EECS 605 Project Report
AWS Based Audio Classifier

Akshay Tondak (akshayt@umich.edu), University of Michigan, Ann Arbor

I. Problem Statement

Audio classification is a difficult problem in the domain of deep learning for the reason that types of audio samples and amount of information in a single audio sample could be large. In contrast, in the field of vision, deep learning has seen a large number of cutting edge research published due to the efficiency of computation continually improving with modern GPUs. Computer Vision has fairly established architectures like ResNet, YOLO, GANs etc. which are able to perform deep learning tasks of object detection and image segmentation efficiently with high accuracy. This projects aims to use pre-trained computer vision architecture (ResNet) in a completely different problem statement of audio classification and more importantly demo it through a live website using AWS' cloud infrastructure running the deep learning model inference in the backend.

II. Training

The training phase for this project utilizes a third party service called clearML to do training phase debugging. ClearML is a ML/DL development and production suite and this project used it as an Orchestration, Automation and Pipelines solution for training. The total training time for the model was **3 hours**. The model was trained using the open source library **pytorch** and was done on Google Colab to utilize google's GPUs to fasten up the process. For training, the type of loss used for error minimization was the cross entropy loss.

$$NLL(y) = -\log(p(y))$$

$$\min_{\theta} \sum_{y} -\log(p(y;\theta))$$

$$\max_{\theta} \prod_{y} p(y;\theta)$$

$$NLL = No - log - loss \tag{1}$$

1 shows a a visualization of how the training loss decreased gradually during the training time. The model reached the saturation accuracy of **75%** at which point the training was stopped and model exported.

Fig. 1: Loss deprecation (Loss vs Time)

I shows the parameters used to train the model. A batch size of 8 was used. Which means the networks is passed with 8 samples simultaneously while training. The number of epochs were 5 and the training accuracy saturated at around 75% and the test accuracy was also close to 75%

Model Parameters	
Parameters	Values
Batch size	8
Base learning Rate	0.005
Dropout	0.3
Number of Epochs	3
Number of Mel filters	64
Sample frequency	22500

TABLE I: Model Parameters.

III. Dataset

The project uses the dataset Ooi et al. (2021). The dataset models 10 different classes and the number of audio samples per class are visualized in 2

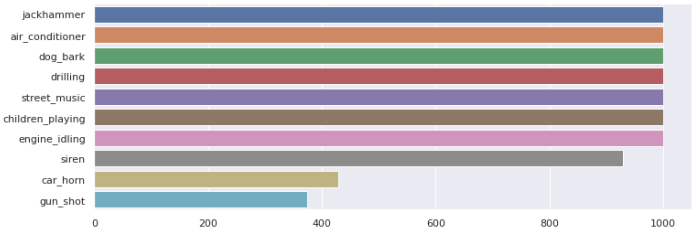

Fig. 2: Sample distribution (Classes vs Number of samples)

The audio files were in .wav format and and on average 5 seconds in length. In addition to the sound excerpts, a CSV file containing metadata about each excerpt is also provided. This csv also contains meta-data information about every audio file in the dataset. Some sound files are different slices of a single audio file thus multiple sounds might resemble each other.

IV. Transfer Learning

Transfer learning is a technique introduced to overcome the tedious and computationally expensive task of learning high number of parameters from the beginning and also providing a concession on the size of dataset required to get a well-trained model with decent detection accuracy. Since Transfer Learning uses pre-trained weights, to learn a good model, smaller dataset are sufficient since the pre-trained model would have already learnt some representation of the image space and only minor modifications might be required. in regards to optimization, initialization of weights of a model using pre-trained is effective. Traditional transfer learning methods involve detaching and discarding the last couple of layers of the pre-trained model and using only the remaining part to perform transfer learning on a newer dataset. Zhuang et al. (2019) provide a detailed view of transfer learning and this project uses transfer learning to avoid training from scratch.

V. Melspectrogram

The mel scale is a scale of pitches that human hearing generally perceives to be equidistant from each other. The name mel derives from melody and indicates that the scale is based on the comparison between pitches.

The mel spectrogram remaps the values in hertz to the mel scale. Mel spectrograms are better suited for applications that need to model human hearing perception. Mel spectrogram data is also suited for use in audio classification applications. A mel spectrogram differs from a linearly scaled audio spectrogram in two ways. A mel spectrogram logarithmically renders frequencies above a certain threshold (the corner frequency). For example, in the linearly scaled spectrogram, the vertical space between 1,000 and 2,000Hz is half of the vertical space between 2,000Hz and 4,000Hz. In the mel spectrogram, the space between those ranges is approximately the same. This scaling is analogous to human hearing, where we find it easier to distinguish between similar low frequency sounds than similar high frequency sounds. A direct conversion from frequency to mel scale uses the formula shown in 2.

$$m = 2959log_{10}(1 + f/70) \tag{2}$$

A mel spectrogram computes its output by multiplying frequency-domain values by a filter bank.

VI. AWS Pipeline and Architecture

3 gives a broader view of how the AWS architecture looks like for the project. The interface to everything is the s3 bucket which stored the model.onnx and also where the audio files are sent for inference. On addition of any data on to the bucket, the lambda function is triggered which does the audio prepossessing and inferencing. The lambda function is also responsible for sending out the graph and mel-spectrogram converted image to the front-end website. The graph for probability distribution is generated through python's MatplotLib and sent over HTTPs using the REST API. The Lambda compute is used with configurations of **1024MB** RAM and **10s timeout**. The result.txt contains textual form of probability distribution and is also sent over HTTPs using the REST APi.

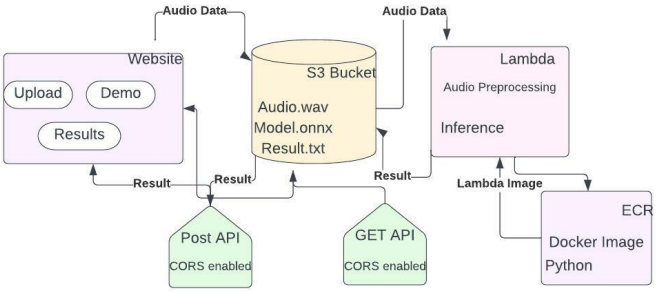

Fig. 3: AWS Architecture

VII. Website

4 shows a sample image of how the final website looks. It has two options of uploading a custom button as well as picking from a fixed set of sample audios which have been pre-stored in the s3 bucket. On loading, the front-end code queries the bucket through a REST API and collects all the files present in the bucket for demo. The user can upload a custom .wav file or selected from a pre-fed file present in the bucket to use the app. The main languages of implementation for the front-end code are HTML and Javascript. For styling the page, CSS is also utilized. For both front-end and back-end implementation, stevesl's implementations were referred to.

VIII. Challenges

- Biggest bottleneck in the project was deploying the lambda code. Project's implementation of pre-processing audio samples and convert them into a spectrogram required some heavy libraries like torchaudio and librosa.

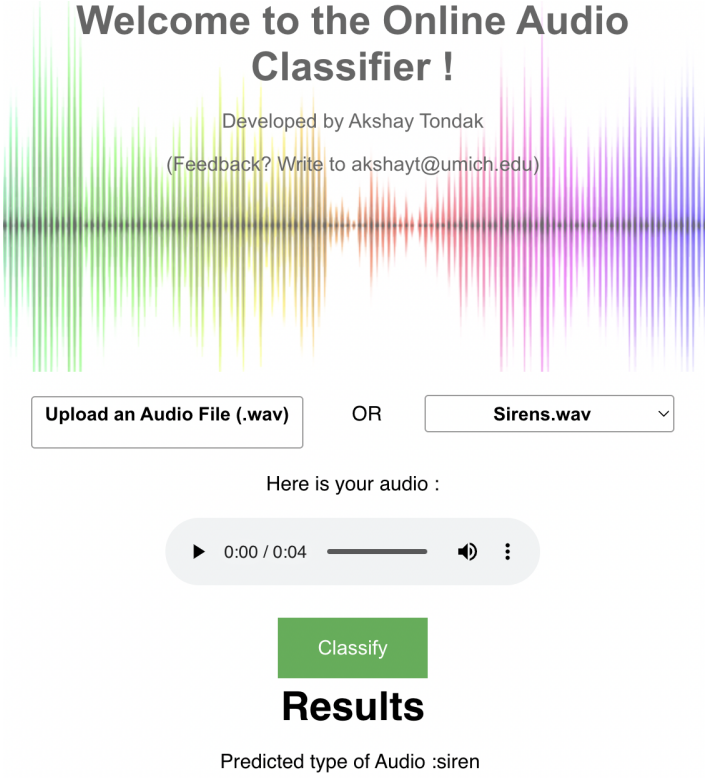

Fig. 4: Demo Website

- Initially the project was coded in pytorch using torchvision and torch audio. Later due to size limits and to keep the cloud architecture light, librosa was chosen to save on size.
- Torch and all dependencies were more than 4 GB. In contrast, the librosa library with all its dependencies was restricted to 800 MB.
- AWS' ECR containerization was utilized to create a docker image of all python libraries and its dependencies.
- ECR posed an altogether different challenge as librosa had OS dependencies, understanding which was also a challenge.
- This was overcome by building the audio based dependency from scratch in the docker environment itself and pushing the whole image to ECR and using that image as the lambda function's source.

IX. Correct and Incorrect classification

5 shows an example of correct classification. The actual class here was "car horn" and the model correctly predicts "car horn as the predicted class".

6 shows an example of incorrect classification. The reason this is incorrectly classified is that the length of audio is enough to bring out information. The image shows that most of the image is empty. Thus a car horn is predicted to be a sound of drilling.

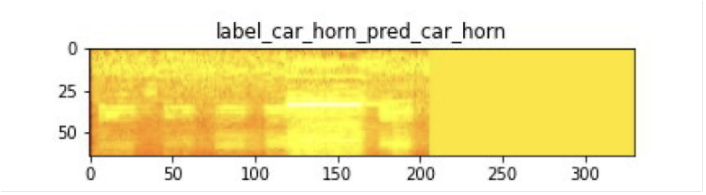

Fig. 5: Correct Classification

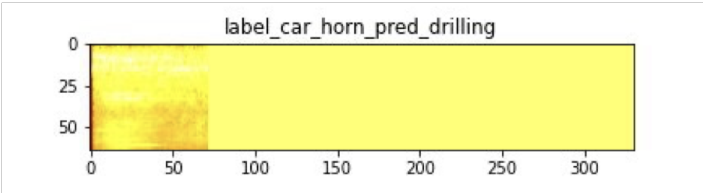

Fig. 6: Incorrect Classification

X. Key Learnings

- **Docker** is a platform as a service which i utilized as an OS level virtualization to containerize and deploy preprocessing code diectly to ECR.
- **Heroku** is a cloud platform as a service which allows us to have a web application without going through the hassles of web development. This was really useful as the development was done minimally in HTML and JS and directly deployed to heroku.
- **AWS** is a one stop solution to all the cloud needs. FOr this project, AWS was the central key where everything is hosted. This project utilized more than one AWS service namely ECR, lambda, s3 etc.
- **ONNX** is a format of packaging and deploying pretrained models. Model exporting and inference for this project was done through onnx.
- **ClearML** is a debugging service which was utilized in this project to pipeline the model learning process.

References

Kenneth Ooi, Karn N. Watcharasupat, Santi Peksi, Furi Andi Karnapi, Zhen-Ting Ong, Danny Chua, Hui-Wen Leow, Li-Long Kwok, Xin-Lei Ng, Zhen-Ann Loh, and Woon-Seng Gan. A strongly-labelled polyphonic dataset of urban sounds with spatiotemporal context, 2021. URL https://arxiv.org/abs/2111.02006.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2019. URL https://arxiv.org/abs/1911.02685.