# Automated Oatmeal Overflow Detection

Gautham Anant

## Abstract

*We present a vision system that detects imminent oatmeal boilovers in household microwave ovens and is lightweight enough to run in real-time on a CPU. We curate a 1744-frame RGB dataset of oatmeal-heating scenes, labeled frame-wise as off, safe, or unsafe. Fine-tuning MobileNetV3-Small with weighted sampling and focal loss on a heavily class imbalanced dataset achieves high recall ($> 90\%$) on all classes.*

## 1. Introduction

Microwaves are ubiquitous, yet their control interfaces have barely evolved. Users still guess power–time settings, often leading to messy boil-overs—especially for foods like oatmeal, soups, and milk that foam rapidly when super-heated. These spills waste food, create tedious clean-up, and in shared kitchens (e.g . offices or dorms) become hygiene hazards.

Microwaves prove to be a challenging environment for tasks due to low light, perforated-mesh, and steam. Vision offers non-contact, low-cost, accurate sensing and can theoretically generalize across foods and containers.

In this project we prototype a purely RGB early-warning system that can be installed in front of a microwave window. We optimize for a process that could be deployed without affecting internals of the microwave using regular devices. However, our dataset is very specific to a certain instance of the problem with a certain type of bowl and microwave.

**Contributions.**

- **Real-time early-warning system for microwave boil-overs.** A lightweight MobileNetV3-Small model runs at $> \mathbf{100}$ FPS on a M2 Macbook Air and achieves $> \mathbf{99}\%$ recall on the *unsafe* class.
- **Public 1.7 k-frame RGB dataset.** We release 33 annotated videos (1744 frames) of oatmeal heating, to possibly be incorporated into future datasets.
- **End-to-end low-cost hardware prototype.** An SG90-driven "button-clicker" arm, controlled via HTTP from the edge device with our model stops the microwave within 300 ms of an *unsafe* prediction.
- **Open-source code and reproducible pipeline.** Training, evaluation, and inference scripts are provided under an MIT license, along with detailed setup instructions for rapid replication.

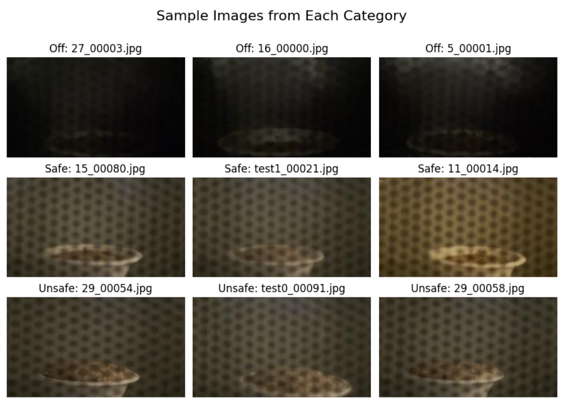

Figure 1. Example frames for each class.

## 2. Related Work

**Smart-appliance vision.** Khan *et al*. [2] retrofit a microwave with a thermal camera and CNN to recommend end-point temperatures, but their objective is *done-ness*, not overflow prevention. Commercial smart ovens focus on baking/roasting with bulky cameras and cloud inference. In addition, they use a complex in-microwave camera that requires reengineering the microwave itself.

**Efficient CNNs for edge devices.** MobileNetV3 [1] and TinyViT [7] balance accuracy and latency on 224×224 images, running at >20 FPS on Raspberry Pi 3 B+. We choose MobileNetV3-Small for its 5.5 M parameters and wide software support. Existing work on finetuning MobileNetV3 [5] has been done in the past.

Table 1. Frame counts by dataset and class.

| Split | off | safe | unsafe | total |
|---|---|---|---|---|
| Train | 64 | 666 | 142 | 872 |
| Val | 54 | 453 | 98 | 605 |
| Test | 10 | 213 | 44 | 267 |

## 3. Dataset

### 3.1. Collection Protocol

33 landscape videos of plain oatmeal heating were captured on an iPhone 12 (1080p resolution) through the mesh window of a 900 W consumer microwave. The videos were taken at the same position on the microwave door. Fluorescent lighting in the kitchen and an internal microwave lamp provided illumination. The same bowl of oatmeal was used for a few videos and was replaced with a new bowl of oatmeal when it was heated to an extent that it no longer had enough moisture to produce meaningful data.

The same microwave, oatmeal brand, and bowl brand were used across all videos, since the model is intended to work in the given kitchen.

Frames were sampled every 300 ms, down-scaled so the longer side equals 224 px. Labels were assigned by manual inspection into three categories: *off*, *safe*, and *unsafe*. *off* consists of images where the microwave being off, *safe* consists of images where the microwave is on, but intervention is not needed, *unsafe* consists of images where the microwave is on and requires intervention in the next couple frames else overflow will occur.

The training set was composed of 18 videos, the validation set was composed 10 videos, and the test set was composed of 5 videos. No bowl of oatmeal was shared across the sets in order to prevent fitting to the same heating run.

### 3.2. Class Imbalance

Since risky frames occur only for a short period before a video ends, they only account for 16% of frames, so naive training skews towards always predicting *safe*. Section 4 details our mitigation strategy. In addition, we are collecting more data to help evaluate and train the model.

## 4. Method

### 4.1. Pre-processing

The input frames are normalized with ImageNet mean and std. No augmentations outside of oversampling are used.

### 4.2. Network Architecture

We choose to fine-tune torchvision's [4] pre-trained MobileNetV3Small on our task. The final 1000-way fully-connected (FC) layer is replaced with a randomly initialized 3-way FC layer to classify between *off*, *safe*, and *unsafe*. We call our new layer the *head* and the rest of the model the *backbone*.

### 4.3. Training Objective

Our training data set sampled points proportional to the inverse of their label's representation in the dataset, with an addition 10x oversampling factor of *unsafe* images. We optimized focal loss [3] ($\alpha$=$[1.0, 1.0, 7.0]$, $\gamma$=2) with parameters that heavily penalize misclassifications of *unsafe* images. Not only did this help combat the effect of the large class imbalance due to lack of *unsafe* images, it also correctly penalized the model for misclassifying *unsafe* images because not stopping the microwave when it needs to be stopped is far more costly than stopping it accidentally.

We trained the head while keeping the backbone parameters constant with a learning rate of 3e-4 which decreases exponentially at 0.95 proportion every epoch using the AdamW optimizer ($\beta_1$=0.99, $\beta_2$=0.999) with a L2 regularization weight decay of 0.01. We stopped training when the validation accuracy on both *safe* and *unsafe* classes was greater than 75%. We checkpointed this model and refer to it as the *head only model*.

Then we trained the entire model (head and backbone) for 3 epochs at a learning rate of 1e-4 which decreases exponentially at 0.95 proportion every epoch using the AdamW optimizer ($\beta_1$=0.99, $\beta_2$=0.999) with a L2 regularization weight decay of 0.01. We stopped training when the validation accuracy on both *safe* and *unsafe* classes was greater than 90%.

### 4.4. Failed Approaches

The above training approach was found after a failure with other methods including

- **Vanilla OpenCV.** Using straightforward techniques for removing mesh from the microwave and finding circles in the picture failed.
- **Segmentation.** For YoloV8 [6], no segmentation masks were found on several images that were fed into the model. For FastSAM [8], a few masks were found, but were quite spiky and hard to interpret.
- **Transitory class** Initially, the dataset included an additional *transitory* class in between *safe* and *unsafe*, but classification proved to be difficult, so it was split between *safe* and *unsafe*.
- **Scalar labels** We also attempted to train on scalar labels (i.e. *off*=0 to *unsafe*=3) in attempts to address issues of samples on the boundary of *safe* and *unsafe*. However, this proved to have low accuracy.

Table 2. Validation confusion matrix. Rows = ground truth.

| | off | safe | unsafe |
|---|---|---|---|
| off | 54 | 0 | 0 |
| safe | 0 | 445 | 8 |
| unsafe | 0 | 0 | 98 |

Table 3. Test confusion matrix. Rows = ground truth.

| | off | safe | unsafe |
|---|---|---|---|
| off | 10 | 0 | 0 |
| safe | 0 | 212 | 1 |
| unsafe | 0 | 0 | 44 |

- **Labeling criteria.** Since labeling was mostly vibes based, initally the data was labeled in a way that made the *unsafe* class extremely rare (63 images in training dataset, 54 images in validation dataset, 9 images in test dataset). When we saw poor real world performance in preliminary 6, we reclassified the dataset to lower the threshold for the *unsafe*, which led to better real world performance.

## 5. Experiments and Analysis

In the following sections, we use recall as the reported metric because precision is not very useful due to the class imbalance.

### 5.1. Quantitative Results

**Validation.** Table 2 shows the confusion matrix of the final model on the validation dataset. The model reached *off* recall of 100%, *safe* recall of 98%, and a *unsafe* recall of 100% on the validation dataset.

**Test.** Table 3 shows the confusion matrix of the final model on the test dataset. The model reached an *off* recall of 100%, *safe* recall of 99.5%, and a *unsafe* recall of 100% on the test dataset.

### 5.2. Head only model

The mode that was checkpointed before backbone training reached an *off* recall of 100%, *safe* recall of 92%, and a *unsafe* recall of 91% on the test dataset.

This indicates that backbone training significantly helped to get a final extra 10% of recall in the *safe* and *unsafe* classes.

### 5.3. Training characteristics

We see that our training losses consistently decrease. However, our model takes a few epochs to become accu-

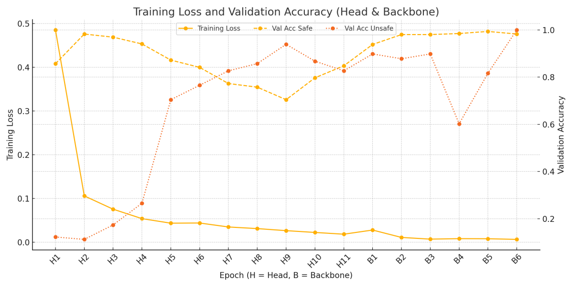

Figure 2. Setup used for evaluating inference.

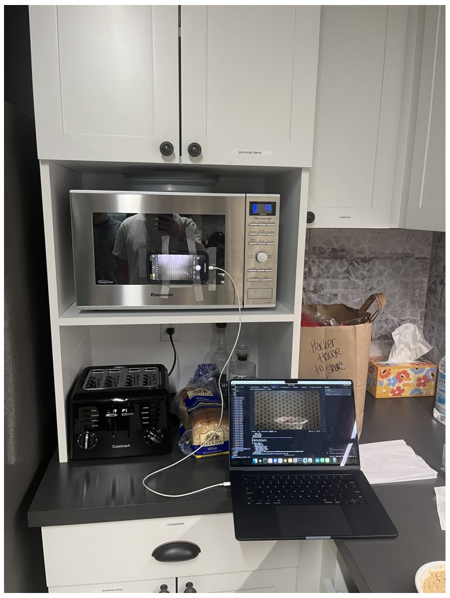

Figure 3. Setup used for dataset collection and evaluating inference.

rate on both the *safe* and *unsafe* class during validation. It seems to start predicting most things as *safe* and then move to over predicting for *unsafe* before settling to a reasonable validation accuracy for both.

There also seems to be a large drop in validation accuracy during backbone training that fixes itself within a few epochs, possibly due to an oversized learning rate.

## 6. Inference

Our inference setup has the iPhone taped to the microwave with scotch tape in approximately the same places used during training. A lightning to USC C cable connects the phone to an Apple M2 MacBook Air (16 GB RAM). We run a program that samples camera snapshots every 300ms and attempts to classify images.

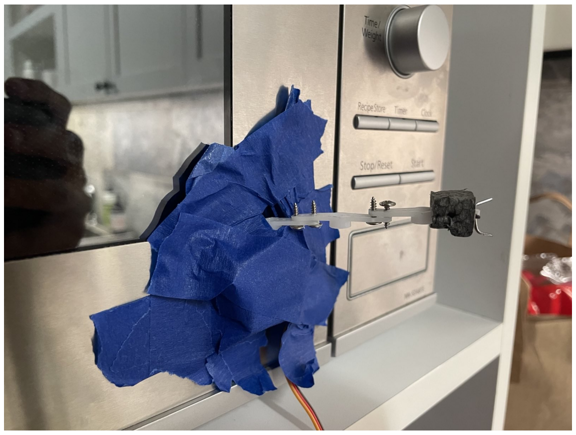
Figure 4. Servo-driven arm that presses the microwave *Stop* button.

## 6.1. Camera and compression specifics

We first tried to run the model with a Logitech C920 USB webcam mounted flush against the microwave door. Unfortunately, the webcam's autofocus kept locking onto the perforated metal mesh, leading to almost all frames being misclassified as *off* likely because the black mesh looked like the dark interior the network sees when the lamp is off. Replacing the webcam with the same iPhone 12 used to collect training data solved the problem. This highlights that, for specific tasks like this one, many factors must remain the same during training and inference without a very diverse dataset.

During inference, we discovered that our model particularly learned on the JPEG compressed resized images. Feeding the resized JPEGs directly from the live camera into the network lead towards a large bias towards *off* classifications. To address this issue, we process the image that the iPhone outputs exactly the same as we did before input into training. This requires resizing the images to 224 pixels on the longer side and applying 75% JPEG compression quality, and transform it to ImageNet standard distribution. This restores the expected performance.

## 6.2. Inference Benchmarks

Benchmarking on an Apple M2 MacBook Air (16 GB RAM) on inference over 157 different frames runs gives the following averages: model forward pass $4.8 \pm 0.3$ ms, preprocessing (transform plus JPEG compression) $2.2 \pm 0.2$ ms. The total 7 ms latency leaves comfortable headroom under the 300 ms frame interval and scales to $> 100$ FPS on an iPhone.

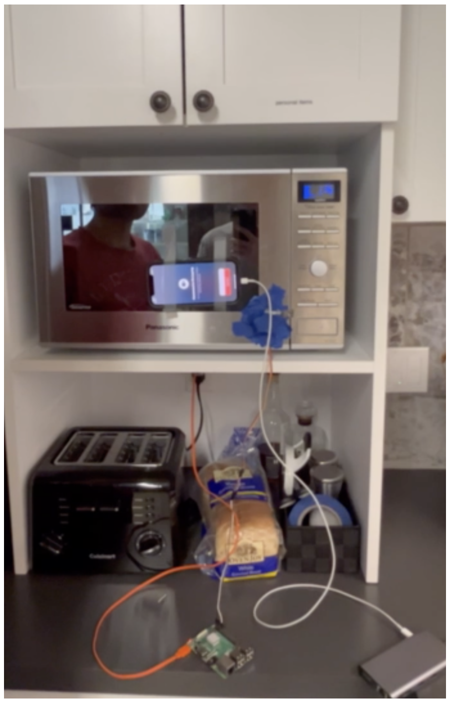
Figure 5. The end to end system for preventing overflows in real time.

## 7. Final Prototype

### 7.1. Button-Clicker Arm

To halt boil-overs we built a compact, servo-controlled arm that presses the microwave's *Stop* button.

The mechanism uses an SG90 micro-servo fitted with two 32 mm horns mounted on the output spline. A 1.5 cm $\times$ 1 cm piece of whiteboard-eraser foam is impaled by the tip to provide a broad, compliant contact surface. The foam pad is held in place with a bent paperclip that slots through the horn holes. The entire assembly is secured to the microwave door using masking tape.

Because the SG90 needs GPIO-level PWM, we drive it from a Raspberry Pi 3 B+. The Pi hosts a lightweight HTTP server that accepts `/move` requests from the MacBook Air, which serves as the main processing hub. Each request triggers the servo to move to a target angle, for which we hardcode a `pressed` and a `unpressed` angle.

### 7.2. Putting it all together

In the end to end system, we sample frames from the camera every 300ms and stop the microwave by triggering the servo arm as soon as we see an *unsafe* frame. Empirically, this solution seemed to guarantee no overflows or spills. This is likely due to the latency in frame sampling and HTTP communication to the Raspberry Pi.

A demo of the end to end system can be found at this link.

## 8. Future Improvements

Although the initial prototype is exciting, several further improvements must be made in the journey to practicality

- **Generalization** The model must generalize to more microwave backgrounds, meshes, vessels, and food types in order to be practical.
- **Camera replacement** The model must be able to use a camera that is not an iPhone 12, since users do not want to give up their phone. Ideally, the model would generalize to several different cameras.
- **Servo arm replacement** The current prototype utilizes masking tape and a makeshift arm. In order to be practical, the arm must be made smaller and sleeker. It must also be fastened to the microwave with something other than masking tape, which wears down over time.
- **Edge only architecture** In order to have a self contained workflow, the model must also run on Raspberry Pi to avoid users needing to hook up their laptops to the system.

## 9. Code

Code for the training and evaluation of the model can be found at: https://github.com/gpsanant/oatmeal.

## References

[1] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019.

[2] Tareq Khan. An intelligent microwave oven with thermal imaging and temperature recommendation using deep learning. *Applied System Innovation*, 3(1), 2020.

[3] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[4] TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. `https://github.com/pytorch/vision`, 2016.

[5] Rishabh Singh. Understanding and implementing mobilenetv3, October 2024. Accessed: 2025-04-28.

[6] Rejin Varghese and Sambath M. Yolov8: A novel object detection algorithm with enhanced performance and robustness. In *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, pages 1–6, 2024.

[7] Kan Wu, Jinnian Zhang, Houwen Peng, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Tinyvit: Fast pretraining distillation for small vision transformers, 2022.

[8] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything, 2023.