CSCE 585: Machine Learning Systems: Milestone 2: Computer Vision Assisted Disaster Rescue

Cole Lewis, David Duggan, Daniella Mallari

November 3, 2022

Contents

1 Project Repository

https://github.com/csce585-mlsystems/CVDisasterRescue

2 Introduction

The problem in question is how to meaningfully combine both visual and audial inputs to find possible disaster victims in the wake of calamity. Not only must the robot or drone be able to recognize victims around it, but must be able to locate and prioritize those closest as well . Through the combination of real time human detection and sound source localization of detected human-generated sound via microphone array, we can determine where potential disaster victims may be located relative to the robot and whether or not there are any victims in its immediate vicinity.

3 Problem Statement

While we continue to hone and develop our human anatomical recognition model through tuning hyperparameters such as learning rate, number of epochs, and batch size, we have shifted our focus towards developing a model to recognize human-generated sounds and then overlay a visual indicator of detected sounds on to the camera input whose positions on screen will correlate to direction of origin.

4 Technical Approach

Our approach to locate and rescue people in disaster zones combines visual and audial inputs to create the perception system for a robot to locate people in need of rescue. Combining visual and audial inputs allows the system that our perception is running on to localize its search to important areas. The audial input will listen for sounds that could be made by a human (things like footsteps, screaming, speech, etc.) using a microphone array (schematic included in Appendix A). The visual system will combine a camera with an object detection algorithm trained to find people within a video feed. This means our algorithm needs to be able to function in real-time, with fast, yet accurate, inferences. To solve this problem, we chose the Yolov3-tiny object detection algorithm. The Yolo object detection algorithms are known for their fast inferences because of their ability to generate predictions while only looking through the image once. To function on a video input, the video will be sliced into frames and inference will be run across each frame. To further increase the prediction speed, we use the 'tiny' version of the Yolov3 algorithm which is a smaller version. Preliminary results are shown in the below section. Due to the numerous difficulties and poor performance experienced with our previous YOLOv3 model, our team decided to transition to using a YOLOv4-Tiny model which was already trained and

quantized which made implementation on personal hardware both easier and more effective. Audio input from the microphone array will be preprocessed into spectrograms and then passed through a Convolutional Neural Network which will classify whether the captured audio is sound potentially made by a human. The idea to use a CNN on spectrograms instead of a sequence-based network such as a Recurrent Neural Network came from this article [1]. This makes our audio system tolerant to background noises that would not benefit the search, effectively creating a ranking model that prioritizes human-made sounds over any other sounds. The microphone array will allow the system to capture the direction that the sound came from, and an arrow will be overlaid on the screen, pointing in the direction that the sound originated from. In the case that multiple human sounds are heard from different directions, the loudest one of them will be prioritized first as they would be assumed to be closer and in need of assistance faster.

5 Preliminary Results

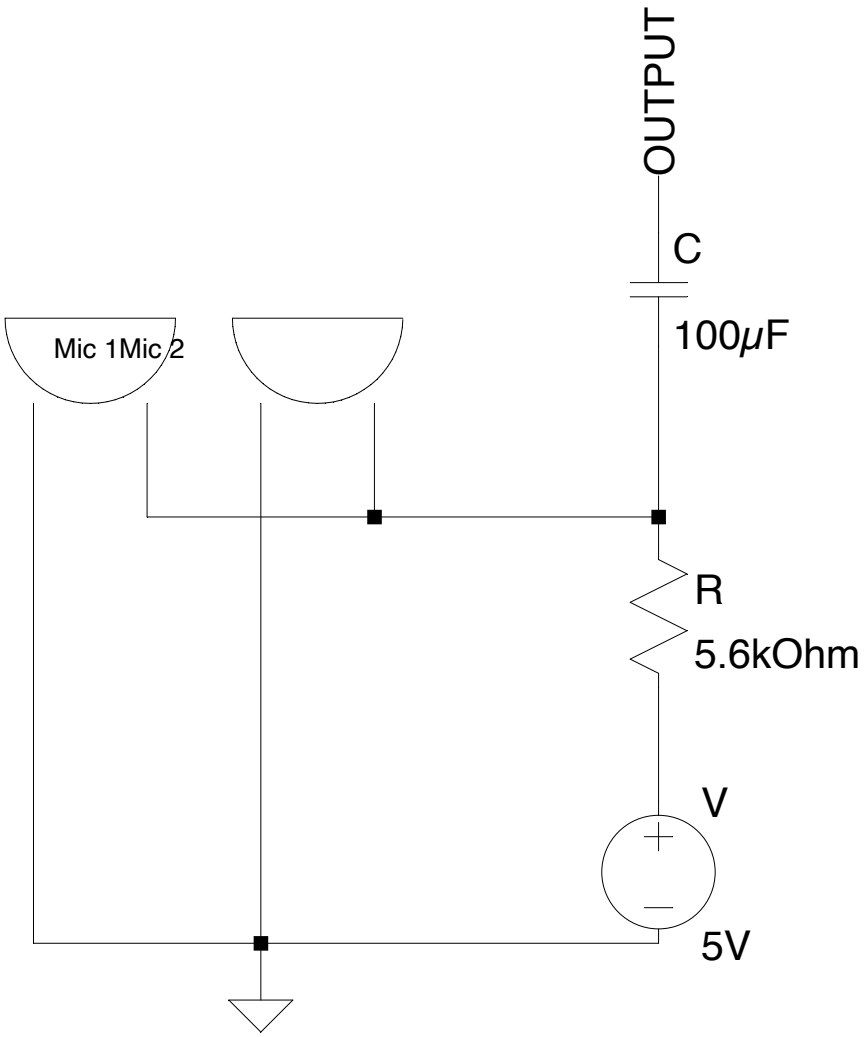
After switching to use of our new human detection model, our team wanted a benchmark for its performance. We downloaded 500 images of people from Google's Open Images Dataset and their associated annotations and calculated the Intersection over Union (IoU) score of each bounding box detected in the image and the bounding box coordinates present in the annotation. The average IoU score per bounding box over the entire set is 0.44. However, the results should be interpreted loosely since the dominant use of this model is real-time human detection which does not demand the same level of accuracy nor depend on it for effective use. Moreover, differences in labeling between the datasets (i.e., who or what in the scene is being marked as a person) are likely to have contributed to a lower IoU score than would otherwise be reported. During real-time detection on our hardware, the model can perform at a consistent 11 to 11.1 frames per second, a marked improvement over the previous model's 7 frames per second and completely sufficient for the needs of our project. To prepare data for the sound classification model, data from the ECS-50 dataset [2] and a subset of the Flickr Audio corpus [3] were combined. The ECS-50 dataset consisted of many different classes, some potentially being sounds created by humans such as footsteps, laughing, and crying. A Python script was used to extract samples from potentially human classes from this dataset. Samples were placed in a directory structure by their classes to serve as the labels for the data. The two classes used were human sounds and nonhuman sounds. After dividing up the ECS-50 dataset, there were 1600 nonhuman examples and less than 1000 human sounds examples. The Flickr Audio corpus consists of 40k examples of human speech, so examples were then taken from the Flickr corpus and added to the human sounds class until the number of samples between the two classes were equal. This resulted in a final dataset of 3200 examples, with an equal amount of data in both classes. For each example in the dataset, the audio was resampled to all be the same length and sample rate. The short-time Fourier transform

was then applied to each example to create a spectrogram from the audio waves. The idea behind this was to convert the audio to images, where a CNN could be applied. Each spectrogram was then converted to RGB and the pixel values were standardized on a scale from 0 to 1. After preprocessing the data, a CNN was trained using a test set size of 20%, and a validation size of 10% of the training set. Hyperparameter tuning was done on different elements of the model training such as the batch size, learning rate, and number of epochs. Different architectures were also tested such as adding batch normalization, adding dropout, and adding additional convolutional/hidden layers. Some preliminary results of the model tuning are shown in Appendix A in Table 1. To evaluate the models, test set accuracy as well as the confusion matrix on the test set were used. Accuracy is a simple measure that allows us to see how often the model was correct, and the confusion matrix allows us to dig into what types of errors the model was making. Some observations were made on the model's performance with different hyperparameters. For example, while sound_model3 had a lower accuracy than sound_model2, it made less Type II errors. Type II errors (false negatives) in our case would be the model predicting that there was no human in need of rescue when there actually was. Missing someone could have catastrophic results, so it's important to reduce the amount of false negatives as much as possible. Type I errors (false positives) would signal there is a person when there actually is not. While this would be inconvenient, a false positive would not have the same implications that a false negative would have. It was also observed that sound_model5 was overfitting very early on, resulting in a poor test accuracy.

6 Appendix

Figure 2. Preliminary hyperparameter search for the sound classification CNN results.

Model name	Epochs	Batch size	Learning rate	TP	FP	FN	TN	Accuracy
sound_model2	150	32	0.001	303	14	109	210	0.806
sound_model3	300	32	0.001	178	139	23	296	0.74
sound_model4	200	16	0.001	296	19	118	204	0.784
sound_model5	300	16	0.0001	309	6	198	124	0.67



--- C:\Users\dmtma\OneDrive\Documents\LTspiceXVII\MicrophoneArray1.asc ---

Figure 1: microphone_array_schematic