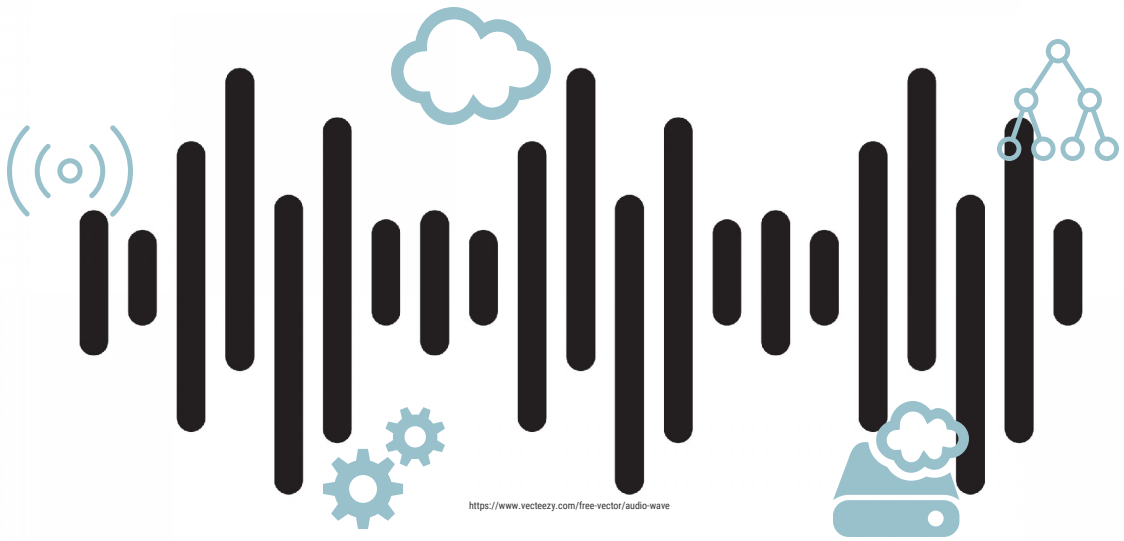
jonas.degnan, andresdelarosa, dbaron, younghoonkim29
@ischool.berkeley.edu

Sound Classification with YAMNet

Near Real-time Acoustic Inference at the Edge with Cloud Scalable Infrastructure Support



Jonas Degnan, Andres de la Rosa, Dmitry Baron, Hoon Kim
Deep Learning in the Cloud and at the Edge
University of California at Berkeley School of Information

GitHub
https://github.com/UC-Berkeley-I-School/w251-sp22-final-adhj

Motivation

Could more impactful acoustic information be delivered by aggregating environmental context and different acoustic classifications generated close in time?

Often, sounds heard in isolation do not trigger a physical or psychological response in people. Either due to fatigue and cognitive saturation or because humans developed this ability [1] to socialize [2] or survive. Even when we do hear sounds, we may not be able to act on them because we cannot interpret their meaning based on other environmental sounds or signals, or we are unable to take action due to physical or psychological [3] limitations.

To explore a possible solution to this problem, our team explored using acoustic based deep learning models to be able to develop and deploy lightweight models optimized for specific distress-related sound classification on edge devices that would aggregate their inferences and observations to a cloud-hosted database, object-store, and web server for analysis and machine- [4] and deep-learning [5] model development.

Our team set out to evaluate and develop the following milestones to address this problem:

Exploring Various Acoustic Deep Neural Networks Derived from Computer Vision Architectures
There is a growing corpus of acoustic models with many tuned to a specific acoustic domain. We wanted to evaluate the applicability or utility of these and the larger models towards an inference solution.

Designing a Distress-Centric Acoustic Model
Using transfer learning on the large acoustic models, we planned develop a lightweight model augmented with more varied training data to classify two distress-adjacent classes and apply data augmentation techniques to simulate the noisy environments [6] edge devices typically operate in.

Deploying a Scalable Architecture to Support Edge Devices
Due to the limitations of acoustic wave energy propagation [7], many edge devices are necessary to mitigate these effects. We planned to develop a easily deployable edge pipeline and scalable cloud architecture to support the streaming data generating processes.

Gaining Unique Insights from Acoustic Environmental Signals
With the inference data collected, we proposed a map-based notification platform to visualize the data collected and possible applications for using it with other data sources.

1 Acoustical Society of America (ASA). "Scientists tuning in to how you tune out noise." ScienceDaily. ScienceDaily, 8 May 2012. <www.sciencedaily.com/releases/2012/05/120508152007.htm>.
2 An example is the cocktail party effect, a phenomenon of the brain's ability to focus one's auditory attention on a particular stimulus while filtering out a range of other stimuli.
3 Large groups of disabled, abused, minor, and elderly individuals are unable to advocate for themselves. Outside observers may also not advocate or interfere due to the bystander effect or social norms.
4 Clustering and regression based models could be developed to identify patterns and forecast incidents of distress or violence when combined with demographic, geographic, and socioeconomic variables.
5 Observation inference validation can be used to improve model training sets and experimental models using generative observations can be evaluated for edge cases.
6 Low signal-to-noise (SNR) ratio environments.
7 Doppler shift, reflections, absorptions, and amplitude/frequency-dependent propagation distances are several challenges facing the collection of accurate, detectable acoustic signals.

Exploring Various Acoustic Deep Neural Networks Derived from Computer Vision Architectures

Out team considered multiple models for this task: Ketos, PAMGuard, PyTorch Audio Classification, VGGish (Visual Geometry Group), and YAMNet (Yet Another Mobile Network). The first two were too large and geared towards underwater acoustics. The last three had large classification ontologies and wide adoption. We selected YAMNet because the dataset (Google AudioSet [1]) used to build the model had a well defined structure and ontology, and quantized models are available for use on the edge. Because YAMNet's architecture does not provide inference confidence valuations, VGGish was also explored in the cloud with the intention of randomly sampling observations and providing a second inference on YAMNet's results.

YAMNet: Under the Hood

YAMNet is a pre-trained deep neural net that predicts 521 audio event classes based on the AudioSet-YouTube corpus, and uses the MobileNet v1 depthwise-separable convolution architecture. The model extracts "frames" from the audio signal and processes batches of these frames. The frames are 0.96 second long and the model extracts one frame every 0.48 seconds. YAMNet returns 3 outputs:

521 predicted class scores per frame

Used to directly identify audio events by aggregating per-class scores across frames (e.g., mean or max aggregation)

1,024-dimensional transfer learning embeddings

To be used as a high-level feature extractor for shallower models with small datasets.

Per-frame log-mel spectrograms

Useful for visualizations or as an image input to other training datasets.

YAMNet is pre-trained on AudioSet, developed by Google, and is both a sound vocabulary and dataset, consisting of an expanding ontology of 632 audio event classes and a collection of more than 2 million human-labeled 10-second YouTube video sound clips. Audio classes include music, speech, vehicle, animal, siren, silence, wind/etc. This dataset is offered for download in 2 formats: CSV files describing, for each segment, the YouTube video ID, start time, end time, and one or more labels; and 128-dimensional audio features extracted at 1Hz. A VGG-inspired acoustic model [2] extracts features trained on a preliminary version of YouTube-8M. PCA-ed and quantized features are made compatible with audio features provided with YouTube-8M and stored as TensorFlow Record files.

YAMNet has limitations: classifier outputs are not calibrated across classes, so outputs cannot be treated as probabilities; for any given task, a calibration with task-specific data is necessary to then allow for proper per-class score thresholds and scaling. Additionally, YAMNet has been trained on millions of YouTube videos and although these are very diverse, there can still be a domain mismatch between the average YouTube video and the audio inputs expected for any given task; one should expect to do some amount of fine-tuning and calibration to make YAMNet usable in any system built.

1 Google Research. (2017). A large-scale dataset of manually annotated audio events. Dataset. https://research.google.com/audioset/
2 Hershey, et. al. (2017). CNN Architectures for Large-Scale Audio Classification. https://arxiv.org/abs/1609.09430

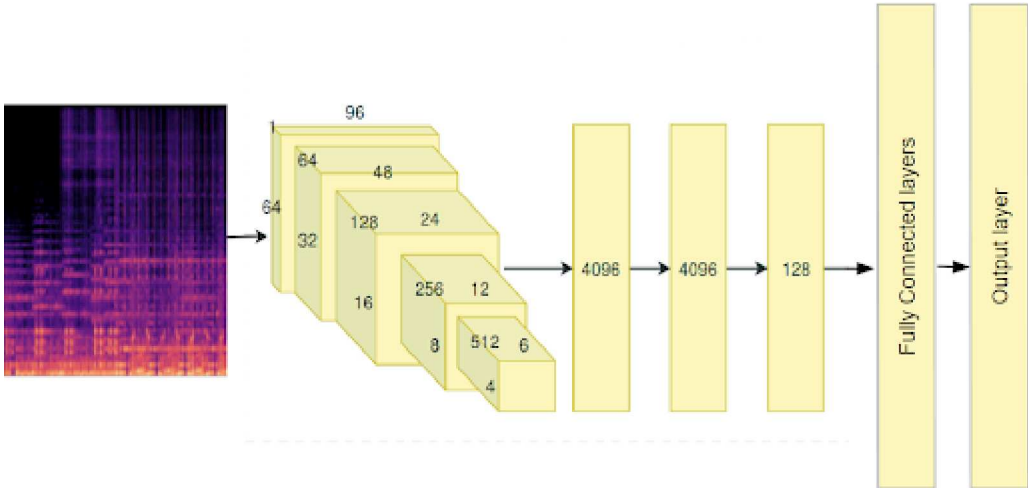
Exploring Various Acoustic Deep Neural Networks Derived from Computer Vision Architectures

Transfer Learning in the Cloud with VGGish

While we have YAMNet on the edge, we decided to develop another classifier model on the cloud for inference. Since AWS EC2 instances can provide powerful computational power to both train and test a model, we decided to do transfer learning with VGGish.

Before diving into the VGGish architecture, it helps to understand how audio signals can be transformed. Audio signals consist of several single-frequency sound waves. When processed with Fourier Transforms, these sound waves transform into spectrums; in other words, frequencies. Spectrograms are generated by stacking multiple spectrums into an image using color for intensity. This series of transformations from raw audio to spectrograms takes place at the first step of VGGish. To train the VGGish model to cater towards our needs, it was important to have our train audio files to have the same dimensions going into Fast Fourier Transform (FFT) to produce spectrograms. Therefore, we set a desired sample rate and duration, and have paddings for those that did not meet the dimension requirements.

VGGish shares its base for training on the Google Audio Set with YAMNet. It is pre-trained on the same YouTube corpus, but inspired by VGGNet instead of MobileNet v1. Unlike VGGNet, however, VGGish has a lighter network architecture as it drops the last group of convolutional and max pooling layers. In short, VGGish has 4 sets of convolutional and max pooling layers followed by 3 sets of fully-connected layers to produce (n, 128) dimension feature embeddings.



Feature embeddings from VGGish then need to be fed to a classifier for inference. Since our input audio files were already detected (pre-classified) for further classification, we decided to construct a very lightweight fully-connected network (double layered fully-connected network with 4096 nodes on each with a dropout layer before the output layer) to finalize the classification. Since time efficiency was also one of the most essential parts of inference, it made sense to have lightweight fully connected layers.

Although VGGish with a fully-connected network classifier was too heavy to be loaded onto our edge devices, it was by no means slow in its inference. On average, inference of a single audio input (.wav file) took no longer than 2 seconds. After the inference step, a classification result would then be sent to the cloud MySQL database to be joined with the edge classification data.

Develop a Distress-Centric Acoustic Model

Unfortunately, our team was unable to achieve this milestone. However, the intended design and utility of the model will be discussed below.

Our team identified the class-pair of "glass shatter" and "human shout" as a good candidate for a joint-classifier that signaled distress. The goal was to tune the YAMNet or VGGish models for only the two classes of interest using both novel recordings that our team generated and by using acoustic data augmentation techniques such as shifting start/stop times, pitch manipulation, noise injection, and time dilation/contraction.

The augmented training data were to be generated using `GStreamer` ingest and `ffmeg` sampling and then manipulated with the Python `librosa` library [1].

The weights generated from training with the novel and augmented data would be used to fine tune the model used to classify our ontologies of interest.

Deploying a Scalable Architecture to Support Edge Devices

After evaluating and testing the models with sample data, we began building the various services and pipelines for our edge devices [2] and the cloud [3].

How did we get the audio in?

The first part was to create a pipeline with `GStreamer` or another library to ingest raw audio from our devices to serve to our models automatically. We could not find an open source library similar to `OpenCV` to take raw audio segments produced from `GStreamer` into our edge ingestion service. For that reason, we decided to create local audio files (.wav) recorded directly from our Jetson devices via `GStreamer`.

To keep the ingest service running and not spend a lot of time recording audio, we added audio files from the ESC-50 dataset, which is a labeled collection of 2000 environmental audio recordings suitable for benchmarking methods of environmental sound classification [4]. The dataset consists of 5-second-long recordings organized into 50 semantical classes (with 40 examples per class). This dataset worked perfectly to put our service into production.

The audio ingest service iterates over the audio files folder and sends the audio inference via MQTT. To simulate "real" conditions, a bash script random published to an edge MQTT broker topic, and the service randomly selected a sample to ingest.

1 The `librosa` package is an audio and music analysis for Python. https://librosa.org/doc/latest/index.html
2 Our team's edge devices were a mixture of several model variations of NVIDIA Jetson embedded computing devices: NX, TX2, Nano 4GB and Nano 2GB.
3 Our team used a wide array of Amazon Web Services (AWS) cloud services to construct our cloud pipelines and services.
4 ESC-50 (Environmental Sound Classification 50) Dataset. https://docs.activeloop.ai/datasets/esc-50-dataset

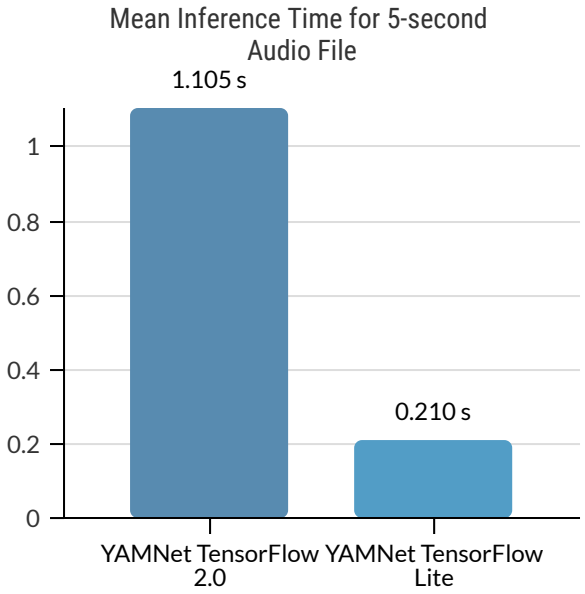
Deploying a Scalable Architecture to Support Edge Devices

How are we going to run inference on our audio?

Of the two models we evaluated for edge inference, YAMNet was most promising over VGGish for two key reasons: VGGish was developed for `x86` architectures, and YAMNet is a much smaller model and had quantized variants available.

The initial implementation of YAMNet worked well on certain Jetson models, but after evaluating is on the Jetson's, inference times were long and on the lowest compute model (Nano 2GB), the model consumed all memory resources making rapid inference useless on those devices.

The original YAMNet model was developed on using TensorFlow 2.0. A TensorFlow Lite model was also developed, specifically for mobile and edge applications, and it reduced our inference time five-fold.



What are we going to do with this inference?

To make use of these inference generated by our Jetsons, we designed a basic MySQL database on the cloud to store tabular data that could be used in other analysis or to generate novel classical machine learning clustering or forecasting models. Our team is located in four different locations around the world with four different Jetson model devices.

The data from the edge is published to the cloud via MQTT services. The MySQL database service runs on the same AWS Elastic Kubernetes Service (EKS) Nodegroup as the MQTT broker receiving data from the Jetson devices. This is the database schema and some sample data for every inference generated by the Jetson device using the YAMNet TensorFlow Lite model:

Deploying a Scalable Architecture to Support Edge Devices

```
CREATE TABLE audio_ingestion (
    inference TEXT,
    second_inference TEXT,
    inference_YamNet_time FLOAT(53),
    lat FLOAT(53),
    long FLOAT(53),
    sensor_current_time TEXT,
    datetime TEXT
)
```

```
{"inference": "Alarm", "second_inference": "Music", "inference_YamNet_time": 0.21558, "lat": 18.5, "long": -69.9,
"sensor_current_time": "03:38:10", "datetime": "UTC"} inference audio
{"inference": "Silence", "second_inference": "Chirp tone", "inference_YamNet_time": 0.20491, "lat": 18.5, "long": -69.9,
"sensor_current_time": "03:38:11", "datetime": "UTC"} inference audio
```

From the Edge to the Cloud

To ensure ease of service deployment to both the edge and cloud and to make the most of the scalability offered by those services, our team designed and implemented an end-to-end Kubernetes-based containerized pipeline.

At the Edge

The edge services were designed to facilitate fast inference and ensure edge clients are durable. All services were containerized using Docker and stored in a public AWS Elastic Container Repository (ECR). The containers are managed using K3S, a lightweight variant of Kubernetes. This implementation allowed for the use of bash scripts to ensure all dependencies, keys, and services could be deployed to every edge device to ensure a similar build.

This build resiliency also extended to our cloud services, as our team used an external secrets management API called Doppler to facilitate AWS key management so edge devices could locate the AWS Elastic Load Balancer (ELB) public IP whenever the AWS EKS cluster was taken down and brought back up.

Our edge devices hosted five (5) services: an MQTT broker, logger, and relay, an inference service, and an ingestion service. As stated earlier, we were unable to get our ingestion service functional, but the containers and deployment scripts were built to add that service when it is developed. Our relay service added some additional dynamic content to the MQTT topics before transmitting the binarized audio and inference JSONs. When building the k3s pod, environmental variables from each Jetson are added to the MQTT topic ontology to provide serial number and locality data to the cloud MQTT broker for additional MySQL field data for each inference.

Deploying a Scalable Architecture to Support Edge Devices

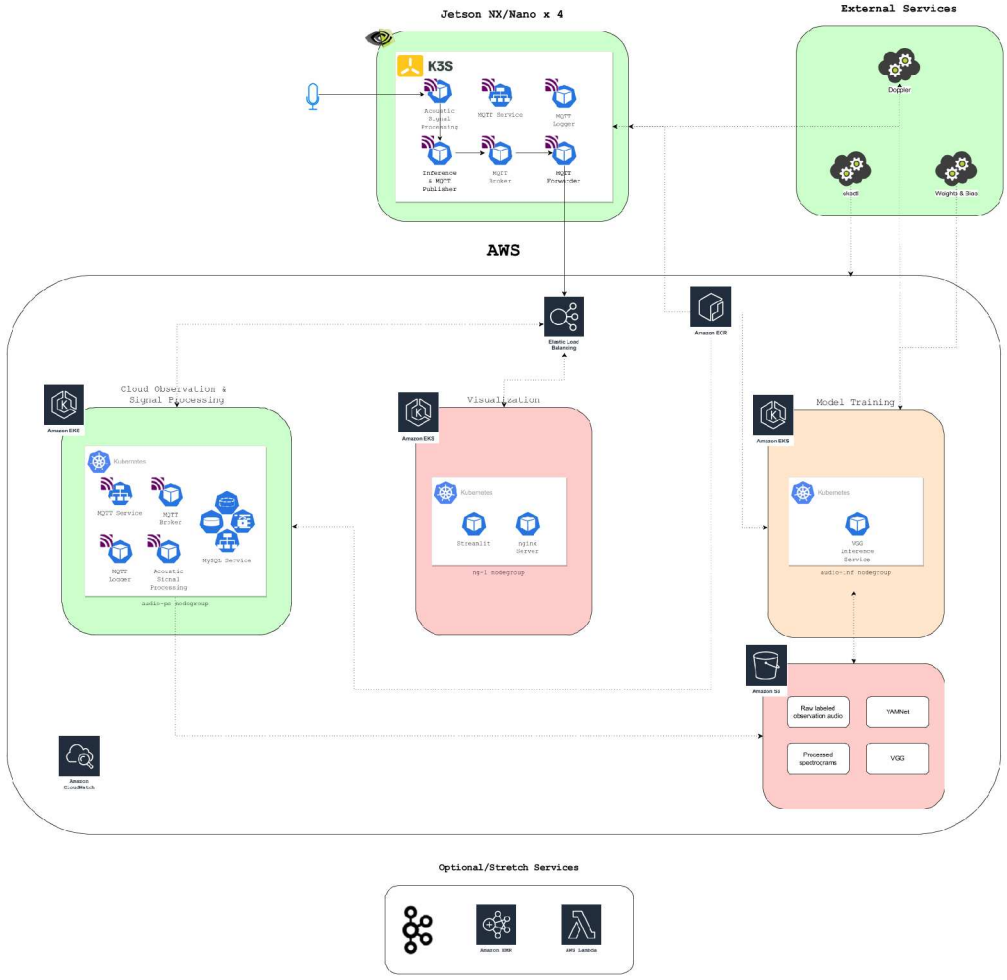
In the Cloud

The cloud services were designed to serve three roles: store tabular inference data in a MySQL database to be used for future modeling, host a web-service for visualization, and provide a cloud-based VGGish inference service for a more robust overall inference service. These capabilities were facilitated by employing AWS EKS. This service allowed for the configuration of Nodegroup clusters that could host each microservice cluster supporting the larger service roles. To configure AWS EKS, we used a CLI tool called `eksctl`.

Gaining Unique Insights from Acoustic Environmental Signals. We did not have time to design or implement our web-server, but the intent was to provide a `Folium`-based map user-interface that used the tabular MySQL data to generate inference points on a map. The distress-pair audio observations would be stored in an AWS S3 bucket, and attached to each map point for a human evaluation of the new class. A Streamlit server was also intended to be used demonstrate basic regression or clustering algorithms using our inference dataset.

System Design

The overall system design is shown below. Services are color-coded according to their level of implementation: green (mostly or fully complete), orange(partial completion), and red (not implemented).



Retrospective and Future Work

Lessons Learned

Working on this project was an incredibly rewarding and challenging experience. Our team explored a variety of new cloud services, explored a new applied deep neural net domain, and experienced some the struggles (joys?) associated with putting production code on a new system and everything breaking.

In retrospect, our team would have benefited from re-evaluating the project scope more often and asking again what question is to be answered and how has what we've learned through the development process may have changed or refined that question's scope. Furthermore, adopting a fail faster and sooner (CI/CD) workflow would have identified critical path failures earlier and may have facilitated a more complete final deliverable.

Future Work

There were several components of our project that were not fully implement. Of those, developing a more complete ingestion and detector service for distress-signaling acoustic combinations would be a priority. That service would also benefit from using standard signal processing and filtering techniques to improve SNR and audio quality to better align with the trained models.

Another goal would be to further quantize the model using TensorRT or similar methods and attempt to run inference on even less robust edge devices, such as microcontrollers and FPGAs.