training accuracy, while revealing a generalization gap on a small local validation set. Our analysis suggests that the main remaining challenge lies in early-stage fire ambiguity and dataset limitations, rather than backbone capacity.

References

[1] Yolov3: An incremental improvement. 1, 2

[2] Pasquale Foggia, Alessia Saggese, and Mario Vento. Real-time fire detection for video-surveillance applications using a combination of experts based on color, shape, and motion. *IEEE Transactions on Circuits and Systems for Video Technology*, 2015. 3

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision (ECCV)*, 2014. 2

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640*, 2016. 1, 2

[6] Qiang Zhang, Jia Xu, and Li Zhang. Fire detection using convolutional neural networks. *IEEE Access*, 2018. 3

[7] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3