EmberVision: Visual early Fire-detection sensor with Machine Learning on Microcontrollers

Ari L. Stehney

Submitted to Ideaventions Academy for Mathematics and Science

Date: 5/26/25

Abstract

Traditional smoke detectors can delay fire detection in many scenarios when smoke must travel long distances, leading to damage to belongings and more lost lives as flames quickly become harder to control. This research presents EmberVision, a low-cost vision-based fire and smoke sensor that uses machine learning on microcontrollers to perform detection with minimal power consumption, no security concerns, and at a low cost. A unified fire and smoke dataset was created from 3 sources with 17,000 images, which was then used to train and optimize a scaled-down MobileNetV2-based binary classification CNN. The model achieved 94.4% accuracy pre-quantization, and after fine tuning with quantization aware training (QAT) and applying dynamic range quantization it maintained 90.5% accuracy with an 82% reduction of size to 616KB. A custom PCB module was then designed to run the model, consuming only 65mA during operation and with easy connectors to integrate it into a larger device. This module demonstrates the feasibility of an inexpensive visual early fire detection solution for the home that was traditionally unavailable but also acts as an blueprint of a way to deploy distributed machine learning on low-power devices to bring machine learning back from the cloud.

Introduction

In 2023 alone, Fire damage caused 10,190 civilian injuries (www.nfpa.org) and $11 billion in property damage. Having an early warning of flames or smoke in the critical moments before a traditional detector can trigger, especially if no people are present, can prevent damage and the spread of fire – saving property, lives, and first responders.

This project aims to create a miniature visual fire sensor module that will use a CNN with TensorFlow Lite running on an ESP32 microcontroller to recognize flames and smoke as soon as the fire starts, and notify peripherals connected to it. The project will start with curating a dataset and creating a larger scale model based on MobileNet and FireNET and then optimizing and simplifying the model architecture by removing layers to fit in the 1 MB size limit necessary to inference in the microcontroller memory. Compared to the few previous solutions that serve a similar function, the sensor will not require internet access, outside server utilization or subscriptions, or large amounts of power, and will cost a fraction of the price.

Methodology

The goal of this project is to create a visual fire and smoke sensor using machine learning principles on microcontrollers. While traditional smoke detectors are used everywhere, they can take time to trigger in scenarios when smoke must travel a longer distance to reach them, allowing the fire to spread further than it would if a person were present to acknowledge the flames and extinguish them.

This is where a visual approach has merit, in areas where flames are not expected from common household candles or fireplaces, a sensor can detect unexpected ignitions in an instant and set off an early alert before extra damage is caused. Implementations of this technique have been proven on security cameras and other systems; each requiring setup or money that normal individuals would not choose to spend. In addition, these systems either use more power than necessary for a home or offload the video processing to outside servers, creating security vulnerabilities or requiring subscription fees.

IEEE has concluded that "Visible spectrum video-based fire detection using non-stationary cameras has been an overlooked research problem." [11] By creating a simple sensor board with a camera with a microcontroller running a fire detection model locally that costs less than $10, this project will make early detection easily accessible, paving the way for companies and smart home devices to utilize it. In addition, this sensor is completely independent, having no reliance on cloud services, subscriptions, or internet connection. By bringing the inferencing back to the device itself, this sensor could inspire a paradigm shift to devices that don't require subscriptions and have a smaller carbon footprint because of their low power draw.

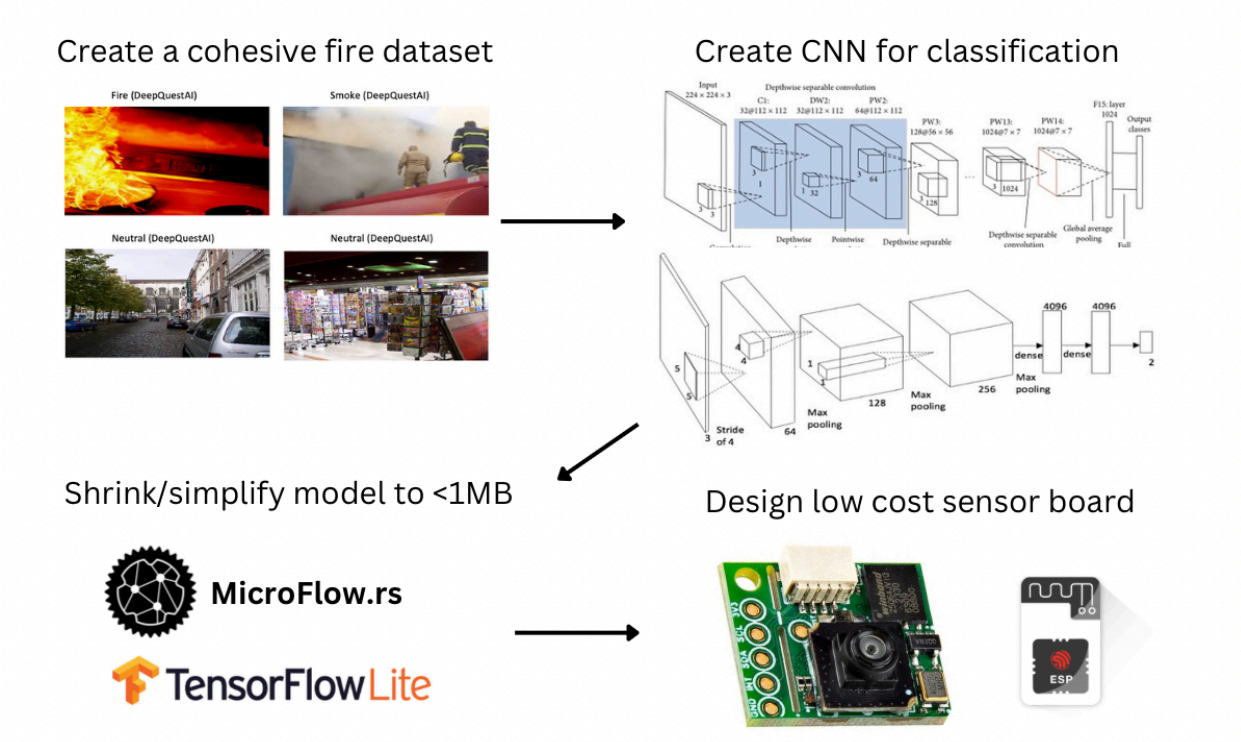


Fig. 1, The workflow for processing imagery and detecting objects.

Objective I. Create a unified Fire and Smoke image dataset

Multiple Fire and Smoke detection datasets already exist, but each are relatively small. First, a unified dataset was created that contained a single standard for labels and many diverse samples.

GAIA D-Fire [5]

Very large dataset of over 10,000 tagged and annotated images of Fire, Smoke, a combination, or neither. This is the primary source of data due to its well-organized file structure and large size.

DataCluster Labs Fire and Smoke Dataset [10]

A dataset composed of real images of fire that were captured on smartphones between 2020 and 2021. With over 7000 images in 400 different environments, this dataset has the breadth of images necessary to train a model for indoor and outdoor detection and was a secondary source.

DeepQuestAI Fire-Smoke-Dataset [3]

A simple dataset, with 1000 color images each of Fire, smoke, and neutral environments. This dataset is commonly used with ResNet or CNNs to achieve over 85% accuracy. Data from this set was employed for validation and real-life testing as many samples are very high-resolution images.

Fig. 2, Source image samples

Class Distribution

To ensure training efficiency and guarantee that the model is not overly biased to one outcome, the dataset has been curated to have roughly a 50/50 distribution of classes. To make sure that there will be the largest amount of data available for testing, the model has not been slimmed down to achieve to those exact proportions, it is at an 87:100 ratio with a light bias towards images with fire. This could ultimately serve as a benefit as it is better for the sensor to overly sensitive than not sensitive enough, and the sensitivity of the model will ultimately be able to be tuned during training and in the final sensor by averaging over time.

The dataset has already been split, 80% for training, 20% for testing, and an additional validation dataset has been created with about 1000 images that will be used to test on the final board.

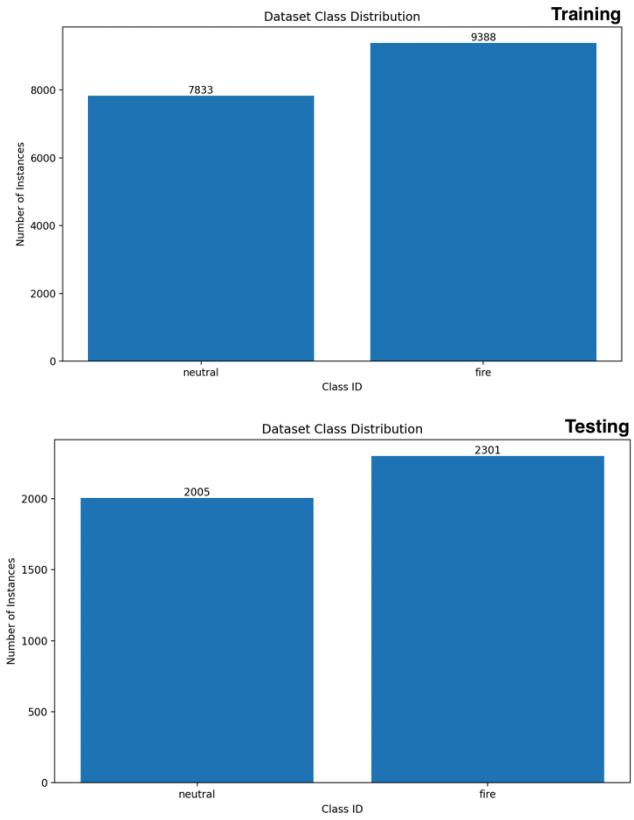


Fig. 3, class distribution of training and testing datasets

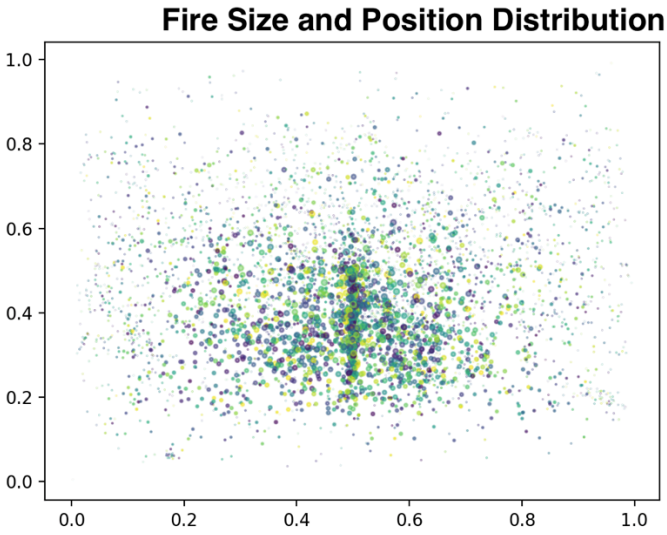


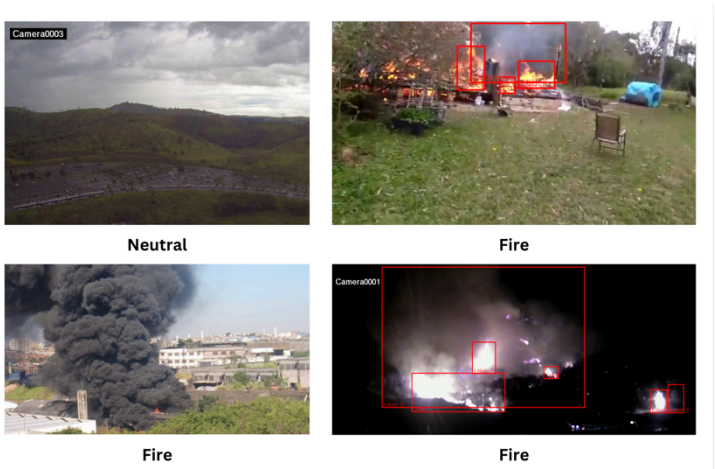
Fig. 4, distribution of bounding box positions, larger bubbles represent larger boxes

The dataset contains images with varying environments, both indoor and outdoor, each having varying lighting and weather. The images also have many different sizes of fire, or even multiple ignition sites throughout the image. The position of large flames tends to be in the center of the image, but many large fires are not fully enclosed by their bounding boxes, and cover most of the image. The CNN model architecture will have many convolution and pooling layers that will ultimately cause the model to learn general fire features that are location-independent in the image.

Data Labeling

For simplicity, the model will only perform binary classification and return one of 2 labels, “0” or neutral environment, and “1” or problem environment (fire or major smoke), for the entire image. Both the D-Fire and Data Cluster Labs datasets use COCO/Yolo labelling, meaning that instead of one label for the entire image, they have multiple bounding boxes with labels. In addition, both datasets include 3 effective classes, neutral (no boxes), smoke (boxes with label 0), or fire (boxes with label 1).

To remedy this, a unique function was created for each input dataset that ingests COCO and Yolo labels, and outputs the images into a basic folder structure based on their newly decided labels using a set of rules. If an image has any bounding boxes labelled fire, it will be labelled as 1 (problem environment). If an image has only bounding boxes labelled smoke, the area of the largest bounding box will be calculated, and the image will only be labelled as 1 (problem environment) if it covers 50% or greater of the image.



This is to prevent the model from being overly sensitive to things like clouds and fog in an image, and to ensure that the CNN will be primarily learning features from flames - which pose the most severe threat – without completely losing classification abilities for severe smoke plumes.

Objective II. Train a model to distinguish between scenes with fire or smoke, and neutral environments.

The following requirements of the model were set:

- >90% accuracy for binary classification of Fire and Neutral environments
- Below 1 MB final model size
- TensorFlow Lite Runtime compatibility

To accomplish this task, multiple compact footprint models from the EfficientNet and MobiletNet families were trained and compared. The EffecientNet architecture was intriguing due to its compound scaling methodology that reduces the overall computational load of inferencing based on the user's requirements. [16] The MobileNet family was also used due to its widespread connection with TinyML and TensorFlow Lite on the ESP32 and similar boards, and because of its use of depth wise separable convolutions and width and resolution scalers for decreasing the number of parameters. [13]

Each model was trained using transfer learning for the top layers of the model, with the lower convolutional layers retaining ImageNet weights. An initial set of training conditions were chosen for the models with a commonly chosen learning rate and activation function, both of which were tweaked over the course of testing. The camera used with most ESP32 boards can capture still images up to a 1600x1200 pixels, but 96x96 was chosen as the model input size since it was the largest feasible size that could be used if the model would possibly fit into allocated memory.

After training, the models were converted into TensorFlow Lite RT model format without quantization to reduce additional variables.

Objective III. Minimize the model to <1MB for running on a microcontroller

Once trained, the model will need to be converted to run on the microcontroller. Tensorflow provides solutions for exporting to the "tflite" (LiteRT) model format that can then be compiled or dynamically loaded by the microcontroller. For this project, the ESP32-S3 microcontroller has been chosen as a target because it is well supported and has a large community surrounding it, meaning that library support is very common. Nearly all CNN models, including all of those tested, are larger than the available memory on the ESP32, meaning that they will require quantization to be used at all for inferencing. Quantization is the process of truncating or rounding data in the weights or activations of neural networks in conjunction with scaling so that they can be stored as smaller data types, decreasing the memory and storage footprints of the model with a minimal amount of precision lost.

At first, after the model was trained, it would undergo a conversion to Tensorflow LiteRT model format which included various types of quantization optimizations. Any model produced with these techniques would produce results that had accuracies as low as 15% and were often heavily biased to one classification regardless of input. This result was extremely surprising and despite deep debugging efforts, the causes are still largely a mystery. One possible explanation is that, due to the addition of data augmentation and preprocessing layers to the model, the quantization was not accounting properly for the change in range of incoming data and causing larger than normal losses of precision. Along with that, some testing code contained two pre-processing passes, although once one was removed the accuracy only increased by a few percent.

To solve these issues, the training and quantization process was completely rewritten. During the quantization process, it is common to see a bias towards the majority class. To rule out that issue, the model was trained using class balance weights that were calculated based on the training dataset to account for the higher number of samples for fire situations. Next, the model was trained using a 2-stage approach with a set of normal fitting epochs and a second set of quantization-aware fitting (QAT) epochs. The QAT process [14] inserts dummy quantization operations into the forward pass that simulate the effect of quantization during the forward-pass of training, allowing the model to learn from the noise induced through rounding and clipping.

```
84  def apply_quantization_to_dense(layer):
85      if isinstance(layer, tf.keras.layers.Dense):
86          return tfmot.quantization.keras.quantize_annotate_layer(layer)
87      return layer
88
89  annotated_model = tf.keras.models.clone_model(
90      model,
91      clone_function=apply_quantization_to_dense,
92  )
93
94  q_aware_model = tfmot.quantization.keras.quantize_apply(annotated_model)
```

Fig. 5, QAT setup code

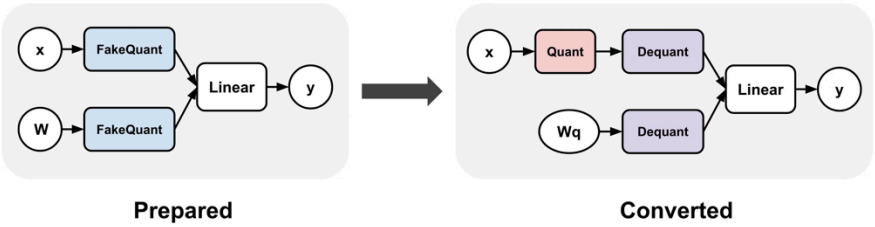


Fig. 6, QAT fitting, credit: PyTorch.org

This allows the model to update the weights using a gradient that retains full-precision FP32 data types. After the QAT model was tuned, it was then quantized normally using the experimental converter mode and a representative dataset with an equal split of samples between classes.

Inferencing

To run the model on the ESP32 microcontroller, a basic firmware was built using ESP-IDF and Tensorflow Lite, based off the existing person-detection example. This code was chosen as a base as it was built in similar constraints and performed binary classification using an older MobileNet architecture, leading to less new code having to be written.

After the model was quantized and converted to the flattened TFLite format, it was again converted into a C++ header file that held the model data as a byte array. The ESP32 contains 4MB of flash storage that can be used as IROM (instruction read-only memory, but only 512KB of IRAM (instruction random-access memory). The model data byte array is 8-bit aligned and stored as a constant, which causes the compiler to place it into the IROM in a way that ensures optimal 64-bit access. While this method allows the model to be stored, the TensorFlow lite interpreter still needs to allocate writable memory space in RAM for the inputs and activation tensors. The memory required for the inferencing "tensor arena" when using the fire detection model is larger than the internal RAM of the ESP32 S3, so the MMU virtual-addressing features must be used to allow the firmware to allocate this tensor arena in external PSRAM (pseudo-static random-access memory), which is 4-8 times larger than IRAM.

```
63  if (tensor_arena == NULL) {
64    tensor_arena = (uint8_t *) heap_caps_malloc(kTensorArenaSize, MALLOC_CAP_SPIRAM | MALLOC_CAP_8BIT);
65  }
```

Fig. 7, Tensor arena allocation code

Once the model inferencing memory has been allocated, the code creates an operation resolver that contains only the mathematical operations necessary to inference the model to save on memory space.

The firmware also must read from the camera and convert the pixel data types into the required range and precision to be used as model inputs. This and the rest of the firmware functionality is all handled through a variety of helper functions that are in the project GitHub.

Objective IV. Design a sensor module with a custom PCB that can be incorporated into other devices.

While the model provides the core detection functionality and can run on development boards easily, deploying it in a scalable way requires a cheap dedicated sensor module. To this end, we designed a minimalistic breakout board that can perform inferencing and be integrated into other products through a serial interface. Because of the choice to use the ESP32 microcontroller, a large selection of open-source development board schematics for it are available online, and the schematic released by Espressif for the ESP32-S3-EYE was used as a reference. After creating the schematic, the PCB design was made in EasyEDA Pro.

The board contains the ESP32-S3-MINI-1U-N4R2 module which packages the CPU, PSRAM, and flash required while saving on cost and reducing complexity. To aid in development of the model and debugging, the board has a USB-C connector for programming, a JTAG/serial debugging header, a 4-pin serial interface for setting detection thresholds and reading the detections, and 2 RGB status LEDs for status and visual diagnostics.

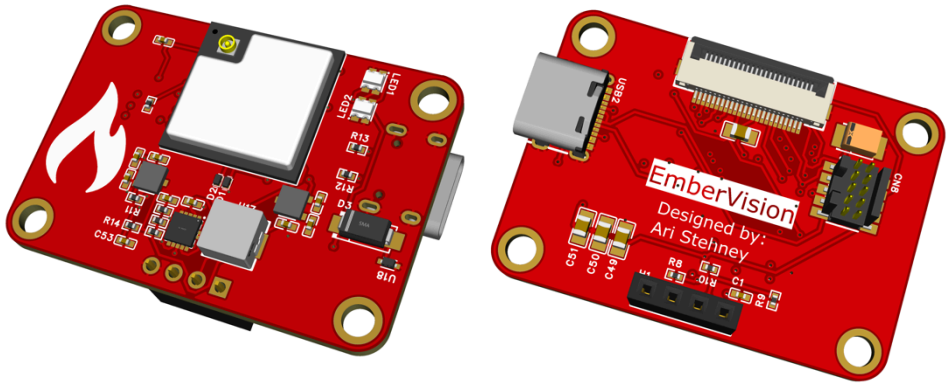


Fig. 8, EmberVision module 3D Renders

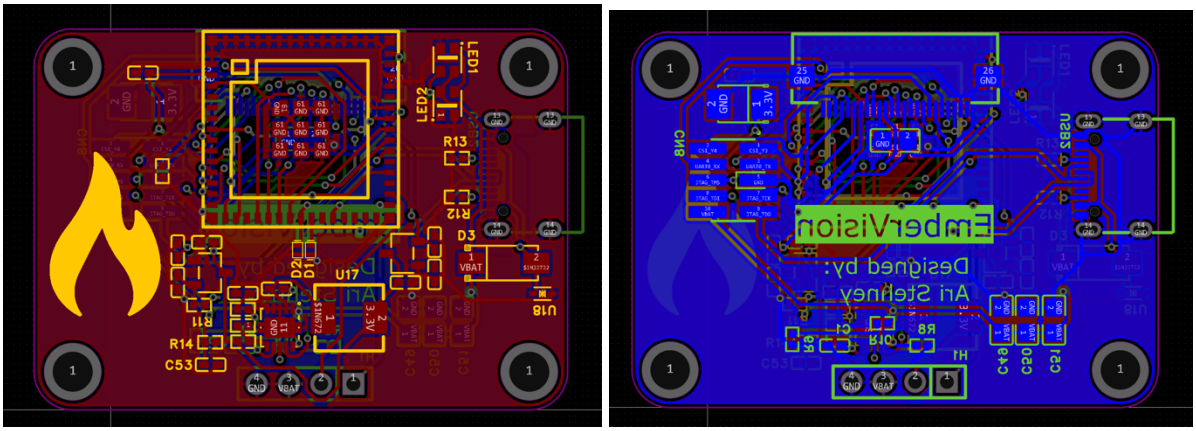


Fig. 9, EmberVision module PCB layout

The most intricate part of the PCB design is the power regulation circuitry, which takes the power inputs from either set of headers or the USB-C connector and accurately regulates them to

the 3 required voltages for the SoC and CSI camera (3.3v, 2.8v, 1.2v), while ensuring that no current is back fed into the USB host when the module is integrated into a product.

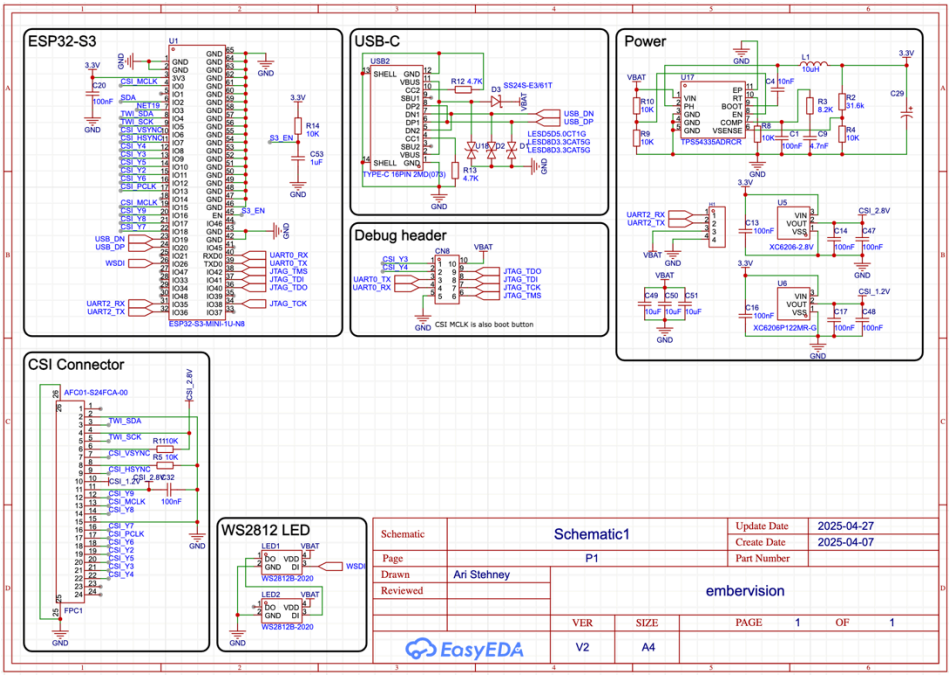


Fig. 10, EmberVision module schematic

For prototyping, the module was produced by JLCPCB and cost roughly $175 for 5 units fully assembled. This cost is heavily inflated because of the minimum order quantity of some components on the board, as well as the PCB production fees for low quantity orders. In addition, roughly half-way through the project, the US increased import tariffs from all countries including China to 145%, and then consequently decreased them back to 55%, drastically increasing the cost of production overhead of low-cost and low-quantity boards like this in general.

Item	Cost
5x 4-layer PCB with assembly	$172.65
Shipping	$41.27
Sales & use tax	$9.83
Import taxes	$86.63
Total	$310.68 ($62 per board)

Fig. 11, Prototype production pricing

In mass production, the cost of the board would likely drop to $10-$15, placing it far below the cost of any AI fire-detection security camera system, but above the cost of extremely inexpensive short-range infrared fire sensors. With this simple breakout board, it is easy for

companies to use the model in a consumer product and benefit from the low power consumption and increased complexity.

Results and Analysis

After training but before quantization, five-hundred unseen sample images were inferenced through each model, and the accuracies and calculated post-quantization memory footprints were compared.

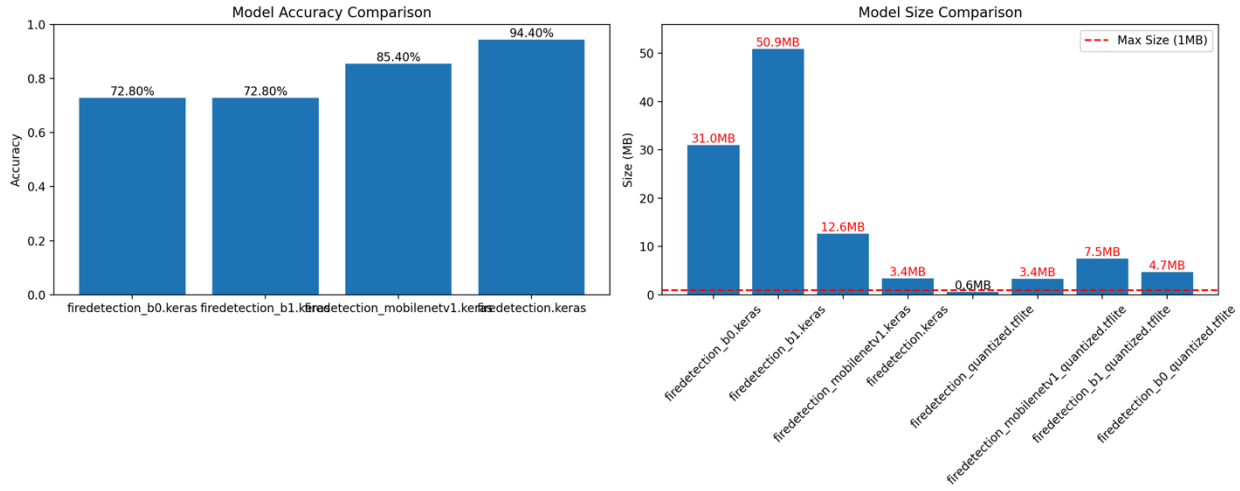


Fig. 12, Comparison of model accuracies and sizes

After testing we can see that the least complex EffecientNet models delivered low accuracy and inconsistent results during testing while being much larger than the maximum size threshold, even when quantized. It is important to note that this is likely due to issues with the transfer-training implementation that could be improved with more research instead of fundamental issues with the EffecientNet architecture, but it was eliminated from further testing due to its size. MobileNet v1 had acceptable accuracy results of 85.4% after 15 epochs of training where it reached a plateau. By changing the model parameters to $\rho = 1.00$ (resolution scaler) and $\alpha = 0.25$ (width scaler), the minimum recommended, it was possible to reach a quantized model size of 3.4MB which is still too large. The resolution scaler was left at 1.00 because it caused a large drop in accuracy during testing and did not cause a significant drop in model size.

MobileNet v2 builds on the architecture of previous MobileNets and replaces residual connections with inverted residual blocks and linear bottlenecks [15], leading to a drastically decreased parameter count. With a $\rho = 1.00$ (resolution scaler) and $\alpha = 0.35$ (width scaler), it was possible to achieve a pre-quantization accuracy of 94.4% on a model that is 616kb after quantization.

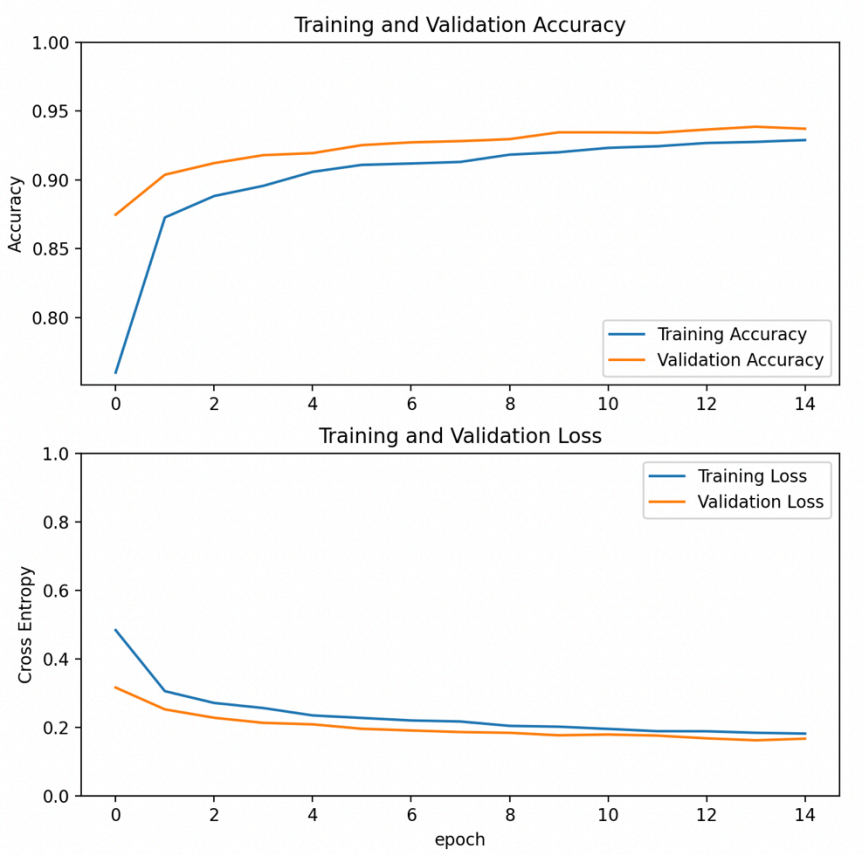


Fig. 13, Transfer learning training graphs for scaled MobileNet v2 model with final parameters

MobileNet v2 meets all requirements and performs well despite the low-resolution input, but during testing on real-world images a few common trends of incorrect classifications appeared. After analysis, they do not seem to create any major concerns about the real-world use of the model.

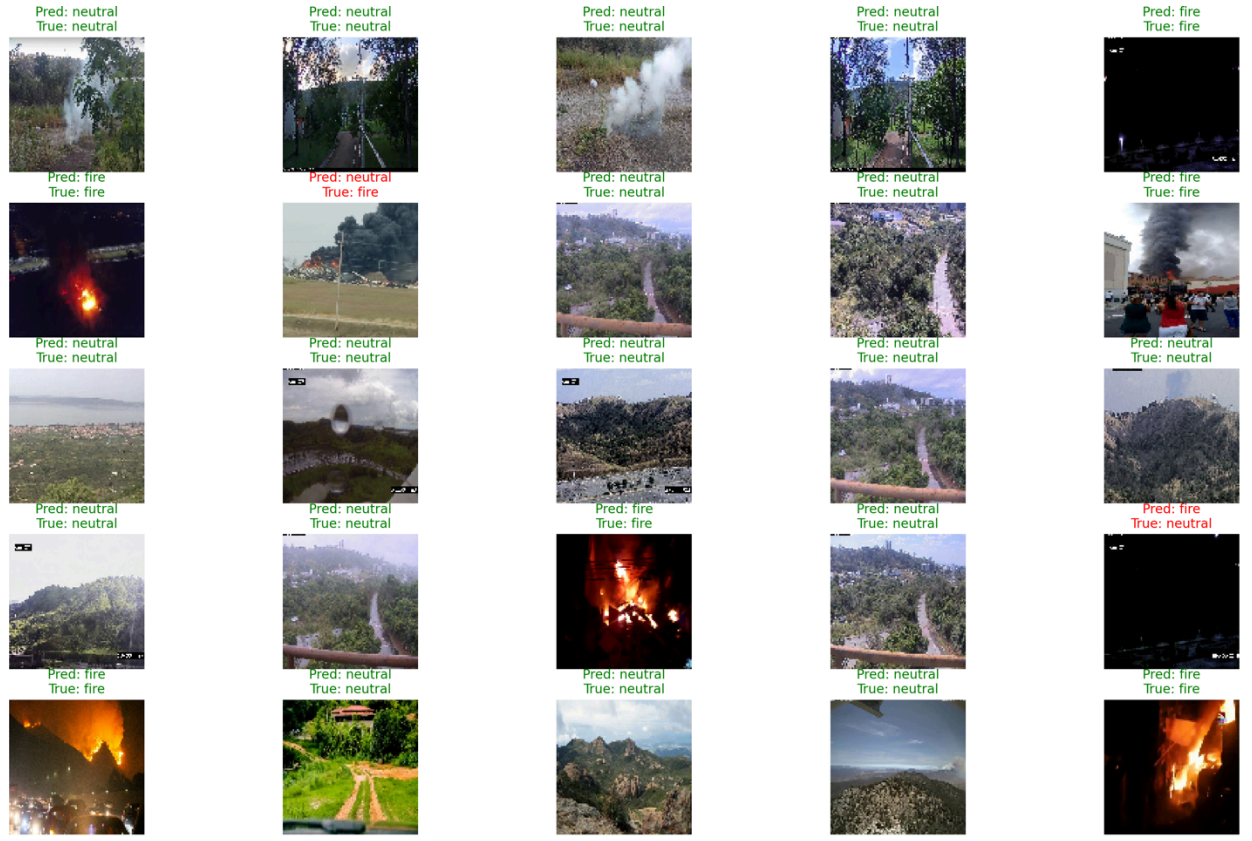


Fig. 14, Sample predictions on testing dataset images

Fig. 15, Common misclassifications

Most incorrect classifications are due the model having a lack of sensitivity to smoke, and oversensitivity to high contrast dark images with bright spots. These quirks mostly manifest themselves in smoke plumes going undetected, likely due to the choice to only include smoke images where the bounding box covered at least 50% of the image when preprocessing data. The benefits of the current methodology outweigh the minimal consequences. Images taken at night, particularly from cameras facing outdoors, tended to be classify as fire when there were visible streetlights or bright patches. This issue is expected and likely will not pose a real problem since this sensor is primarily for early warning and is only expected to be useful in lit areas.

Fig. 16, Samples of misclassifications due to dark images

After the scaled and modified MobileNet v2 model was selected, it was trained again using the Quantization Aware Training techniques documented above and quantized. Four versions with different types of quantization and with/without the Quantization Aware Fine-tuning were generated and their accuracies are shown below:

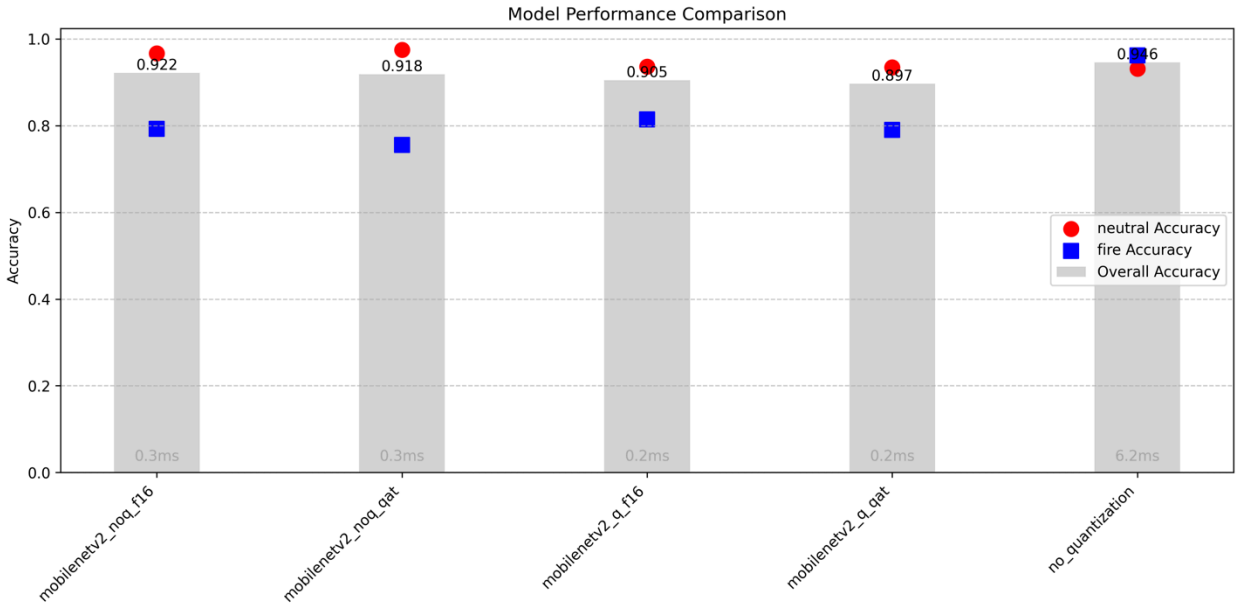


Fig. 17, Results of models with varying quantization

The quantization-aware model with dynamic range quantization (mixed FP16 and INT8, mobilenetv2_q_f16) was used as it both fits the size constraints and meets the requirements for accuracy. Dynamic range quantization likely performs better because, while it quantizes the

weights of the model to 8-bit integers, the activations are still calculated as floating-point values, reducing the accuracy loss while still optimizing the memory footprint dramatically.

Utilizing Quantization Aware Training with proper class-balance and dynamic range quantization, the optimized model only showed a 3.9% drop in accuracy despite a size decrease of 82%. When compared to larger scale fire detection models using YOLOv9 architecture [17], this model requires 95.6% less memory during inferencing.

While the 90.5% accuracy when using the final model is impressive considering the memory constraints, it is certainly possible to optimize the training process more and introduce additional layers to preprocess the input images so that the model is better able to differentiate between neutral and fire scenes. Scenes containing smoke and clouds are very challenging, and in the future, it may make sense to create a model that has 3 output classes (fire, smoke, and neutral) so that it is able to learn the unique features of smoke separately from fire. If research of this project were to continue, the most important change would likely be to shift the target from a microcontroller-based platform to a more powerful microprocessor-based platform not dissimilar from the Raspberry Pi. This would allow for a larger model with more sophisticated architectures like EffecientNet to be used, and if designed well would not require much more power or a larger board.

Hardware Results

Due to some minor production issues, the fire sensor boards produced were manufactured with a different version of the ESP32 module that did not contain any PSRAM. As discussed in the methodology section, this is key to loading and running models that are larger than the internal memory like the one created. The board functionality was tested and demoed with a simpler model that performed face detection that was a drop-in replacement for the fire detection model but is slightly smaller, and all features worked as expected. If tariffs and time constraints had not been as large of a factor, it would have been possible to produce a second round of prototype boards with this one issue fixed.

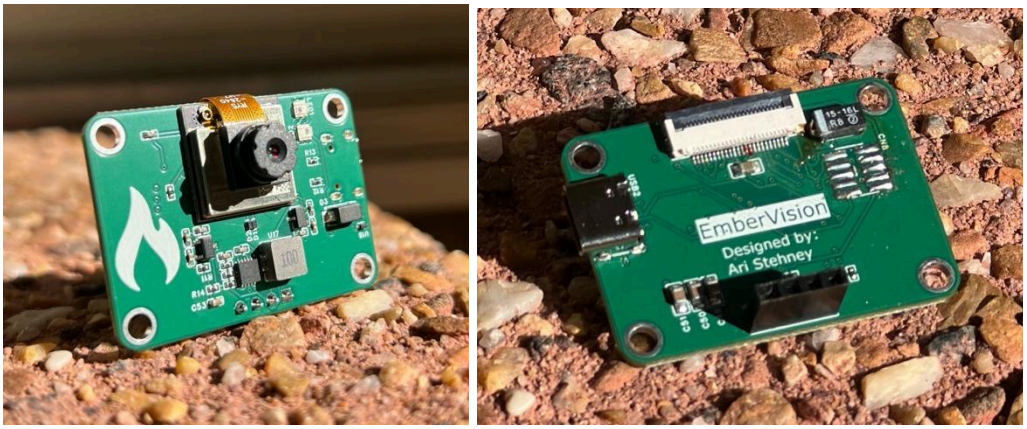


Fig. 18, The first prototype board

Power consumption was tested using a bench power-supply with the prototype boards and the substituted model. At idle it consumes roughly 10 mA (WiFi and Bluetooth disabled), when it is actively capturing and processing frames the power consumption peaks at about 65 mA. These figures are not perfectly accurate but demonstrate the energy efficiency possible with microcontrollers.

Instead of using the sensor boards, the model was tested using the exact same inferencing firmware on a ESP32-S3-EYE development board and worked as expected. This board perfectly simulates the expected performance of the EmberVision board as it uses an identical camera but also has the PSRAM that the first prototypes were lacking.

When testing with two printed images, one of a fire and one of a neutral environment, the predicted result consistently fluctuated immediately after the board experienced fast panning and tilting. This is likely due to the camera having poorly tuned contrast and potentially non-functional auto-white-balance and autofocus features, leaving it subject to extreme motion blur. This could be fixed in future firmware updates, but due to the limited time scope of the project it was not completed.

Conclusion

The goal of this project was to create a miniature visual fire sensor module and a lightweight CNN model to detect fire before it spreads and prevent damage that would have traditionally resulted with smoke detectors. Existing solutions range from large complex security camera systems with AI fire detection features that can cost thousands of dollars and require consistently high energy usage, to infrared fire detection sensors that only detect a small area. This project's niche is in that the sensor will not require internet access, outside server utilization, subscriptions or large amounts of power, and will cost a fraction of the price.

After training and minimizing a model to make it suitable to run within tight memory constraints that could still achieve 90% accuracy, we successfully developed a board that could run it that had a low cost and used minimal amounts of power. With the ease of use and low cost to integrate this module in products, it would be plausible to use this module in indoor home security and monitoring devices, public safety monitoring, and large swarms of sensors to watch the spread of wildfires or flames in buildings and provide early insights and warning to first responders.

This research shows the impact of effects like Quantization and QAT, as well as scaling techniques at decreasing the footprint of models in a way that allows them to still complete their desired tasks. This project also highlights the fact that older and simpler CNN architectures like MobileNet and EffecientNet still have a large place in computing, despite the rise of large and complex transformer-based models to combat exponentially increasing computational demands of our society.

For the smart home industry, this research paves the way for a new paradigm of distributed machine learning for interpretation of data. Most smart home devices released today include some functionality that relies on underlying machine learning models that are run remotely on

power-hungry servers, widening carbon footprints and introducing security vulnerabilities. With a model like this sensor, smart home devices could operate autonomously from outside services with improved security, low power use, and less operational costs in the form of subscriptions.

Along with these findings, there have also been some critical lessons learned. The biggest is that quantization strategy should be considered at all steps of the training process, not as an afterthought. It is an extremely sensitive step and can make or break any existing model, fully optimizing it almost always requires additional steps that may seem unrelated. Next, it is important to remember that dataset balance is key in all steps, not just training. An unbalanced dataset will hurt the model's accuracy, but it will also skew the accuracy numbers and make quantization much more difficult.

Future Work

The biggest question that is left throughout this entire process is how to handle the detection and classification of smoke. There are many solutions for indoor smoke detection already, but mastering a camera-based solution could help with widespread detection in large indoor rooms like airplane hangars or in outdoor industrial scenarios.

This model is accurate with large dark plumes of smoke, but in testing consistently missed lighter shades, and due to the low resolution, the camera will only perform worse the larger area it has to cover. In the future, it would be interesting to approach this project again with new goal to create a similar sensor using a higher resolution camera and a more powerful arm-based SoC, allowing for more sophisticated models using EffecientNet or newer architectures like VGG nets and huge input sizes. The processing power of the current board is less of a restriction than the small amount of memory.

This kind of hardware could also be applied to other scenarios that were not discussed. Companies like FLIR [18] are using thermal cameras to detect traffic incidents and monitor waste and recycling plants, but the hardware is expensive. Other companies like Kami and independent researchers from Tianjin University [19] are using security camera systems to perform fall detection for the elderly. This sensor or a future derivative could be used as an inexpensive platform to perform those tasks with similar benefits as it was intended to bring for with fire detection.

References

1. *(PDF) Innovating Fire Detection System Fire Using Artificial ...*, www.researchgate.net/publication/363658929_Innovating_Fire_Detection_System_Fire_using_Artificial_Intelligence_by_Image_Processing. Accessed 20 Dec. 2024.
2. David, Robert, and Jared Duke. "Tensorflow Lite Micro: Embedded Machine Learning on Tinyml Systems." *arXiv.Org*, 13 Mar. 2021, arxiv.org/abs/2010.08678.

3. DeepQuestAI. “DeepQuestAI/Fire-Smoke-Dataset: An Image Dataset for Training Fire and Frame Detection AI.” *GitHub*, github.com/DeepQuestAI/Fire-Smoke-Dataset. Accessed 20 Dec. 2024.
4. Edwios. “Edwios/Fire-Detection-CNN-Tflite: A Tensorflow Lite CNN to Detect Fire.” *GitHub*, github.com/edwios/fire-detection-cnn-tflite. Accessed 20 Dec. 2024.
5. Gaiasd. “GAIASD/Dfiredataset: D-Fire: An Image Data Set for Fire and Smoke Detection.” *GitHub*, github.com/gaiasd/DFireDataset. Accessed 20 Dec. 2024.
6. OlafenwaMoses. “Olafenwamoses/Firenet: A Deep Learning Model for Detecting Fire in Video and Camera Streams.” *GitHub*, github.com/OlafenwaMoses/FireNET. Accessed 20 Dec. 2024.
7. Robmarkcole. “Robmarkcole/Fire-Detection-from-Images: Detect Fire in Images Using Neural Nets.” *GitHub*, github.com/robmarkcole/fire-detection-from-images. Accessed 20 Dec. 2024.
8. Steffensbola. “Steffensbola/Furg-Fire-Dataset: A Dataset and Evaluation Scheme to Provide an Straightforward Manner to Compare the Performance of Different Non-Stationary Video Based Fire Detectors.” *GitHub*, github.com/steffensbola/furg-fire-dataset. Accessed 20 Dec. 2024.
9. Wahyono, et al. “Real-Time Forest Fire Detection Framework Based on Artificial Intelligence Using Color Probability Model and Motion Feature Analysis.” *MDPI*, Multidisciplinary Digital Publishing Institute, 12 Feb. 2022, www.mdpi.com/2571-6255/5/1/23.
10. Labs, DataCluster. “Fire and Smoke Dataset.” Kaggle, 22 Apr. 2023, www.kaggle.com/datasets/dataclusterlabs/fire-and-smoke-dataset.
11. “First Response Fire Combat: Deep Leaning Based Visible Fire Detection | Ieee Conference Publication | IEEE Xplore.” IEEE Xplore, ieeexplore.ieee.org/document/8215312/. Accessed 20 Dec. 2024.
12. Tensorflow. “Tensorflow/Tflite-Micro: Infrastructure to Enable Deployment of ML Models to Low-Power Resource-Constrained Embedded Targets (Including Microcontrollers and Digital Signal Processors).” GitHub, github.com/tensorflow/tflite-micro. Accessed 20 Dec. 2024.
13. Howard, Andrew G., et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.” *arXiv.Org*, 17 Apr. 2017, arxiv.org/abs/1704.04861.
14. Jacob, Benoit, et al. “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference.” *arXiv.Org*, 15 Dec. 2017, arxiv.org/abs/1712.05877.
15. Sandler, Mark, et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” *arXiv.Org*, 21 Mar. 2019, arxiv.org/abs/1801.04381.
16. Tan, Mingxing, and Quoc V. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.” *arXiv.Org*, 11 Sept. 2020, arxiv.org/abs/1905.11946.

17. spacewalk01. “Spacewalk01/Yolov5-Fire-Detection: Training Yolov5/YOLOV9 to Detect Fire in a Video.” *GitHub*, github.com/spacewalk01/yolov5-fire-detection. Accessed 26 May 2025.

18. “FLIR: Detect Fires Early, Avoiding Costly Damage and Loss.” *FLIR Fire Detection & Monitoring Systems | Teledyne FLIR | Teledyne FLIR*, www.flir.com/instruments/fire-prevention/. Accessed 26 May 2025.

19. Huang, Xiangqian, et al. “SDES-Yolo: A High-Precision and Lightweight Model for Fall Detection in Complex Environments.” *Nature News*, Nature Publishing Group, 15 Jan. 2025, www.nature.com/articles/s41598-025-86593-9.