AI Fire Detection

Duncan Grimes, Ricky Osgood, Justin Cheon

Researching real-time lightweight fire detection software compatible with a doorbell camera for home fire alerts

Introduction

Background

Wildfires have destroyed over 16,000 structures in California throughout 2025. More than 22% of homes in the USA have a doorbell camera. These smart home cameras will notify the homeowner of an intruder, but they send no alert if fire appears in the camera frame. This inspired our team to research lightweight fire-detection software compatible with a regular doorbell camera.

Doorbell Camera Footage during Palisades Fire, 2025

Objective

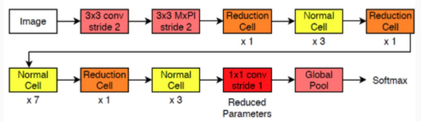
Create a real-time AI fire-detection and alert system that will notify users within 30 seconds of a fire appearing in the frame of their home security camera.

Methodology

Based on time and budget constraints, we elected to use a pre-trained machine learning model to detect fire in an image. We found a binary image classifier and generated our own validation dataset of images, with and without fire, to verify the model's accuracy on a real camera. We then plotted ROC curves with our validation data and experimented with different thresholds and preprocessing techniques to optimize our model.

Pre-Existing Research

Fast inference was a high priority for our team, so we sought out high-accuracy fire detection models with minimal complexity. We came across a model called ShuffleNetV2-OnFire, trained with the lightweight ShuffleNet architecture (pictured below) boasting 95% accuracy for full-frame binary fire classification with public access under the MIT license.



ShuffleNetV2-OnFire architecture

Researchers from Durham University in the UK used over 26,000 images in the training and validation process to create this model. The model was implemented in PyTorch and trained for 40 epochs using Stochastic Gradient Descent with a learning rate of 0.0005. They also utilized transfer learning from ImageNet-pretrained models, initially freezing most layers and then unfreezing specific final layers for further tuning on the fire detection dataset

Methods

Data Collection

To determine the accuracy of the ShuffleNetV2-OnFire model, we created a validation set of fire and non-fire images. We collected these images on a webcam in a range of natural lighting conditions, from daytime to nighttime. Fire was introduced by igniting cardboard in a controlled setting outdoors, simulating realistic visual characteristics of small-scale open flames.

Fire in daylight

No fire in daylight

Fire at night

No fire at night

We conducted multiple trials in which we started collecting images on the webcam 5 seconds before lighting the fire, capturing a frame every 2 seconds until the fire burned out. We varied the camera angle as well as the fire's location and size to increase data diversity. This approach allowed us to isolate fire-related visual features while capturing varied perspectives and lighting environments.

After we finished gathering this data, we manually labeled the images as 'fire' or 'no fire' to give us ground truth for our experiments. The counts are shown below.

Light Conditions	Fire	No fire	Total
Daytime	87	32	119
Nighttime	77	6	83
Total	164	38	202

Experiments

We tested the ShuffleNetV2-OnFire model using the daytime, nighttime, and aggregate data we collected. We began using the default threshold of 0.5 for binary classification.

We later experimented with various pre-processing techniques to improve the model's accuracy, including adjusting color ratios, tweaking brightness and contrast, and rotating the frames. Additionally, we explored architectural changes by replacing the model's final global average pooling layer with a global max pooling layer. We also tested multiple threshold values to fine-tune the sensitivity of the fire detection output, both during the day and at night.

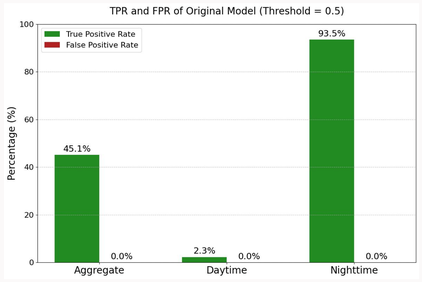
Results

Key Metrics

True Positive Rate (TPR): of all images that contain a fire, what percentage did the model predict as fire?
False Positive Rate (FPR): of all images that did not contain a fire, what percentage of images did the model predict as fire?

Analysis of Original Model

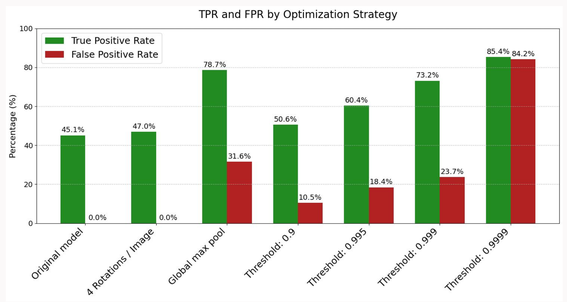
Here we look at the TPR and FPR of our model with the default threshold of 0.5, examining the daytime, nighttime, and aggregate datasets.



Across all data segments, our FPR was 0%, meaning we had no issues with false positives, regardless of light conditions. At a threshold of 0.5, the model was able to identify only 2.3% of the fires in the daytime images, meaning the model was practically useless during the day. However, it was able to correctly identify 93.5% of the images that contained fire at night, a result strong enough for production use.

Model Optimization

We then tried various optimization techniques, examining the model's TPR and FPR on the aggregate data (day and night).



One method we used was taking 4 rotations of an image (0°, 90°, 180°, 270°) and seeing if the model predicted fire in any of these. This slightly improved the TPR without increasing the FPR. However, this makes inference 4x as expensive per image, meaning it is not practical for production.

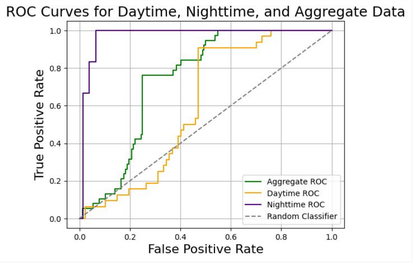
We also tried changing the final global average pooling layer to a global max pool. While this greatly improved TPR to 78.7%, it raised the FPR to an unacceptable level of 31.6%

Finally, we experimented with increasing the threshold for binary classification. As we increased the threshold from 0.9 to 0.9999, we saw a steady improvement in TPR, peaking at 85.4%, but this came at the cost of raising the FPR to an impractical 84.2%.

Results (cont.)

ROC Curves

We further explored threshold tuning by identifying different optimal thresholds for the distinct light conditions. We plotted Receiver Operating Characteristic (ROC) curves using our daytime, nighttime, and aggregate data segments. The model's overall performance at a given threshold is measured by Area Under the Curve (AUC), with 1.0 being perfect and 0.5 being truly random.



Data Type	Optimal Threshold	TPR	FPR	AUC
Aggregate	0.999	76.3%	25.0%	0.743
Daytime	0.999	90.6%	47.1%	0.604
Nighttime	0.608	100.0%	6.5%	0.974

ROC Analysis

The ROC curves confirm that the model performs well at night (AUC = 0.974), with perfect TPR and low FPR, but struggles during the day (AUC = 0.604). The aggregate AUC of 0.743 reflects this imbalance, reaffirming that the model lacks robustness across lighting conditions. For our daytime data, a high threshold of 0.999 increased the TPR from 2.3% to 90.6%. While significantly raising the threshold improves TPR, this leads to unacceptable increases in FPR (from 0% to 47.1%), making this strategy impractical for real-world use.

Discussion and Conclusions

Using the weights from ShuffleNetV2-OnFire, we wrote a Python script that can identify fire in real-time using any webcam attached to the user's laptop. We used the threshold 0.5, as any higher threshold results in false positives. The only input required is the user's email address. When the script is run, the webcam is launched by OpenCV and captures a frame every 2 seconds. The classifier then predicts if the frame contains a fire. If the model predicts a fire, an alert is emailed to the user from the Resend API.

In its current state, if we were to pitch our project to a home security company, we would recommend activating the software 30 minutes after sunset and deactivating it 30 minutes before sunrise. In a dark setting this software will identify a fire in the first frame in which it appears 93.5% of the time, with no false positives. On average, the user receives the alert in under 10 seconds from the fire appearing on camera. To truly improve the model's accuracy for daytime fires, we would need a robust dataset containing upwards of 10,000 fires in daylight, which we could use for retraining.

References

1. Fire.ca.gov, "2025 Fire Incident Reports."
2. AARP, "Guide to Video Doorbells," 2024.
3. Thompson, "Efficient and Compact CNN Architectures for Real-time Fire Detection," ICMLA, IEEE, 2020.