Ayina Anyachebelu
Deep Learning for Sensor Networks

Detecting Household Mold Using Transfer Learning, Edge Impulse, TensorFlow Lite and Arduino

Introduction

Research question: Can I create a sensor that identifies whether mold is present or not from real-time image capture?

This project aims to use deep learning on an edge device to detect the growth of mold in homes. The idea came from the personal experience of my neighbors complaining about mold regrowing in their apartments after their landlords haphazardly cleaned it. Mold is concerning because it leads to health problems such as respiratory issues, allergies and infections (CDC, 2021). It can also damage the structure of a home by weakening walls, floors and ceilings (Environmental Protection Agency, 2021). A mold detection device would be helpful for homeowners and tenants to monitor the success of any mold remediation service.

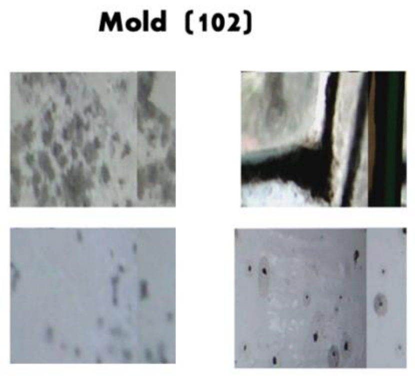
Device

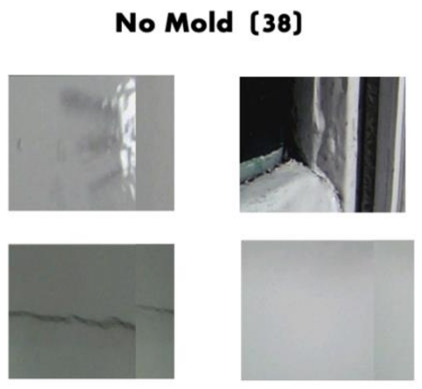
For this project, I use an Arduino Nano 33 BLE Sense with an OV7675 camera. I did not have access to a better-quality edge device and using a mobile phone would have been impractical, given that in reality, nobody would keep a phone strapped to a wall for days with only the task of detecting mold.
Using the Arduino Nano had several implications. The microcontroller's maximum RAM memory is 256 kb. This meant that I would have to deploy a small-sized model for my device to function, necessitating the use of TensorflowLite to enable quantizing my models.

Data

I collected all the data manually using the Arduino Device and OV7675. I chose to create my own dataset because I figured that given the poor camera quality of my device, training my model on high quality images from the internet would be ineffective during deployment because the images taken by the Arduino camera would be very different.

The photos in my dataset were taken in the apartments of my friends who had identified mold in their homes. Most of the photos taken were of window frames and ceilings. Although this might appear as a limitation of the dataset, considering that these were the locations with the most mold across all 3 apartments, it is reasonable that the model would be trained with a focus on these locations. It was quite difficult to aim the camera in the right diretion to capture the photos, so I used the Processing software to view the images being taken. While capturing "no mold" images, I made efforts to capture walls, ceilings, and windows that might have stains or marks, so that the final model would be able to differentiate between regular stains and mold. In the end, I had 38 no mold photos and 102 mold photos for a total of 140 photos taken with the Arduino.





Model Training

For this project, I chose to use a MobileNet model because I wanted to try out transfer learning since I had worked on creating custom CNNs before. I specifically chose the MobileNet family of neural network architectures because they are designed to be lightweight and efficient for use in mobile and embedded devices. The models achieve this efficiency by using a combination of depthwise separable convolutions, which separate the standard convolution operation into a depthwise convolution and a pointwise convolution. This approach reduces the number of parameters and computations required, allowing the models to run faster than complex models while still maintaining high accuracy (Sandler et. al., 2018).

I used the Edge Impulse software to ease the iteration process of testing out different parameters during model training. I separated my images using an 80-20 training-testing split and added a pre-processing step that would crop the images to be the same width and heigh, because this is the input type required by the MobileNet models. The original images were 176 x 144 and then cropped to 144 x 144. Each model was run based on a callbacks mechanism checking for a reduction in validation loss.

Experiments

I experimented with various parameters and evaluated the models based on the f1-score in addition to accuracy because my data is very unbalanced as there are more mold photos than non-mold photos.

Parameter Experiments

Below are the parameters that were explored during training and results for some of the parameter experiments. All these experiments used the MobileNetV2 0.35 model.

Parameter	Parameter Description	Experimented Range
Color Depth	If the input image is grayscale or in color	Grayscale, RGB
Number of Neurons	Number of neuros in the model's final dense layer of the model architecture.	0, 8, 16
Dropout	Dropout rate for the final layer of the model	0.1, 0.3, 0.5
Activation Function	Activation function chosen for the model	SoftMax, Sigmoid
Data Augmentation	Random data transformation including image rotation, resizing, or change in brightness	Yes or No

Model Description	Color Depth	Data Aug.	Number of Neurons	Dropout Rate	Activation Function	Accuracy	Mold F1 score	No Mold F1 Score	Validation Loss
Worst Performing Model	RGB	No	8	0.3	SoftMax	53%	0.52	0.55	10.57
Best Performing Model	Grayscale	Yes	0	0.1	Sigmoid	100%	1.0	0.99	0.05

In general, the models performed better when using grayscale images with data augmentation. This seemed reasonable given that most of the mold tended to be black and the backgrounds (ceilings, walls, window frames) were almost always white so RGB images were not necessary to extract important features. Removing the final dense layer (making number of neurons = 0) helped the model perform better by avoiding overfitting while reducing the dropout helped to avoid underfitting. The sigmoid activation function performed only slightly better, aligning with the fact that sigmoid functions is a great fit for binary classification problems like my project.

Memory Experiments

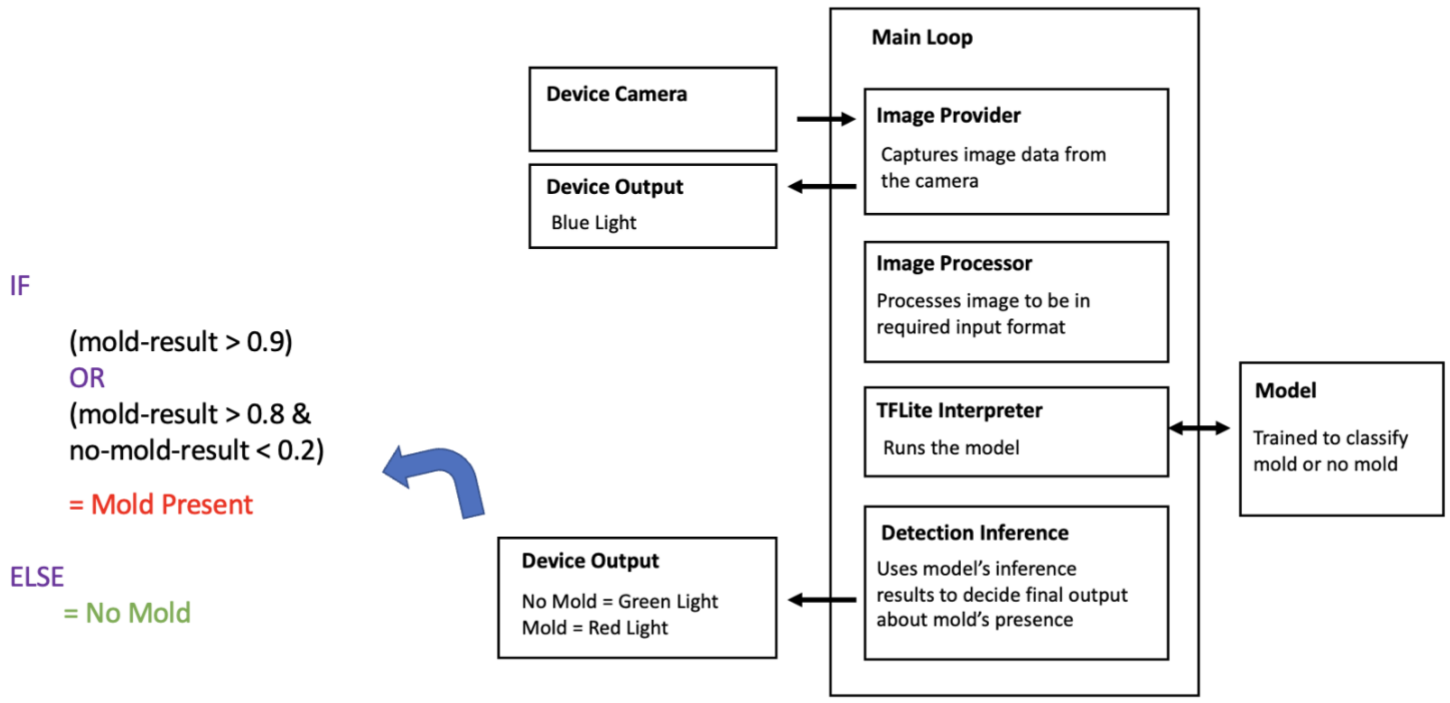
In addition to parameter exploration, I also had to experiment with different types of MobileNet neural networks and image processing in order to arrive at a memory usage that could be deployed. My initial memory problems originated from my lack of attention to the RAM usage of the various MobileNet models. I conducted my initial experimentation using the MobileNetV2 0.35 architecture. However, this model uses around 297K RAM with default settings which is incompatible with Arduino's memory capacity. All the V2 MobileNet models required a RAM usage greater than 256 kb so I tried out the V1 architectures. While the most lightweight V1 models all had very small memory usage, their performance was deplorable. The only V1 model with good performance was the MobileNetV1 0.25. However, even though its maximum RAM usage was below 256 kb, I kept running into memory issues during deployment. In the end, I had to reduce the input size of the images, changing them from 144 x 144 to 100 x 100, so that the model size was smaller enough to be deployed. The final model used was a MobileNetV1 0.25 with input images of 100 x 100, resulting in a peak ram usage of 137 kb and validation accuracy of 89.3%.

Model	Input Image Size	Peak RAM usage	Testing Accuracy	Able to be Deployed?
MobileNet V2 0.35	144 x 144	357 kb	100%	No
MobileNet V1 0.25	144 x 144	221 kb	98%	No
MobileNet V1 0.2	144 x 144	109 kb	56%	Yes
MobileNet V1 0.25	120 x 120	186 kb	94%	No
MobileNet V1 0.25 **	100 x 100	137 kb	89%	Yes

**final model used in deployment

Deployment

Below is the framework for the deployment of the model on the edge device. I used the template Arduino Sketch file produced by Edge Impulse, which I edited to add features and functionality based on my project's needs. With the final sketch file, when an image is captured (every 4 seconds), a blue light switches on and off. If the final decision is that there is no mold, then a green light is turned on; if not, a red light turns on.



The development of the framework was also a result of various experiments during deployment. For example, the blue light feature was added because it was difficult to tell when an image was being captured versus when inference was occurring.

Additionally, the final output for “mold” or “no mold” was decided after testing out the device and seeing that the model's output probabilities that the image was of mold almost always tended to be higher than that of no mold. After various iterations I decided that for the final output to say “mold”, the probability for the mold category had to be greater than 0.9 or greater than 0.8 with the probability of no mold category being less than 0.2.

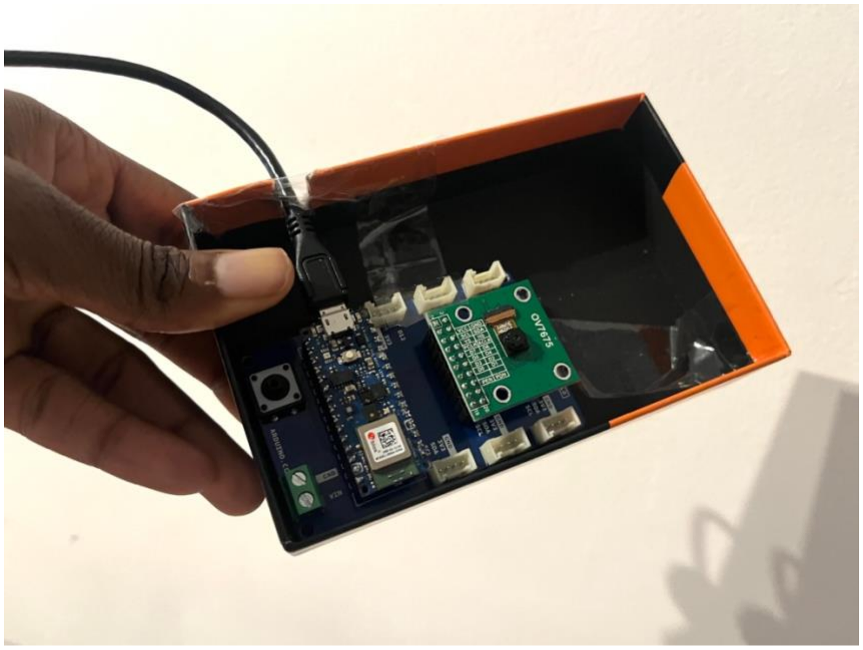
I also had to play around with how and when the lights turned on and off and delays in image capture because it was difficult to see view the blue light when the green or red inference lights were already on from the last image captured.

In general, the deployment worked well especially on moldy ceilings and on paper with dots and marks used to simulate mold. It was more difficult to work with window, likely because it is very hard to aim the Arduino device so that the moldy parts of the windows are exactly in the frame of the OV 7675 camera, and without the Processing software running, the user cannot tell what the device is capturing. However, in a real-world situation, the device would be perfectly directed at a spot with mold problems, so this would not be an issue.

During testing, I also realized that the device works best in places with bright or white walls. I noticed this because the deployment worked well in most rooms but when tested in one room with black-designed tiles and wallpaper sometimes it would classify the scenery as mold. It makes sense that the brighter, white walls would be better environments for testing since these where the places I took images that the model was trained on.

Future Work

I also created a preliminary build for the device to mirror how it could easily be deployed in homes (seen below). It could also be connected to hooks from ceilings as this is where a lot of mold tends to accumulate. It will be most useful to be placed in locations with past mold problems to monitor if the mold regrows after remediation. While this is a very early-stage prototype for the device, in the future, instead of changing lights, the device could send an email notifying the user of mold. This would especially be useful if the device is in location of the home like a basement or cellar that the user does not access every day.



Reflections

- Don’t take shortcuts in data collection. Even though it took much more time, making a custom dataset by taking photos with the Arduino device allowed me to train my model with images similar to the photos that would be captured during actual deployment, which was much better in the long run.

- Be mindful of underfitting and overfitting while training. When using the very lightweight versions of MobileNet, the models were underfitting the data resulting in very low accuracies. On the other hand, the heftier architectures required my removing an extra final layer in order to not overfit. At the same time, I kept a low dropout to avoid underfitting –It was a balancing act.

- Be conservative when considering memory allocations. Despite, having RAM usage being lower than required, I kept running into memory errors. This might have been because of the memory taken up by other variables needed for my program to work. It is always better to be conservative about how much memory you have available.

- Inference is what you make of it. The Deep learning model doesn’t actually classify the image. It spits out a probability of it being in each category. This is a good thing, because with experimentation, you can observe what probabilities of each category actually match the ground truth and use this information to decide the final output which is how I developed the extra code making the final classification of mold or no mold.

- Consider timing and latency of device based on the end use. My final sketch file had photos taken every 4 seconds. I chose this because it was slower than the original edge impulse code (2 secs) for easier viewership while still being quick enough to show the deployment. In the real world, a photo checking for mold would only need to be taken once a day, so timing for the device really depends on the end use (e.g., real world vs demonstration).

- Test deployment in different environments and scenarios. By testing in different scenarios (day vs night, various locations), I was able to find the limitations. For example, I noticed that testing worked better in my room than in my kitchen because of the black tiled walls in my kitchen, which meant that the device worked better with white/bright walls. This may not necessarily be a limitation depending on the end use of the project. In my case, most ceilings and window frames (which is where the mold was found) tend to be white or bright colors, so the focus of this project would be getting the best accuracy in this specific environment.

Works Cited

CDC. (2021). Mold. Retrieved from https://www.cdc.gov/mold/default.htm

Environmental Protection Agency. (2021). Mold. Retrieved from https://www.epa.gov/mold/mold-and-your-home

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 8985-8993).

Appendix

The Edge Impulse Project: https://studio.edgeimpulse.com/studio/184531

Video Presentation: https://www.youtube.com/watch?v=LyyfgAZ16uk

GitHub Repository: https://github.com/yinaanyachebelu/mold-detection-arduino