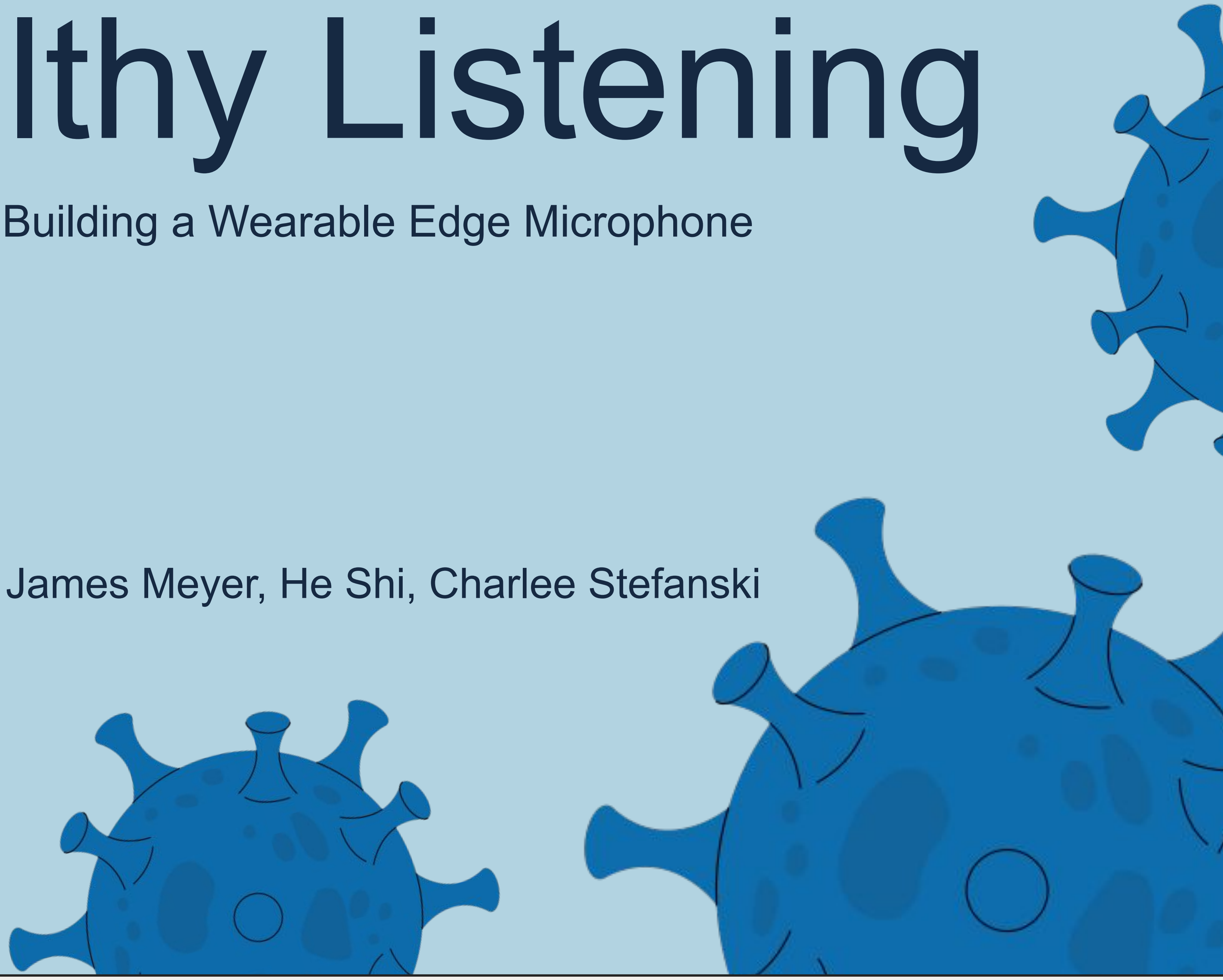
Healthy Listening

Building a Wearable Edge Microphone

Team Members: James Meyer, He Shi, Charlee Stefanski



Abstract

Visits to the doctor's office play an important role in the healthcare process. Unfortunately however, not all medical information that may be necessary to properly diagnose a patient can be observed or quantified in a short in-person visit. Given these limitations, wearable edge devices have the potential to add significant value to the medical field, allowing for longer-term data collection on patients that can assist medical professionals in making more accurate diagnoses.

One example of this would be the symptom of coughing. Coughs come in many different varieties, and both the tone and frequency of coughs may vary depending on the particular ailment, however it can be difficult to precisely measure these within the confines of an office visit, with medical professionals often forced to rely on their patient's memories or efforts to recreate their symptoms.

In this project, our goal will be to develop an "edge microphone", capable of continuously recording audio from a patient, detecting coughing, and transmitting the recordings back to a cloud server to be used by medical professionals during their diagnostic process.

Outline

01 Process Overview

02 The Data

03 The Models

04 Architecture

05 Conclusion

Process Overview

The process to build our audio system

Step 1

Identify a high-quality dataset that can be used to train and validate a high-accuracy audio classification model

Step 2

Various models will then be trained on the dataset in the cloud to take advantage of GPU computing resources.

Step 3

Once an optimal model is identified it will then be optimized for the edge before exporting the weights and parameters for use in inference on the edge device.

Step 4

Set up the programs and network necessary to record audio on the edge device, run inference on it.

Step 5

Then transmit it back to a long-term cloud storage device for future analysis.

The Data

Dataset & Data Creation

Model Training Dataset: ESC-50

1. Contain 2,000 five-second WAV audio files, each of which fall into one of 50 different labels spread over 5 categories including Animals, Natural Sounds, Human Sounds, Domestic Sounds, and Urban Noises.
2. The audio files are distributed evenly across all 50 labels, with each label containing 40 unique examples.
3. All the data files were manually collected and sorted from the larger Freesound.org database which includes a vast array of Creative Commons Licensed (CCL) audio files.

Test Dataset: collect, label our own audio samples

1. To better replicate the types of audio that would be generated by the edge device, we decided to collect and label our own audio samples using the same USB Webcam hardware.
2. In total we collected a test dataset of 120 five-second audio files, totaling 10 minutes of audio.
3. We simply separate the data into either “cough”, which would be assigned a value of 1, or “not a cough”, which would received a value of 0.
4. Of the 120 files in the test dataset, 20 were labeled as cough.

The Data

Understanding Audio Data

Image Data vs Audio Data

- Pixels vs Samples
- 44.1kHz
- 16 bits/sample

Understanding Spectrograms

- Y-axis: Frequency (Hz)
- X-axis: Time (sec)
- Color: Strength (dB)

Fig. 1 - Coughing

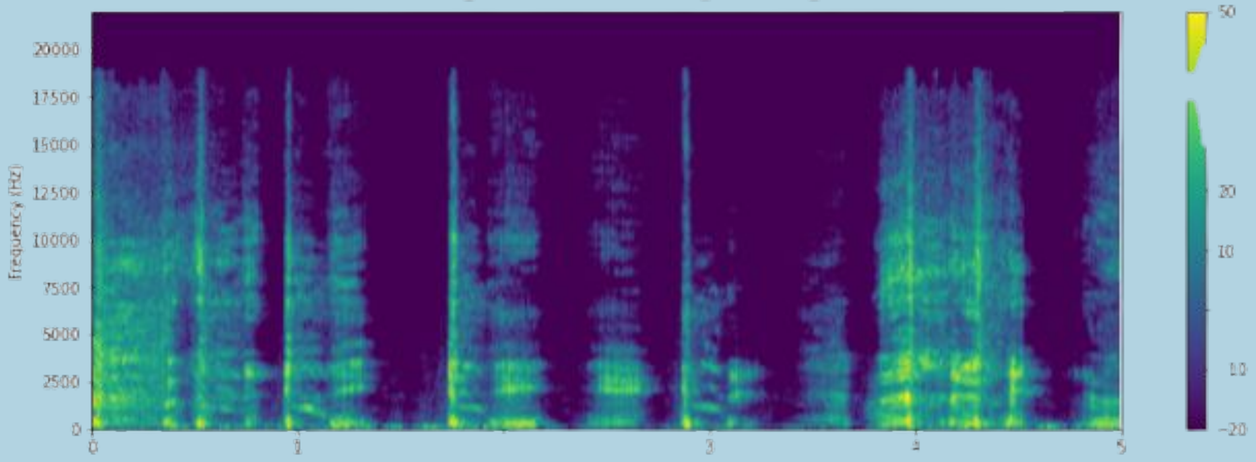
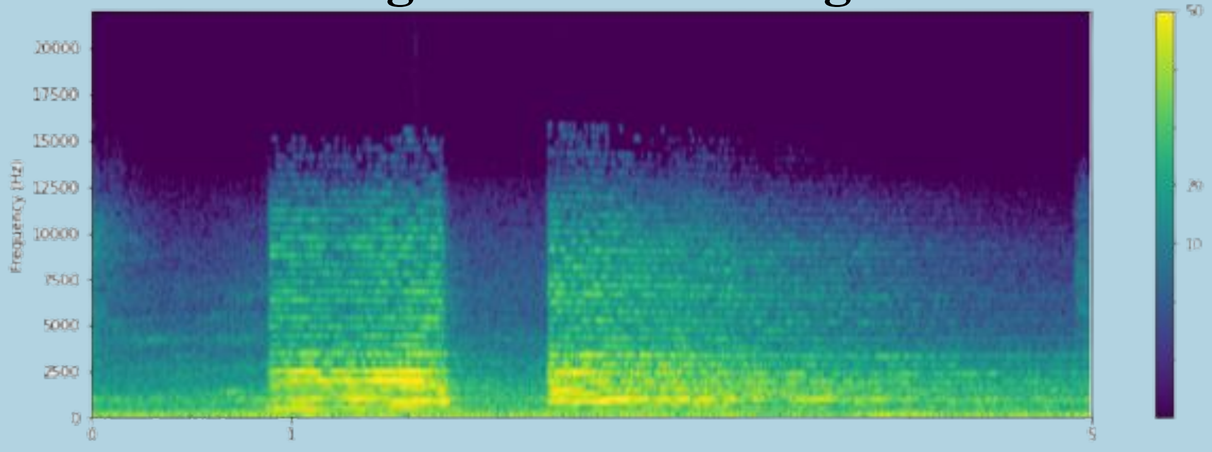


Fig. 2 - Car Honking



The Data

Data Augmentation

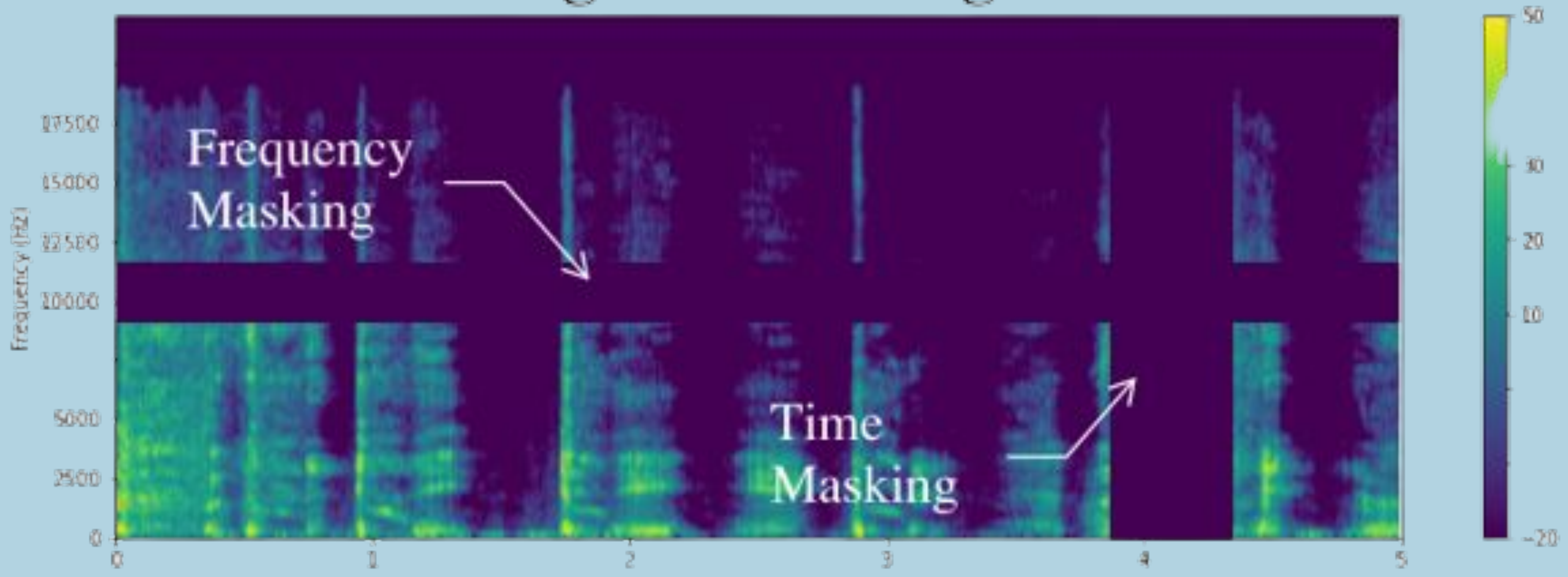
Masking

- Similar to cutout image augmentation technique
- Frequency vs Time masking

Shifting

- Translations of the spectrogram
- Pitch vs Time shifting

Fig. 3 - Masking



The Model: Data Preprocessing & Augmentation

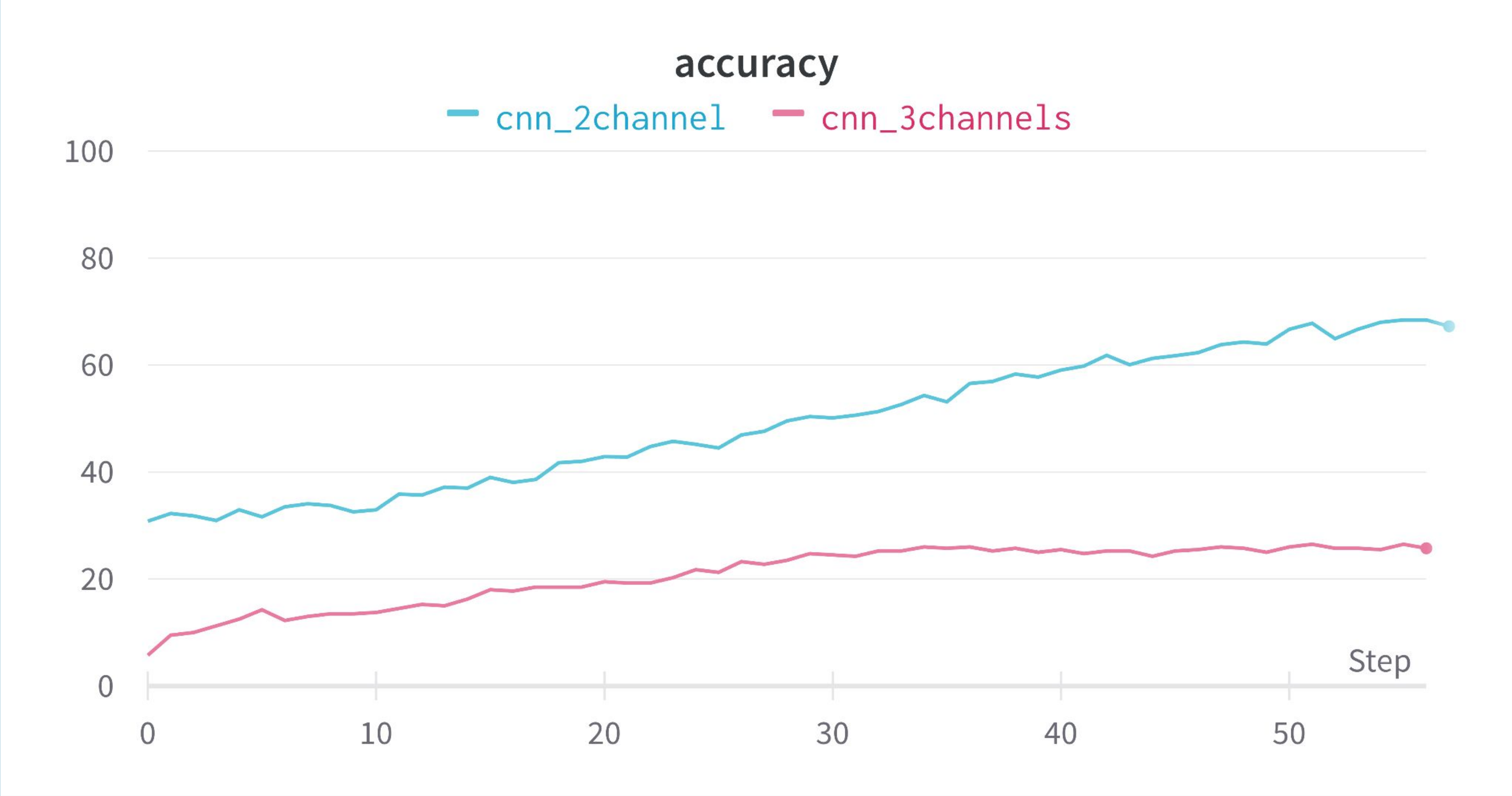
Method A

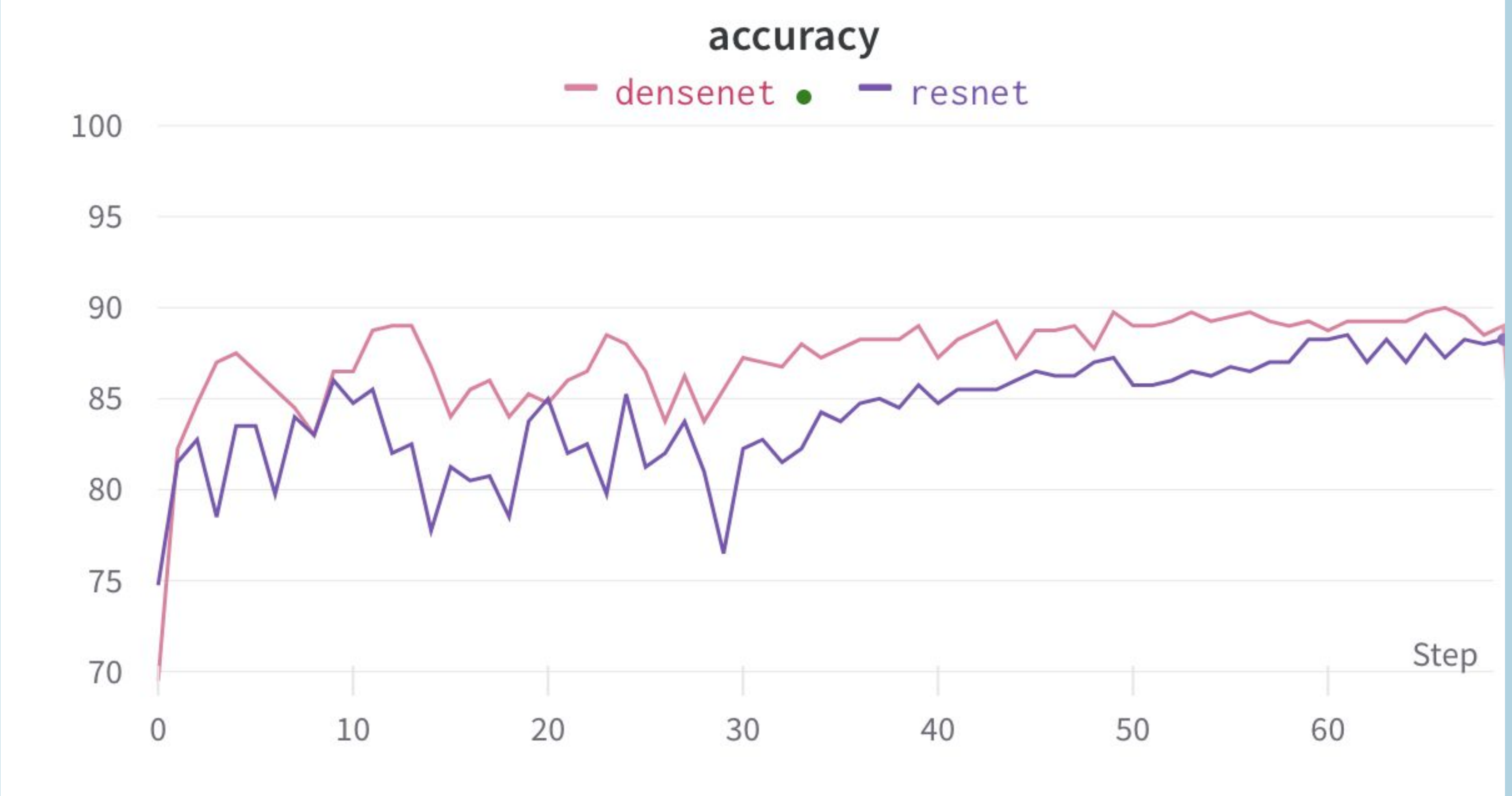
```
data, sr = torchaudio.load(full_file_name)

# Resample to two channels
data = torch.cat([data, data])

# Generate a Spectrogram from the audio file
data = transforms.MelSpectrogram(sr,
n_fft=1024, hop_length=None, n_mels=64)(data)
data =
transforms.AmplitudeToDB(top_db=80)(data)
```

Method B

```
num_channels = 3
window_sizes = [25, 50, 100]
hop_sizes = [10, 25, 50]

specs = []
for i in range(num_channels):
    window_length = int(round(window_sizes[i]*sampling_rate/1000))
    hop_length = int(round(hop_sizes[i]*sampling_rate/1000))
    clip = torch.Tensor(clip)
    spec = torchaudio.transforms.MelSpectrogram(sample_rate=44100,
n_fft=4410, win_length=window_length, hop_length=hop_length,
n_mels=128)(clip)
    eps = 1e-6
    spec = spec.numpy()
```

The Model: Training





The Model: Inference

CNN: 2/20 Coughs
Densenet: 16/20 Coughs
Resnet: 19/20 Coughs

```
model_load = torch.jit.load('DataSci251_FinalProject/pt_Files/best_resnet.pt')
```

```
model_load.eval()
cough_count = 0
with torch.no_grad():
    for batch_idx, data in enumerate(test_dataloader):

        outputs = model_load(data[0])
#         print(outputs)
        _, predicted = torch.max(outputs.data, 1)

        pred_int = int(predicted)
        print('file name: ', data[1], ' label: ', pred_int)
        if pred_int == 24:
            cough_count += 1
            print('COUGH')
```

```
file name:  ('audio_222002.wav',)  label:  15
file name:  ('audio_222026.wav',)  label:  25
file name:  ('audio_222051.wav',)  label:  46
file name:  ('audio_222006.wav',)  label:  31
file name:  ('audio_cough_8.wav',)  label:  24
COUGH
file name:  ('audio_221654.wav',)  label:  31
```

Architecture

Due to a lack of hardware resources, a low-powered local VM and USB webcam were used to emulate the hardware and computing/storage resources that would be available in an actual edge microphone.

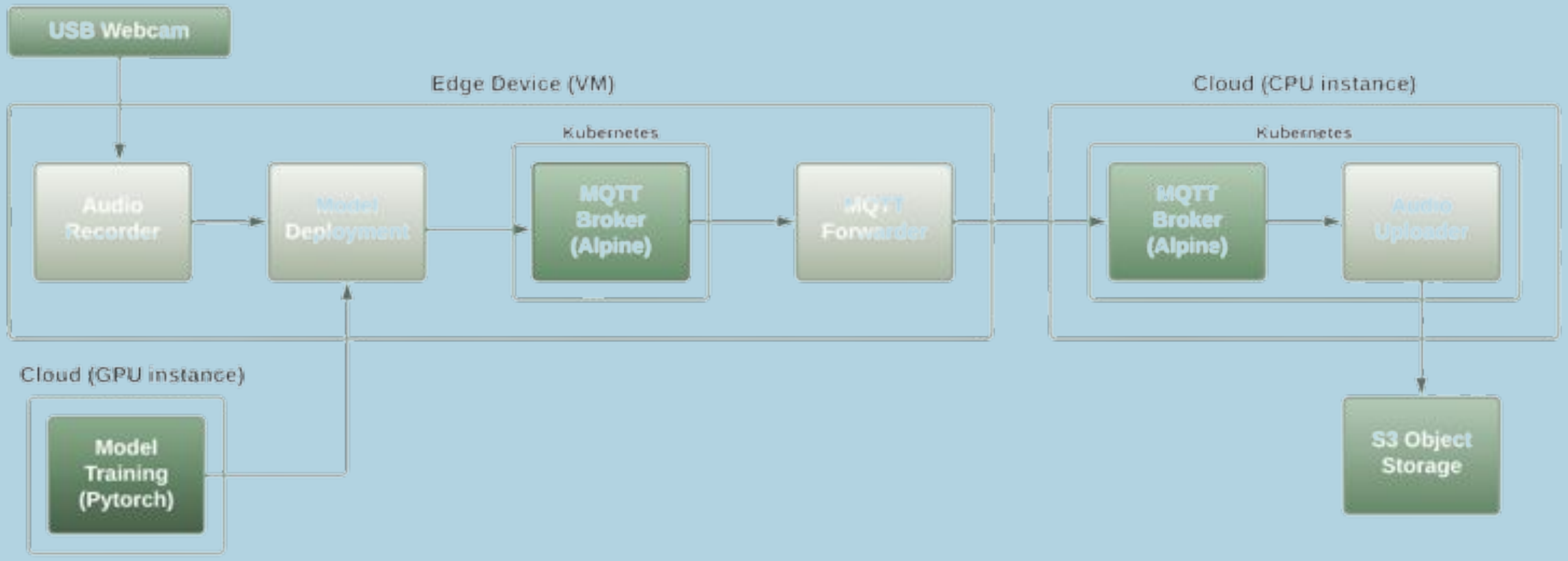
1. **Record audio** and save it in 5-second audio files in a temporary staging directory.

2. Load the model, scan the staging directory for audio files, and **run inference** on them audio files as they come in

3. If a cough is detected, **pass file along to the MQTT broker** before deleting it. Otherwise, just delete the file.

4. **Forward messages** on to another MQTT broker running on a low-powered cloud instance.

5. In the cloud, **convert** the messages back to audio files before **uploading the file to an S3 object storage bucket** on the cloud.



Demonstration Video

Conclusion

Challenges

Preprocessing the test data

- Different test code required for different models

Edge VM system delay

- Multiple processes and limited resources

Conclusion

Areas of Future Development

Implement optimization techniques

Fully containerize and deploy in Kubernetes

Refine model through human-review of collected data

Thank you for listening!

Q&A