12-2020

Efficient Edge Analytics: Addressing Cyber-Physical MASINT with Machine Learning on Audio at the Edge

David Elliott

Efficient Edge Analytics: Addressing Cyber-Physical MASINT with Machine Learning on Audio at the Edge

by

David Elliott

Master of Science in Computer Engineering
Computer Engineering and Science
Florida Institute of Technology
2018

A dissertation
submitted to the College of Engineering and Sciences at
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Engineering

Melbourne, Florida
December, 2020

We the undersigned committee
hereby approve the attached dissertation

Efficient Edge Analytics: Addressing Cyber-Physical MASINT with Machine Learning on Audio at the Edge by David Elliott

---

Carlos E. Otero, Ph.D.
Associate Professor
Computer Engineering and Sciences
Committee Chair

Munevver M. Subasi, Ph.D.
Associate Professor
Mathematical Sciences
Outside Committee Member

Josko Zec, Ph.D.
Associate Professor
Computer Engineering and Sciences
Committee Member

Samuel P. Kozaitis, Ph.D.
Professor
Computer Engineering and Sciences
Committee Member

Philip Bernhard, Ph.D.
Associate Professor and Department Head
Computer Engineering and Sciences

ABSTRACT

Title:

Efficient Edge Analytics: Addressing Cyber-Physical MASINT with Machine Learning on Audio at the Edge

Author: David Elliott

Major Advisor: Carlos E. Otero, Ph.D.

With the growth of the Internet of Things and the rise of Big Data, data processing and machine learning applications are being moved to cheap and low size, weight, and power (SWaP) devices at the edge, often in the form of mobile phones, embedded systems, or microcontrollers. The field of Cyber-Physical Measurements and Signature Intelligence (MASINT) makes use of these devices to analyze and exploit data in ways not otherwise possible, which results in increased data quality, increased security, and decreased bandwidth. However, methods to train and deploy models at the edge are limited, and models with sufficient accuracy are often too large for the edge device. Therefore, there is a clear need for techniques to create efficient AI/ML at the edge. This work presents training techniques for audio models in the field of environmental sound classification at the edge. Specifically, we design and train CNNs, first, to classify office sounds in audio clips. Then, we design and train Transformers to classify office sounds in audio clips. Results show that a BERT-based Transformer, trained on Mel spectrograms, can outperform a CNN using 99.85% fewer parameters. Our final model outperforms the state-of-the-art MFCC-based CNN on the office sounds dataset, using just over 6,000 parameters – small enough to run on a microcontroller.

Table of Contents

List of Figures

Acknowledgements

This work was supported by the Air Force Research Lab (AFRL). I would like to thank the staff at the AFRL for their support.

I am thankful to my colleagues, Steven Wyatt and Evan Martino, for their incredible support over the course of my research. I could not have done it without you both, and, even if I could have, it wouldn't have been nearly as fun.

I would like to thank my advisor, Dr. Carlos Otero. Never have I met someone who so genuinely and selflessly cares about their students. He was always available, always encouraging, and always pushing me to do my best. The six years we've been together have been life-changing, and I will always be grateful for his impact in my life.

Finally, I'd like to thank my family, my friends, and my girlfriend, for believing in me at all times.

Chapter 1

CNNs for Environmental Sound Classification at the Edge

1.1 Abstract

Recent advances in embedded system technology have created opportunities for alleviating or eliminating common Big Data problems, by providing the resources necessary to perform AI/ML algorithms on-board edge devices. This has led to the emergence of a sub-discipline of Measurements and Signal Intelligence (MASINT) known as Cyber-Physical MASINT, wherein analysts can receive and exploit data directly from cyber-physical devices, and execute algorithms on-board, without the need to transfer to cloud servers. This type of edge analytics can decrease latency, improve security, decrease the amount of data transferred out of the device, and increase the quality of the data being transferred. With this motivation, we approach the task of environmental sound classification, a task which has seen a substantial amount of research in recent years, but which has had very limited implementation at the edge. In this work, we

design and deploy an application on a mobile device to perform event detection and sound classification using a novel ensemble of deep neural networks optimized for a mobile environment, capable of classifying six common office sounds with high accuracy and low latency. We provide an accuracy and performance analysis at varying levels of optimization.

1.2 Introduction

A major obstacle in edge analytics stems from the resource-constrained nature of embedded devices. These devices are traditionally limited in memory and computing power, which pose a problem for performing inference on trained models [2]. Trivial issues that pose no problem on personal or cloud computing platforms, e.g., floating-point data types needed to represent network weights, pose significant problems in embedded devices, given their limited memory. These problems are exacerbated by the complex and fast-changing landscape for AI/ML and signal processing, in which new statistical and mathematical algorithms, prediction techniques, and modeling methods, as well as multidisciplinary approaches to data collection and analysis all need to be supported on heterogeneous edge devices with different architectures, sensor payloads, operating systems, and build systems [3]. For these reasons, there is a clear need for research and development to create a framework and tools to support efficient development and rapid deployment of edge analytics applications that leverage on-board sensors and provide actionable intelligence to support mission needs.

There are a wide variety of tasks in edge analytics applications [4]. In this paper, we focus specifically on the application of Environmental Sound Classification (ESC) to recognizing office sounds, which falls under a type of proactive edge analytics known as

"stream IoT analytics," with no real-time requirements [4]. Office sounds, in particular, were chosen due to the ease of data collection and the availability of real-world sounds for rapid system tests.

ESC at the edge is in the domain of Cyber-Physical MASINT, which involves the phenomena transmitted through cyber-physical devices and their interconnected data networks [5]. By leveraging the on-board microphone, we locally monitor and classify events in a raw audio stream, which were previously unexploited. We use event detection and feature extraction algorithms before inputting the selected data to an optimized model for inference. Although not explored in detail here, this work may be used in a similar way to [6], where physical interactions are inferred through effects on a digital system. Extraction of information in this way, using the induced effect of the physical world through the cyber-domain, is known as cyber physical sensing [6].

Typically, data will be retrieved from sensors, the data will be sent over the network to a server, and the server will run a machine learning model to perform a prediction based on the data [4]. Feature extraction usually takes place on the server as well. Recent advances in mobile device computational capabilities, however, mean that more computation can be performed at the edge. This has the effect of reducing the amount of data that must be communicated over the network, improving the scalability of large IoT systems. This also has important implications in secure environments, in which data captured at the edge may be too sensitive to transfer for analysis. A thorough analysis of deep learning capabilities on more than 200 Android devices and chipsets was performed by Ignatov *et al.* [7], where it was found that most computer vision tasks can be performed on a mobile device, albeit with latency, memory, and power considerations. Some tasks were met with difficulty, especially those using larger models like Inception-V3 [8], so careful design choices must still be made to ensure successful

deployment to mobile. We note that additional considerations must be taken into account when deploying to embedded devices and generic microcontrollers, which are often far more resource-constrained than mobile phones [9, 10]. The machine learning models in this work are not optimized for extremely low-power embedded devices.

The work by Ignatov *et al.* also brings to light a technology known as TensorFlow Lite[1], which was introduced in 2017 by Google to address some of the difficulties with performing machine learning on embedded devices. TensorFlow Lite allows the user to optimize models based on the characteristics of the deployment environment. For large models, it is possible to reduce the size through quantization, which converts weights from floating-points to 8-bit integers, at the cost of a reduction in accuracy. It is possible to minimize this reduction with quantization-aware training.

In this work, we design a machine learning ensemble made of two separate models, and we build upon the progress made in previous efforts. Many ESC works focus on finding the optimal features to extract from an audio signal, in order to obtain the best classification accuracy. Mitrovic *et al.* [11] organized the types of feature extraction into six broad categories: temporal domain, frequency domain, cepstral domain, modulation frequency domain, phase space, and eigen domain. Features that have been used to date include Mel Frequency Cepstral Coefficients (MFCCs) [12], log Mel-spectrogram [13], pitch, energy, zero-crossing rate, and mean-crossing rate [14]. Tak *et al.* [15] achieved state of the art accuracy with phase encoded filterbank energies. Agrawal *et al.* [16] determined that Teager Energy Operator-based Gammatone features outperform Mel filterbank energies. To combat noisy signals, Mogi and Kasai [17] proposed the use of Independent Component Analysis and Matching Pursuit, a method

[1] `https://www.tensorflow.org/lite`

to extract features in the time domain, rather than the frequency domain. With reference to [17], we note the assumption in our work that noise in an office environment will be minimal. A survey was performed in 2014 by Chachada and Kuo [18] that enumerated the features used in literature, with comparisons between each, but no more recent survey has been found.

In constrast to improving the features, other works focus on better machine learning architectures to improve classification accuracy. Zhang *et al.* [19] obtain state of the art accuracy by using both a temporal and channel attention mechanism over CNN feature maps, fusing the two results by averaging their probabilities. Although we do not use this architecture, this is similar to our approach, in which we ensemble the outputs of two models, using an additional classifier instead of simple averaging. In [20], Zhang *et al.* use a dilated CNN and LeakyReLU activation functions to outperform comparable CNN-based architectures on the ESC task. A CNN-based classifier with parallel convolutional layers of different sizes is used by Chong *et al.* [21], along with multi-level feature aggregation, to obtain state-of-the-art results on ESC-10 [22], ESC-50 [22], and UrbanSound8k [23]. However, none of these works have focused on building a model for deployment onto a mobile device. To account for the limitations of a mobile device, we chose modest architectures with simple features that still had comparable accuracy to the state-of-the-art, described in more detail in Section 1.4. Implementing more advanced feature extraction, and training with additional architectures is a task for future work.

1.3 Data

Finding datasets for office-sound-specific ESC was a challenging task. Few ESC datasets exist, and those that do exist are tuned toward specific types of problems, such as DCASE's noisy labels or rare events datasets [24]. The popular ESC-50 dataset contains an excellent sampling of sounds from different classes, but suffers from a limited amount of data, at only 40 clips per class [22]. Since our goal was to create a machine learning system that behaves well in the real world, under a variety of circumstances and using the microphone on a mobile phone, we found it necessary to use as much data as possible. As a result, we chose to combine the datasets we found, and to collect our own data as necessary. We selected six office sound classes that we determined to occur often in an office environment:

- Snap
- Knock
- Laugh
- Cough
- Keys Jangling
- Keyboard Typing

1.3.1 Data Collection

Our dataset is composed of data from two open datasets, supplemented with data we collected ourselves. We used the data from DCASE 2018, Task 2 [25], which is based on the FreeSound dataset [26]. The FreeSound dataset is a large, open audio dataset,

with crowd-sourced annotations. The dataset was collected automatically based on tags and categories from the FreeSound website[2], and therefore does not have very high accuracy. The accuracy for each class was generally 60-80% for our classes, with our dataset of office sounds being a subset of the DCASE dataset. This resulted in our office sound classification task being considered a "weak classification task," as discussed in literature [27]. The ESC-50 dataset [22] also had data that fit into our office sound categories, so that was included as well. Although a substantial amount of data is available from only those two datasets, we found that, based on initial tests, additional data was required.

We supplemented these weakly-labeled datasets with manually collected data of our own, which were targeted at those aspects of the classes which were poorly represented or missing altogether. This kind of data supplementing emphasizes the overall sentiment of modern-day machine learning: a machine learning model is only as good as its data. Sources for online collection included YouTube [3] and SoundBible [4]. For some classes, it was possible to use a script to automate the search and download of desired clips. Manual editing of the sound files were needed for clips that were excessively long or had a large number of alternate sounds present. Table 1.2 shows the statistics for our dataset, where the total time for all 1608 audio clips is 2.8 hours. Most of the data came from DCASE (68.8%), though that data size was increased roughly 1.5x by adding ESC-50 and our manual collection.

---

[2] `https://freesound.org/`

[3] `https://www.youtube.com/`

[4] `http://soundbible.com/`

Table 1.1: The number of audio clips contributed by each dataset.

	DCASE	**ESC-50**	**Collected**	**Combined**
Knock	279	40	88	407
Laugh	290	40	68	398
Keyboard Typing	119	40	43	202
Cough	243	40	3	286
Keys Jangling	99	40	7	146
Snap	77	40	52	169
Total	**1107**	**240**	**261**	**1608**

1.3.2 Data Processing

Processing training data in such a way that it can be effectively used by the training algorithm is important in any machine learning system. There are often many choices that must be made during this step that result in time and accuracy trade-offs later on. All audio files are sampled at 44100 Hz, in mono.

The first trade-off we must consider is the size of a window for the classification model to view. Modern neural networks are capable of processing a variable-length audio file [27], but with the limitation that the audio file cannot exceed a certain length, or else run out of memory. Since we do not want to limit the recording length on the mobile device, we chose to extract fixed-length windows from the audio files on which we train. The method for extracting these windows can be seen in Figure 1.1, using a sound clip of a cough as an example. A window size of one second and a hop length of 0.5 seconds was chosen, resulting in overlapping windows. The last window was padded with zeros to meet the required window size. Once the windows had been obtained, each window was added to the dataset with the label of the audio file from which it came.

The result of performing window extraction in this manner was that the network

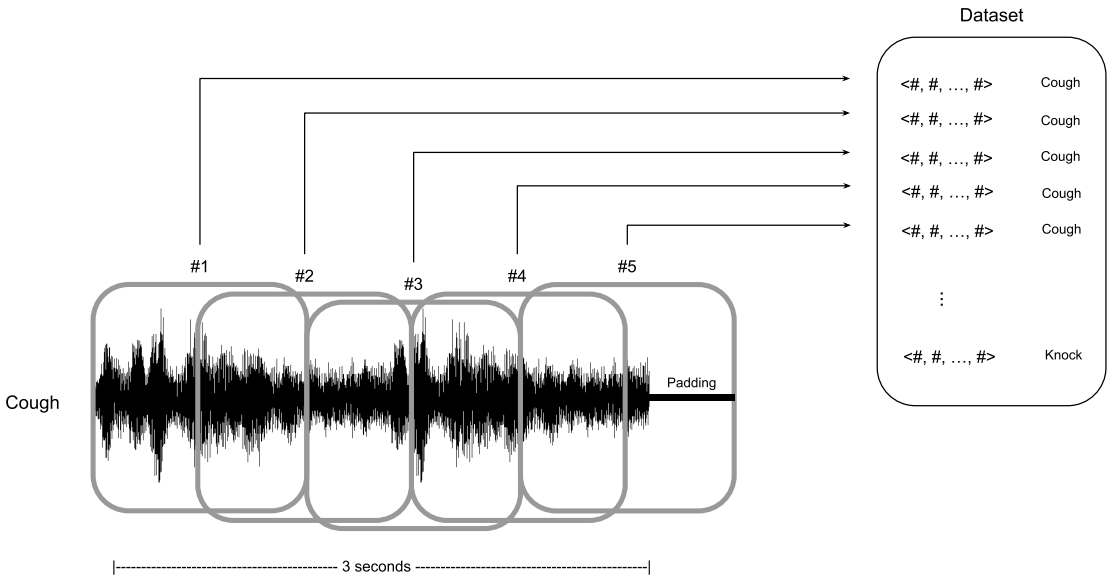

Figure 1.1: A single audio file is divided into overlapping segments of one second windows. The final segment is padded with zeros, as needed. Each segment is placed in the dataset with the same label as the file from which it was taken.

was able to have a sufficiently large view of a sound when learning to classify it. Very few sounds had lengths greater than one second, and, for those that do, they generally could be understood and identified within the first second. We also observed an improvement in classification accuracy when increasing the window size from small values (20 - 200 ms) to larger values (0.8 - 1.0 s), and a slight improvement when increasing it from 0.8 to 1.0 seconds.

1.4 Model

Feature extraction plays a large part in a machine learning model's ability to learn. If features are too nondescript, the machine learning model may not reach the sufficient level of abstraction to learn high-level of features for classification. An example of this is training a text classification algorithm on characters in a document. The algorithm

must first gain an understanding of words before being able to classify on words. To train on words themselves often results in the model achieving a better understanding of the task in a shorter period of time, with less data. However, a trade-off occurs here, in that by using words instead of characters, the algorithm misses out on the ability to generalize to words that it has not seen. This is a difficulty that requires careful handling in natural language processing tasks even today [28].

A similar challenge faces us with audio classification. We approach the problem by using both kinds of feature extraction: nondescript, "raw" data, as well as processed, more abstract data. The forms that these features come in is amplitudes, the raw data, and Mel-Frequency Cepstral Coefficients (MFCCs), the more abstract data.

1.4.1 Feature Extraction

For amplitude feature extraction, the only processing that is performed is standard scoring normalization on each window in the dataset. This ensures that the mobile device is able to perform classification without much delay due to feature extraction.

For MFCC feature extraction, each one-second window passes through the following steps, using librosa's MFCC library [29]:

1. The signal is normalized by standard scoring normalization.
2. The Mel spectrogram is computed on the audio signal.
3. The spectrogram is converted from power to dB units.
4. The discrete cosine transform (DCT-II) of the spectrogram is taken.
5. A number, N, of values are taken from the resulting spectrum. These are the MFCCs of the signal.

In our MFCC processing, we used a sampling rate of 44100 Hz, 128 MFCCs, a FFT window length of 1024, a hop length of 512, and 128 Mel bands. Note that this same feature extraction occurs on the embedded device as well after deployment.

1.4.2 Model Architectures

We created three separate networks: one designed for learning from raw amplitudes, one designed for learning from MFCCs, and an ensemble of both. The amplitude and MFCC models are based heavily on architectures found in [30] and [27], respectively. We used Keras [31] in order to create and train our models.

1.4.2.1 Amplitude Model

The architecture for our amplitude model is a deep neural network made up of 11 convolutional layers. There are several important considerations in training a model based on amplitude alone. First, given a one-second signal sampled at 44100 Hz, the network will have 44100 features. In order to gain any useful understanding of a sound, it must be able to sufficiently generalize in order to perform the rather large conversion between amplitude data and sound classes. These considerations imply a deep network. The layers we use are described in the following:

- Layer 1 is a 1D convolutional layer with input shape (44100, 1), 64 filters, window size of 80, stride length of 4, and padding. It is followed by batch normalization, ReLU activation, and a 1D max pooling layer with pool size of 4.

- Layers 2-3 are 1D convolutional layers with 64 filters, window size of 3, stride length of 1, and padding. They are each followed by batch normalization and ReLU activation. Layer 5 ends with max pooling, with pool size of 4.

- Layers 4-5 have the same structure as layers 2-5, but with 128 filters.
- Layers 6-8 have the same structure as layers 6-9, but with 256 filters.
- Layers 9-10 have the same structure as layers 10-13, but with 512 filters, and no max pooling.
- Layer 11 performs global average pooling.
- A final softmax layer converts the features from layer 18 to classifications.

This model has a total of 1.79 million parameters. In training the model, we trained on the common office sounds task with 6 classes. We used an Adam optimizer, categorical cross-entropy loss, and reduced the learning rate by half when the validation accuracy plateaued for 10 epochs, reducing to a minimum of 0.0001. We trained for 200 epochs with batch size 128, saving the model with the highest validation accuracy as we went. This is the same network known as M11 in [30], chosen for the high accuracy it provides in relation to its size; larger networks, such as M18 and M34-res, provide only a slight increase in accuracy.

1.4.2.2 MFCC Model

The architecture used in training a model based on MFCCs is described in the following:

- Layer 1 is a 2D convolutional layer with 16 filters, and input shape (128, 87, 1). 128 is the number of MFCCs, and 87 is the result of $\lceil r * l_{hop} \rceil$, where r is 44100 Hz, and l_{hop} is the hop length used in MFCC extraction. This layer has a window size of 3, ReLU activation, and padding. It is followed by batch normalization.
- Layer 2 is the same as layer 1, but is additionally followed by max pooling with pool size of 2 and stride length of 2.

- Layers 3-4 are the same as layers 1-2, but with 32 filters.
- Layers 5-6 are the same as layers 3-4, but with 64 filters.
- Layers 7-8 are the same as layers 5-6, but with 128 filters.
- Layers 9-10 are the same as layers 7-8, but with 256 filters, and a stride length of 1 during max pooling.
- Layer 11 is the same as layer 9, but with 512 filters. There is no second convolutional layer before the max pooling. Max pooling occurs immediately after layer 11, with pool size 2 and stride length of 2.
- Layer 12 is the same as layer 11, but with 1024 filters.
- Layer 13 is a 2D convolutional layer with the number of filters given by the classification task, a window size of 1, and sigmoid activation. This is functionally the same as a softmax layer, in that it converts the previous features into predictions.
- Layer 14 is a global average pooling layer.

This model has a total of 4.47 million parameters. Training hyperparemeters are the same as in amplitude training.

1.4.2.3 Ensemble Model

An ensemble model was trained on the outputs of the amplitude model and the MFCC model. The model was made up of three dense layers of 50 neurons with dropout of 0.2 and ReLU activation, followed by a single dense layer with softmax activation.

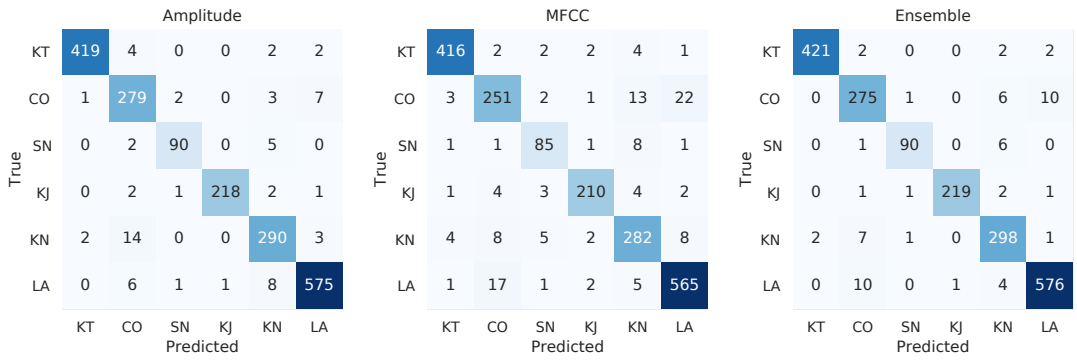

Figure 1.2: Confusion matrices for our models. Evaluation took place using unoptimized TFLite models, against our test set of segmented audio clips. The classes are abbreviated as follows: KT is keyboard typing, CO is coughing, SN is snap, KJ is keys jangling, KN is knock, and LA is laugh.

Table 1.2: Comparison of Optimized Models: Accuracy, Recall, Size, and Latency

	Acc.	**Recall**	**Size**	**Lat.- S9**	**Lat.- S8**
Amp.	96.4%	95.8%	21.6MB	–	–
Amp. Lite	96.4%	95.8%	7.1MB	49.6ms	70.0ms
Amp. Lite Q.	89.4%	83.1%	1.8MB	47.0ms	62.2ms
MFCC	93.2%	91.9%	53.8MB	–	–
MFCC Lite	93.2%	91.9%	17.9MB	57.4ms	77.6ms
MFCC Lite Q.	89.8%	88.5%	4.5MB	40.4ms	55.6ms
Ens.	96.9%	96.2%	25.1MB	–	–
Ens. Lite	96.9%	96.2%	25.0MB	113.0ms	148.4ms
Ens. Lite Q.	94.0%	91.2%	6.3MB	92.2ms	118.6ms

1.4.3 On-Device Event Detection

Our model requires one-second windows of audio in order to be able to predict a class. The naive method to obtain these windows on the device is to extract them via a sliding window approach, as discussed in Section 1.3.2. Doing so, however, would prevent the user from being able to classify multiple sounds in a single recording. Therefore, we run a detection algorithm on the device to find one-second windows on which to perform classification. We do this by moving two fixed windows, a front and a back window,

across the recorded audio with a stride of 0.1 seconds, where the back window is 0.1 seconds and the front window is 0.05 seconds (similar to STA/LTA [32]). The front and back windows are adjacent to each other and non-overlapping, with the front window being farther ahead in time than the back window. For each window, the mean and variance is calculated on the waveform it is viewing. These values are used in Equation 1.1 to find the Bhattacharyya distance [33], where p and q are the back and front windows, respectively, the variance of a window is given by σ^2, and the mean of a window is given by μ.

$$D(p,q) = \frac{1}{4}\ln\left(\frac{1}{4}\left(\frac{\sigma_p^2}{\sigma_q^2} + \frac{\sigma_q^2}{\sigma_p^2} + 2\right)\right) + \frac{1}{4}\left(\frac{(\mu_p - \mu_q)^2}{\sigma_p^2 + \sigma_q^2}\right) \tag{1.1}$$

When the Bhattacharyya distance is above a threshold of 0.4, an event is marked as having taken place. We set a fixed event window with a length of one second to begin one stride length prior to the leading edge of the front window, to ensure that the beginning of the sound is not missed. This one-second event window is then passed to the model for classification, and detection continues from the end of the previously detected event. If an event is immediately detected again, a rule exists to prevent events from overlapping.

1.4.4 Evaluation

We provide an evaluation of our model against our test set, using the unoptimized TFLite models, as shown in Table **??**. We used an 81-9-10 split between the training, validation, and test data, respectively. The accuracy shown in Table **??** reflects model accuracy on the *segmented* test data, rather than the file-level data, due to the fact that recorded audio on the mobile device is only classified in one-second segments.

The latencies are measured on the Samsung Galaxy S9 and S8, using the Snapdragon 845 and 835 respectively. Memory consumption was found to be the same as the size of the models.

We observe that the accuracy of the amplitude model exceeds that of the MFCC model; this was unexpected, given the larger model size and superior features of the MFCC model. The ensemble performs as expected, increasing the total accuracy of the entire system, at the cost of increased model size and inference latency. That this latency is could be further reduced by calculating inferences for the ensemble input in parallel. The size of the model decreases substantially when converted to TFLite, and no accuracy decrease was seen at all. When quantized, the size of each model decreased by nearly a factor of four, owing to the change from 32-bit floating point numbers to 8-bit unsigned integers, and the latency decreased by an average of 18% across all models, with the greatest improvement in the MFCC model, of 30%, and the least in the amplitude model, of 6%. Interestingly, the amplitude model's accuracy and recall decreased by a much wider margin than the MFCC model when quantized. We expect that this is due to the fact that there are many more features in the amplitude model (44100) that may suffer from the decreased precision, compared to the number of features in the MFCC model (11136). The ensemble, however, even when quantized, preserved a high level of accuracy. The confusion matrices of the TFLite models are reported in Figure 1.2.

We also observe that the TFLite amplitude model was more accurate and had lower latency than the quantized ensemble, while being only slightly larger and with a slightly greater amount of energy consumption. For this reason, and due to its relative simplicity, it may be preferred to run ESC using only the amplitude model, out of these three. Though, given additional research into better features and models, and with the

use of quantization-aware training, this decision may easily be changed.

We note that these models have the potential for real-time operation. Given that classification takes less than 200 ms in the worst case, a single classification can be performed once per second without falling behind. Although a energy consumption analysis was unable to be performed, running inference continuously for 24 hours would result in the CPU being active for a total of 1 hour using the MFCC quantized model on the S9, and 3.5 hours using the ensemble on the S8. This may cause the phone to run out of power before the 24 hour period is over. We leave energy-specific performance enhancements to future work.

1.5 Conclusion

Edge analytics has the capacity to fundamentally change the way we process data, bringing with it a vast number of possible applications to improve user experiences, ensure privacy, secure sensitive information, and broaden the capacity for an IoT system to learn and make decisions. In this paper, we described the design of a system to perform environmental sound classification of office sounds on a mobile device, achieving high accuracy and low latency using a dataset of more than 1600 audio clips. Our contributions include analysis of a novel ensemble of mobile-optimized models based on amplitudes and MFCCs for the classification of office sounds, an event detector for embedded use, and an analysis of the impact of TensorFlow Lite optimizations on machine learning models of this kind.

Future work could include adding more feature extraction models to the ensemble model input, retraining models locally on the mobile device as sensor data is consumed, applying transfer learning to improve accuracy, universally converting between different

model architectures for ease of use, and real time classification.

Chapter 2

Tiny Transformers for Environmental Sound Classification at the Edge

2.1 Abstract

With the growth of the Internet of Things and the rise of Big Data, data processing and machine learning applications are being moved to cheap and low size, weight, and power (SWaP) devices at the edge, often in the form of mobile phones, embedded systems, or microcontrollers. The field of Cyber-Physical Measurements and Signature Intelligence (MASINT) makes use of these devices to analyze and exploit data in ways not otherwise possible, which results in increased data quality, increased security, and decreased bandwidth. However, methods to train and deploy models at the edge are limited, and models with sufficient accuracy are often too large for the edge device. Therefore, there is a clear need for techniques to create efficient AI/ML at the edge. This work presents training techniques for audio models in the field of environmental sound classification at the edge. Specifically, we design and train Transformers to clas-

sify office sounds in audio clips. Results show that a BERT-based Transformer, trained on Mel spectrograms, can outperform a CNN using 99.85% fewer parameters. To achieve this result, we first tested several audio feature extraction techniques designed for Transformers, using ESC-50 for evaluation, along with various augmentations. Our final model outperforms the state-of-the-art MFCC-based CNN on the office sounds dataset, using just over 6,000 parameters – small enough to run on a microcontroller.

2.2 Introduction

The field of environmental sound classification (ESC) has been actively researched for many years, with applications in security, surveillance, manufacturing, AVs, and more [34]. In modern days, ESC has important applications to autonomous vehicles (AV), as they can be used to detect sirens, accidents, locations, in-cabin disturbances, and much more. As vehicle-based computational power increases, and algorithms improve, it becomes vital to explore a wide number of options to perform a given machine learning task. For ESC, this means exploring transformers [1] as a possible means to perform ESC at the edge.

Recent work in transformers has profoundly affected the field of natural language processing (NLP), seeing models such as BERT [35], XLNet [36], T5 [37], GPT [28, 38, 39], and BigBird [40] – to name a few – iteratively setting new state-of-the-art for a variety of difficult NLP tasks. In many cases, transformer accuracy exceeds the performance of humans on the same tasks.

Even though most of the highly public work with transformers has been done in NLP, a transformer, which fundamentally is simply a series of self-attention operations stacked on top of one another [1], is a general architecture that can be applied to any

input. OpenAI, an AI research and development company focused on ensuring artificial general intelligence benefits humanity [1], made this clear in several of their recent works. In ImageGPT [41], Chen *et al.* showed how the GPT architecture, which is transformer-based, can be trained in an autoregressive fashion on a wide variety of images, in order to generate realistic image completions and samples. Notably, images were restricted to 64x64 pixels, as a greater amount of pixels requires substantially more compute than was feasible. In Jukebox [42], a transformer is used along with vector-quantized variational autoencoders (VQ-VAEs) [43], in order to generate realistic audio. Also notable is the fact that Dhariwal *et al.* trained the transformer on a highly compressed representation of the audio waveform generated by VQ-VAEs, rather than the raw waveform, and that the outputs from the transformer are not used directly, but are passed through an upsampler first. Even so, the total cost of training the full Jukebox system is in excess of 20,000 V100-days – an enormous cost [42]. More recently, in a paper under review at ICLR 2021, it has been found that, given enough data (hundreds of millions of examples), transformers can exceed even the best CNNs in accuracy [44]. We take this, in addition to the recent trend in larger datasets and more compute, to mean that any work we perform here with small datasets and modest transformers can easily be scaled up at a later date.

The field of audio speech recognition (ASR) has picked up transformers with vigor. Indeed, it is understandable, as applying transformers is straightforward for many approaches to ASR. Often based on encoder-decoders [45], the most obvious use of a transformer is in the decoder of an ASR system, which usually has text as input and output, thus making the application of any state-of-the-art language models, such as

[1] `https://openai.com/`

BERT or its variants, available to it with little adaptation needed. Some work [46] has also made use of a transformer in the encoder as well, thus making the ASR system an end-to-end transformer model. Recent work has seen transformers improve the state-of-the-art for ASR by reducing word error rate by 10% in clean speech, and 5% in more challenging speech [47].

ESC, in some ways, is much simpler than ASR, as it is not concerned about both text and audio, nor does it need to perform fine-grained classification of words or sounds per variable segment of time. However, in other ways, there are more challenges with ESC than ASR. The first major challenge that ESC presents is one of data availability; there is very little data for ESC tasks available, and even the largest and most accurate (DCASE[2][3]) is only tens of thousands of audio files [48]. In addition, there is no agreed-upon standard for what sounds make up ESC. Different ESC datasets often have overlap, and Google's AudioSet [49] provides the largest set of audio labels to date, but many of AudioSet's labels have to do with music or speech, and are not necessarily "environmental". Additionally, ESC datasets are highly heterogeneous, having an extremely broad range of sounds, varying in length, frequency, and intensity. This can increase the difficulty that a machine learning model has in learning the sounds, as they may vary widely from clip to clip.

Environmental sound classification has recently been performed mostly by convolutional neural networks (CNNs) [20,50–55]. These networks vary somewhat in structure, with some being fully convolutional and able to take varying-length input [27], others making use of modified "attention" layers to boost performance [19], and others using

[2] http://dcase.community/

[3] https://www.kaggle.com/c/dcase2018-task1a-leaderboard

deep CNNs [54, 55]. Performance on ESC is typically measured using ESC-50 [22], ESC-10 [22], Urban-8k [23], or DCASE 2016-2018 [56]. Top reported performance on ESC-50, to the best of our knowledge, is 88.50%, by Sharma *et al.* [55], using a spatial attention module and various augmentations. Human accuracy has been measured at 81.3% [22]. We use ESC-50 in this work to allow comparison of our different models in the first stage of our research, in Section 2.6.1.

Unfortunately, state-of-the-art transformers are too large to run on edge devices, such as mobile phones, as they often contain billions of parameters. The largest model of GPT-3 [57] contains 175 billion parameters. When most mobile phones contain several GBs of RAM, any model exceeding one billion parameters (which, when quantized, is 1 billion bytes, or 1 GB) is likely to be inaccessible. When considering that many microcontrollers have only kilobytes of SRAM, it becomes particularly obvious that state-of-the-art transformers are not yet edge-ready. There are also latency and power considerations, as the sheer number of computations by such large models pose a limitation on how quickly a forward pass can be computed using the available processors on the device, and may drain the device's battery very quickly, or exceed power requirements. There are methods to perform knowledge distillation on transformer models, to produce models with slightly reduced accuracy, but substantially smaller in size [58]. However, these methods require the existence of a pretrained model alongside or from which they can learn [59], which does not exist in the field of ESC.

In this work, we attempt to tackle some of these problems. For simplicity, to lower costs, and to improve the iteration time of our development process, we restrict our models to a modest size for most of our analyses. This approach, though motivated by limited resources, is supported by literature, as larger models can always be built later, after promising avenues have been discovered [60].

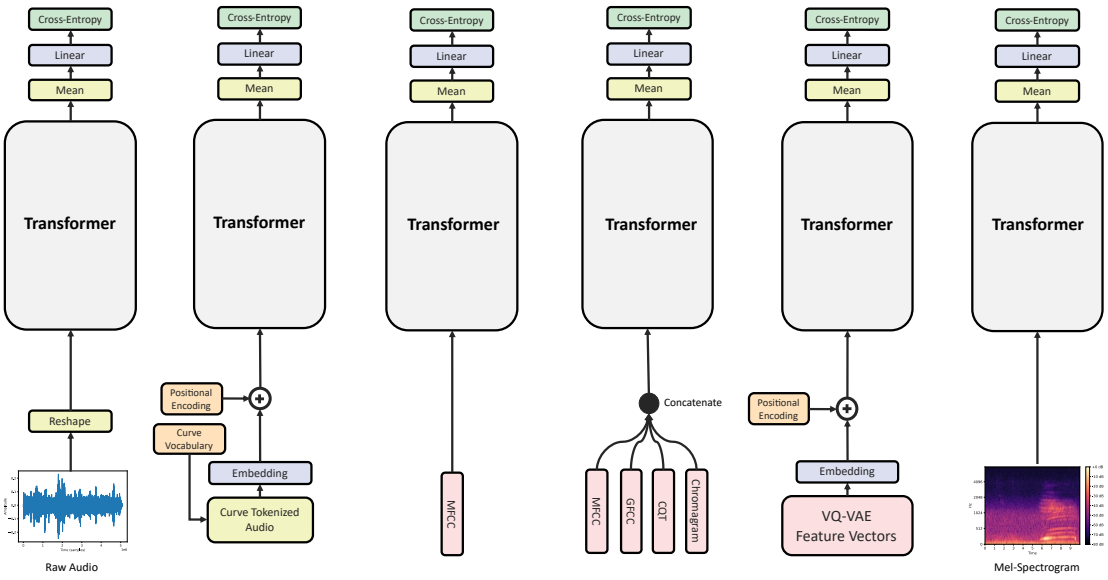

Figure 2.1: We take six unique approaches to training a transformer on ESC. From left to right: raw amplitude reshaping, curve tokenization, MFCC feature extraction, multi-feature extraction, VQ-VAE tokenization, and Mel spectrograms.

Our contributions in this work are as follows:

1. We provide a thorough evaluation of transformer performance on ESC-50 using various audio feature extraction methods.

2. For the most promising feature extraction method, we perform a Bayesian search through the hyperparameter space to find an optimal configuration.

3. Based on the optimal model discovered through the Bayesian search, we train transformers on the Office Sounds dataset, and obtain a new single-model state of the art. We also train a 6,000-parameter model that exceeds the accuracy of a much larger MFCC-based CNN.

4. We test selected models' performance on a mobile phone, and report results.

2.3 Related Work

There have been many attempts to classify environmental sounds accurately, At first many of the attempts used a more algorithmic approach [61], focusing on hand-crafted features and mechanisms to process sound and produce a classification. However, CNNs, which had been shown to perform well on image recognition tasks, eventually became the state of the art on ESC. The current reported state of the art on the ESC-50 dataset is by Sharma *et al.* who used a deep CNN with multiple feature channels (MFCC's, GFCC's, CQT's and Chromagram) and data augmentations as the input to their model [55]. They achieved a score of 97.52% on UrbanSound8K, 95.75% on ESC-10, and 88.50% on ESC-50. (Note that the original publication of Sharma *et al.*'s work included a bug in their code, which resulted in a much higher reported accuracy. That has since been corrected, but has been cited incorrectly at least once [62].) Additionally, a scoreboard has been kept in a GitHub repository [4], but appears to be out of date.

Very little work has been performed with transformers on ESC tasks. Dhariwal et al. [63] used transformers in an auto-regressive manner to generate music (including voices) by training on raw audio. Miyazaki et al. [64] proposed using transformers for sound event detection in a weakly-supervised setting. They found that this approach outperformed the CNN baseline on the DCASE2019 Task4 dataset, but no direct application of transformers to ESC has been found.

In audio, there has been a wide number of feature extraction methods used. Mitrovic *et al.* [11] organized the types of feature extraction into six broad categories: tempo-

[4] `https://github.com/karolpiczak/ESC-50`

ral domain, frequency domain, cepstral domain, modulation frequency domain, phase space, and eigen domain. Features that have been used to date include Mel Frequency Cepstral Coefficients (MFCCs) [12], log Mel-spectrogram [13], pitch, energy, zero-crossing rate, and mean-crossing rate [14]. Tak *et al.* [15] achieved state of the art accuracy with phase encoded filterbank energies. Agrawal *et al.* [16] determined that Teager Energy Operator-based Gammatone features outperform Mel filterbank energies. To combat noisy signals, Mogi and Kasai [17] proposed the use of Independent Component Analysis and Matching Pursuit, a method to extract features in the time domain, rather than the frequency domain. With reference to [17], we note the assumption in our work that noise in an office environment will be minimal. Sharma *et al.* [55] obtain state of the art using MFCC, GFCC, CQT, and Chromagram features. Jukebox [42] was trained to differentiate music and artist styles using features extracted from three different VQ-VAEs, with varying vector lengths for each. A survey was performed in 2014 by Chachada and Kuo [18] that enumerated the features used in literature, with comparisons between each, but no more recent survey has been found. We choose some of the most successful of those feature extraction methods, and attempt some new ones designed specifically for transformers.

2.4 Models

Transformers are neural networks based on the self-attention mechanism, which is an operation on sequences, relating positions within the sequence in order to compute its representation [1]. We emphasize that the attention mechanism operates on a list of sequences, which means that the input to a transformer must be 2-dimensional (excluding batches). In NLP, we want the transformer to operate on sequences of

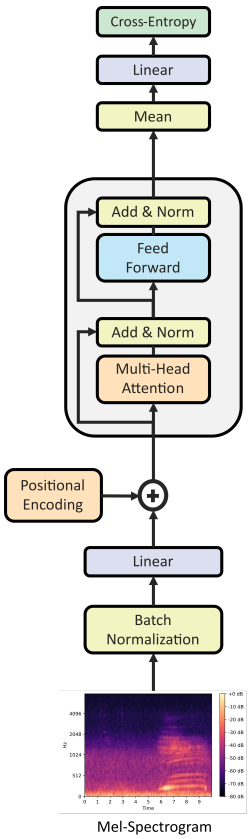

Figure 2.2: The architecture on which our smallest Transformer model is based. Visualization based on the diagram by Vaswani *et al.* [1].

words, characters, or something similar, which we refer to as "tokens". Therefore, in order to meet the 2-dimensional input requirements of the transformer, each token must be converted to a sequence. This is traditionally done using an embedding layer, which takes a token, represented by the integer value of the token's position in a pre-computed vocabulary, and looks up its corresponding vector representation in a matrix. This embedding matrix is able to be learned. The embedding vector length is typically referred to, in transformers, as the "hidden size", or H. The number of tokens is referred to by the length of the input L, also referred to as the sequence length. In this way, H defines the number of dimensions that are used to represent tokens –

where more dimensions typically mean greater learning capacity – and L determines the context length, or window size, of the input.

Audio is represented at an extremely fine-grained level of detail (many samples per second), which poses challenges that NLP does not have to face. For example, the common sampling rate of 44.1 kHz in a 5-second audio clip (the length of an audio clip in ESC-50) results in 220,500 samples. Combine this with the limitations of modern-day transformers, which, with some exceptions, are limited to roughly $H < 2000$ tokens, depending on available hardware, and the task of analyzing audio data becomes quite difficult. There is hope that this will change in the near future, with the creation of linear-scaling models like BigBird [40] proven to have the same learning capacity as BERT, and recent improvements in AI hardware by NVIDIA. But, for the sake of our discussion and analysis, we will assume that we cannot use a transformer sequence length of more than 2048.

This results in the maximum audio window that a transformer can view – in the naïve case, where a single token is a single amplitude – to be 2048 samples, or 0.046 seconds (46 milliseconds). Since sounds in the ESC-50 dataset often last much longer than 46 milliseconds, we must therefore abandon the naïve approach initially. The thought exists that it is possible to downsample the audio to make the 2048 sequence length be able to view a longer length of audio, but in practice this results in substantial information loss below 16 kHz, and reduces model accuracy. We would like our work to be constrained solely by hardware and algorithmic limitations, which have a strong likelihood of improving in the near future, rather than constrained by the information content in downsampled audio clips. Therefore, we assume a sampling rate above the Nyquist rate of 40 kHz for human-detectable audio, and, specifically, use the conventional value of 44.1 kHz in all of our analyses.

All models in this work are based on BERT [35] or AlBERT [65]. The Transformer base structure, whether BERT or AlBERT, does not change in this work. The only alteration performed is to remove positional encodings for some models, which is noted in Figure 2.1 outside of the Transformer base. We note that our base structure in our ESC-50 experiments does not make use of an embedding layer for input tokens, as is customary in language models, and any tokenizations and embeddings that do occur are explicitly called out in Figure 2.1.

We make a change to the transformer design in our second series of experiments on the Office Sounds dataset (Figure 2.2), which allowed the size of the input to be decoupled from the size of the model. In the six models shown in Figure 2.1, the input must be either reshaped or the features must be extracted in the shape required to create a transformer of the desired size. For example, using 128 Mel bands when calculating MFCCs resulted in a transformer that had a maximum hidden size of 128. We remove this dependency in our Office Sounds experiments by adding a mapping layer, as shown in Figure 2.2. The mapping layer is simply a linear layer that takes input of any size and maps it to the size of the transformer. It also provides representational advantages, as this layer is able to be learned, similar to the embedding layer in traditional transformers.

Additionally, in our ESC-50 experiments, we normalize all inputs to a number between 0 and 1 as a preprocessing step, where input is not tokenized. We remove this normalization step in our Office Sounds experiments, in favor of a batch normalization layer [66], which may also have provided representational advantages to the Transformer by being learnable.

2.5 Approach

We divide our approach below into sections on data, feature extraction, data augmentations, models, and model conversion. These methods work together to produce the results in Section 2.6.

2.5.1 Data

We use three datasets in this work, AudioSet [49], ESC-50 [22], and the Office Sounds dataset [67]. AudioSet is a large-scale weakly labeled dataset covering 527 different sound types. The authors provide a balanced and unbalanced version of the dataset; we use the balanced dataset, with some additional balancing that we perform ourselves. Note that in order to train on the audio from this dataset, we had to download the audio from the sources used to compile the balanced dataset. This was an error-prone process, as not all sources from the original AudioSet are still available. More details on the datasets are available in Table 2.1. ESC-50 is a strongly labeled dataset containing 50 different sound types. Each sound category contains 40 sounds, making it a balanced dataset. The Office Sounds dataset is both an unbalanced and weakly labeled dataset, owing to its origins in DCASE, but nearly the same number of audio files as ESC-50, with slightly longer total length, and only 6 labels.

All audio files are converted to wave files, if they are not already formatted as such. We read from each file at a sampling rate of 44100 Hz, in mono.

Table 2.1: Information on the datasets used for training.

	# of files	**# of hours**	**# of audio types**
AudioSet	37948	104.52	527
ESC-50	2000	2.78	50
Office Sounds	1608	2.80	6

2.5.2 Feature Extraction

Feature extraction is a critical part of any machine learning architecture, and especially so for transformers. In fact, some of the critical work that went into making BERT such a success was the use of word pieces, rather than words or characters [35]. In an attempt to discover a feature extraction method that can be of similar use in audio, we attempted several, some of which are well-known methods, others of which we have adapted to our particular use case. The approaches can be seen in Figure 2.1.

2.5.2.1 Amplitude Reshaping

Motivated by works such as WaveNet [68], Jukebox [42] and, in general, the move toward more "pure" data representations, we developed a method for the transformer to work with raw amplitudes.

Using the notation in [69], we reshape audio in the following way, where X is a sequence of amplitudes $X = \{x_0, x_1, ..., x_n\}$, l is the sequence length, and d is the hidden dimension:

$$X \in \mathbb{R}^{l*d\times 1} \rightarrow_{reshape} X \in \mathbb{R}^{l\times d} \tag{2.1}$$

In this way, the amount of audio that we are able to process is a combination of the sequence length of the model, and the size of the hidden dimension. Under this reshaping operation, with $l = 512$ and $d = 512$, we are able to process data up to $262,144$ samples, or nearly 6 seconds.

2.5.2.2 Curve Tokenization

Curve tokenization is an audio tokenization method that we propose, based on WordPiece tokenization [70] in NLP. The intuition behind this method is that, since audio signals typically vary smoothly, there may exist a relatively small number of "curves" that can describe short sequences of audio signals. These curves are commonly represented in audio signals by sequences of floating point numbers. In wave files, a single audio amplitude can be one of 65,536 values, or 2^{16}, values; as such, our audio is, effectively, already quantized. We term the quantization level of the audio the resolution R.

Although wave file audio is already quantized, it is advantageous to quantize it further, as doing so reduces the maximum number of theoretical curves that can exist within any given sequence of audio. As an example, an 8-token curve with $R = 100$ has a maximum number of theoretical curves of 100^8. We performed quantizations at varying levels and found that $R = 40$ produces signals that remain highly recognizable. However, we chose $R = 64$ to ensure minimal information loss.

Once quantized, we processed all the audio in our dataset using a curve length of L samples. We created a dictionary, the key of which was every unique curve sequence that we encountered, and the value of which was the number of times that curve had been seen. At $L = 8$, sliding the the L-length window across each audio signal with a stride of 1, on ESC-50, this produced a dictionary of $3.87 * 10^7$ keys. We took the top 50,000 sequences as our vocabulary, which covers 76.49% of the observed curves. At inference time, we used a stride of $L = 8$, which resulted in a overall sequence length decrease of L, also. We find that when curve-tokenizing our audio signals in this way, 76.39% of the curves are found in the vocabulary, with the remaining 23.61% represented by the equivalent of the <UNK> token in NLP.

We also created a relative vocabulary by shifting the quantized values, such that the minimum value in any 8-token span was set to zero, and all the other values maintained their relative position to the minimum, according to the Equation 2.2, where $X = \{x_0, x_1, ..., x_n\}$ is a span of audio with individual quantized values x_i.

$$X = \sum_{i=0}^{n} x_i - min(X) \tag{2.2}$$

Using the top 50,000 spans from the relative vocabulary, we find that the it covers 85.44% of the number of unique spans in the dataset. When using the relative vocabulary to tokenize audio from the dataset, we find that an average 85.43% of the curves in each audio clip are represented, with 14.57% represented by the <UNK> token.

2.5.2.3 VQ-VAE

This method was motivated by Jukebox [42], which made use of vector-quantized variational autoencoders to produce compressed "codes" to represent audio. The VQ-VAEs that we trained used the code that the authors provided, and details on the specifics of training can be found in their paper. We used their default VQ-VAE hyperparameters, which trained three VQ-VAEs, each with a codebook size of 2048, a total model size of 1 billion parameters, and downsampling rates of 128, 32, and 8. We trained the VQ-VAEs on AudioSet for 500,000 steps. In our experiments, we use the VQ-VAE with a downsampling rate of 32x.

2.5.2.4 MFCC

Mel-frequency cepstral coefficients (MFCCs) have a long history of use in audio classification problems [18, 34, 55], and so we tested their usefulness with transformers, as

well. Unless otherwise mentioned, we used 128 mels, a hop length of 512, a window length of 1024, and number of FFTs of 1024.

2.5.2.5 MFCC, GFCC, CQT, and Chromagram

Sharma *et al.* [55] reported a new state of the art on ESC-50, using four feature channels at once. They made use of MFCCs, gammatone frequency cepstral coefficients (GFCCs), a constant Q-transform (CQT), and a chromagram. Roughly speaking, the usefulness of each feature can be broken down in the following way: MFCCs are responsible for higher-frequency audio, such as speech or laughs; GFCCs are responsible for lower-frequency audio, such as footsteps or drums; CQT is responsible for music; and chromagrams are responsible for differentiating in difficult cases through the use of pitch profiles. A more extended discussion of these features is available in Sharma *et al.*'s work [55]. We made use of the same features with our transformer models, using the same parameters for feature extraction as Sharma *et al.*. In order to facilitate feeding the features into the transformer model, we concatenate the features, creating a combined feature vector of 512, which became the size of the hidden dimension.

2.5.2.6 Mel spectrogram

Other works obtaining high accuracies on ESC-50, such as the work by Salamon and Bello [51], and, more recently, Kumar *et al.*'s work with transfer learning and CNNs [27], made use of Mel spectrograms. Therefore, we also chose to include the Mel spectrogram as a feature extraction method.

Motivated by early attempts at downsampling the spectrogram, and seeing little to no decrease in accuracy on ESC-50, we perform downsampling on the spectrogram in order to reduce memory usage, which sped up experiments. The downsampling was

performed by taken every Nth column of the spectrogram matrix, where the column was frequency data at a particular timestep. In our experiments with ESC-50, we used $N = 2$ and $N = 3$. In our experiments with the Office Sounds dataset, we used $N = 1$, or no downsampling. In experiment #9 on ESC-50 (Table 2.2), we used 128 Mel bands, 1048 FFTs, hop length of 512, and window length of 1024. In experiment #10 on ESC-50, we used 256 Mel bands, 2048 FFTs, hop length of 512, and window length of 1024.

2.5.3 Augmentations

Inspired by Sharma *et al.* [55], we performed a number of augmentations to the our raw audio. We performed twelve different augmentations:

- **Amplitude clipping**: all samples are clipped at a random amplitude, determined by a percentage range, from 0.75 to 1.0, based on the maximum value in the audio.
- **Volume amplification**: all samples are multiplied by a random value, determined by a percentage range between 0.5 and 1.5.
- **Echo**: a random delay is selected between 2% and 40% of one second, and, for each value in the audio, values from the delay value number samples prior to it are added to it. E.g. at index 10,000 in an audio clip, with a random delay number of samples of 4,410, the sample from index 5590 is added to the sample at index 10,000.
- **Lowpass filter**: a fifth-order lowpass filter is passed over the audio, with a cutoff determined by a random value between 0.05 and 0.20.

- **Pitch**: the pitch is shifted by a random value from 0 to 4, using a function provided by librosa, `librosa.effects.pitch_shift`.

- **Partial erase**: a random amount of audio, from 0 to 30%, is replaced with Gaussian noise.

- **Speed adjust**: The speed of the audio is adjusted randomly between a value of 0.5 and 1.5, where greater than one is faster, and less than one is slower, using a function provided by librosa, `librosa.effects.time_stretch`.

- **Noise**: a random amount of Gaussian noise is added to every sample in the audio.

- **HPSS**: harmonic percussive source separation is performed, with a random choice between returning the harmonic part or the percussive part of the audio.

- **Bitwise downsample**: audio is downsampled by multiplying each sample by a resolution value R between 40 and 100, taking the floor of the value, and then dividing by the resolution. This reduces every sample in the audio to be represented by a maximum of R possible values.

- **Sampling rate downsample**: a value k is selected between 2 and 9, inclusive, and for every audio sample x_i, where $i = 0, k, 2k, ...$, the next k positions in the audio are overwritten with x_i. The number of samples in the audio stays the same with this method, but the overall information content of the audio is decreased. This method is similar to augmentations that downsample an image, while keeping the size of the image the same.

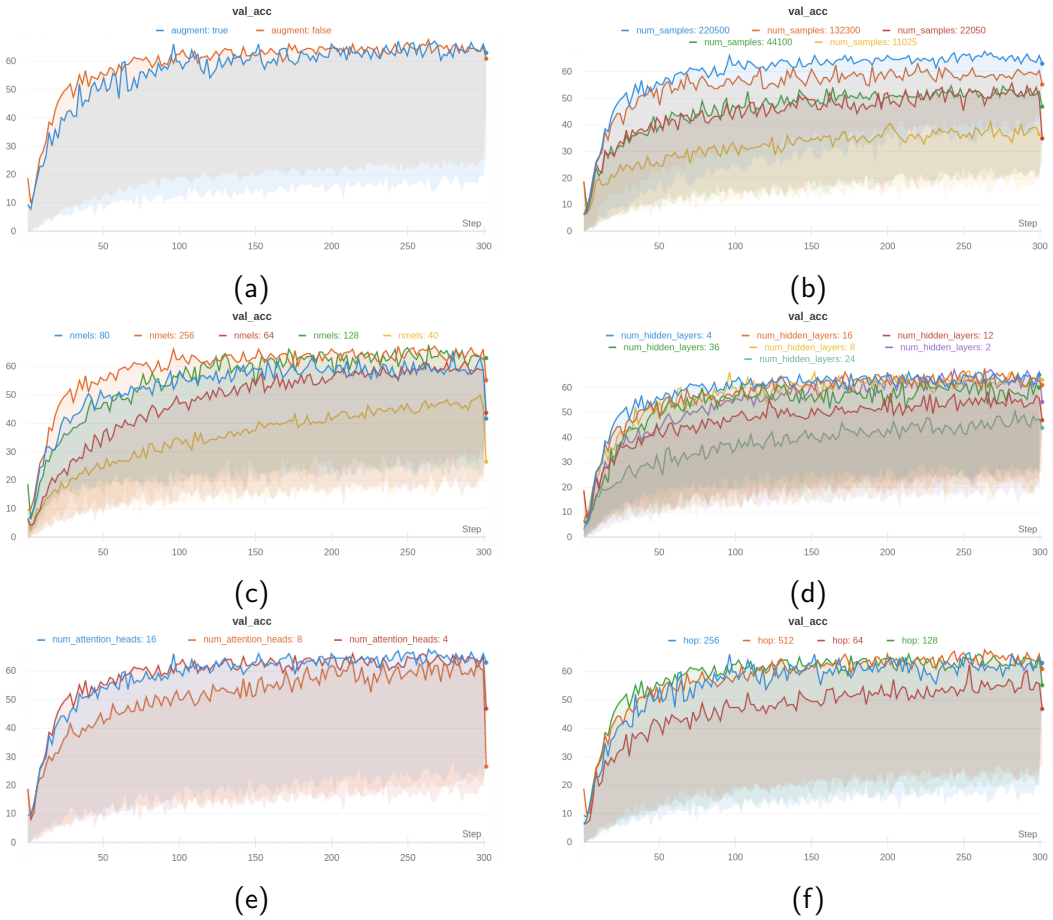

Figure 2.3: Validation accuracy on AIBERT, trained using a Mel spectrogram with varying parameters, aggregated over a total of 159 runs. The figures show results as the following parameters are varied: (a) augmentations, (b) number of samples viewed by the model at once, (c) number of Mel bands in the Mel spectrogram, (d) number of hidden layers in the model, or the depth of the model, (e) number of attention heads, and (f) the hop length when calculating the Mel spectrogram.

2.5.4 Model Conversion

To convert the model, we used PyTorch mobile and torchscript [5]. We also quantized the model using PyTorch's dynamic quantization, which is a part of PyTorch Mobile.

[5] `https://pytorch.org/mobile/home/`

We did not perform static quantization due to complexity and time constraints. We converted the model into Open Neural Network Exchange (ONNX) format [6], in an attempt to convert to TensorFlow, then to TensorFlow Lite. However, we were unsuccessful in this attempt, due to various limitations in the frameworks and conversion process.

Similarly, we attempted to convert the model to a representation that is supported on a Arduino Nano 33 BLE Sense. We attempted to convert the ONNX version of our model to TensorFlow Lite, but encountered multiple issues, one related to missing operators. We also attempted to convert it to deepC [7], but encountered similar issues, including missing support for quantized PyTorch models. We also did not complete the conversion to a microcontroller-supported representation due to complexity and time constraints.

2.6 Experiments

We performed two sets of experiments, one on ESC-50 using the six feature extraction methods described in Section 2.5.2, and a second on our Office Sounds dataset, using the best model from the first set of experiments, with some adjustments.

We used HuggingFace's Transformers library [8] for our transformer implementations. Note that HuggingFace's library assumes that a positional embedding is desirable, and has no option to remove it. Therefore, we ran a modified version of their code for our

---

[6] https://onnx.ai/

[7] https://github.com/ai-techsystems/deepC

[8] https://github.com/huggingface/transformers

experiments that did not return integer tokens during feature extraction, namely, raw amplitudes, MFCCs, GFCCs, CQTs, Chromagrams, and Mel spectrograms. We did use positional embeddings in our curve-tokenized and VQ-VAE experiments.

We used librosa v0.7.2 [9] for Mel spectrogram, MFCC, CQT, and Chromagram feature extraction, and spafe v0.1.2 [10] for GFCC feature extraction. We also used Python v3.7.7, PyTorch v1.6.0, and PyTorch Lightning v0.7.6 for machine learning. To make our experiments more accessible, we designed our models to be able to run on consumer hardware. We used two NVIDIA RTX 2080 Ti's to train all of our models, each with 11GB of RAM, with the exception of the VQ-VAE with a sequence length of 2048, experiment $\#4$, for which we used a NVIDIA Tesla V100 with 16GB of RAM.

We trained using a learning rate of 0.0001, a learning rate warmup of 10000 steps, and the Adam optimizer. Our data pipeline is implemented such that, every epoch, a random slice is taken from each audio file, optionally passed through augmentations, and then passed to the model. This has the advantage of vastly simplifying the data processing implementation, and increasing the number of ways in which a model can view a particular sound (assuming that the number of samples viewed by the model is less than the number of samples in the audio file). It does, however, have the disadvantage of reading from every audio file an equal number of times, regardless of the length of the audio. This was not a substantial issue for us, as AudioSet, ESC-50, and Office Sounds all contain roughly the same length audio files within themselves.

---

[9] `https://github.com/librosa/librosa`

[10] `https://github.com/SuperKogito/spafe`

2.6.1 Experiments on ESC-50

Table 2.2 describes the results of the trainings that we performed with each of our model types, and we discuss the results below.

2.6.1.1 Amplitude Reshaping

Experiment #1 with amplitude reshaping tested how well a transformer could learn to predict under a few unusual circumstances: (1) the inputs to the models are not constant with respect to tokens, as is usually the case with learned embeddings, (2) the model is not pretrained, and (3) the dataset is small. The performance of this model was far below comparable CNNs, but better than expected, given that transformers traditionally are pretrained with massive amounts of data, and are known to perform poorly when trained on small datasets alone. We observed that the model began to overfit around 60 epochs.

We also performed a supervised pretraining on reshaped raw amplitudes in experiment #2. This pretraining comes in the form of training on audio from AudioSet, described in Table 2.1, which has 527 labels. We trained on AudioSet for 75 epochs, with augmentations, to a maximum top-1 validation accuracy of 6.36%, after which it began to overfit. We then took that pretrained model, and finetuned it on ESC-50, without freezing any layers, according to standard practice with transformers. It is notable that this pretraining increased accuracy by 3%, compared to the non-pretrained model. When pretrained on a much larger dataset than AudioSet, it may be the case, as in [44], that a model like this obtains far higher accuracy when finetuned.

2.6.1.2 VQ-VAE

We were surprised by the inneffectiveness of VQ-VAE codes in producing good classifications. Judging by Jukebox [63], it seemed reasonable to believe that the VQ-VAE would encode a substantial amount of knowledge in the codes, which, if they are enough to produce a good reconstruction of the original audio, might also be enough to produce a classification. We did not find this to be the case, however, as they vastly underperformed compared to MFCCs, raw audio, and others. We can think of several reasons for this: first, the lack of large-scale data reduces the maximum accuracy that can be obtained from any input by a transformer, and this may be particularly true for VQ-VAE codes, since it could have been compounded by the lack of data supplied to both the VQ-VAE in learning codes through AudioSet, and the lack of data in learning classifications in ESC-50. Second, the heterogeneity of sounds in AudioSet may have significantly limited the VQ-VAE's ability to represent sounds in the codes. It has previously been shown that VAEs in general do not perform well on heterogeneous datasets [71]. As such, our VQ-VAE may not be able to perform as well on environment sound tasks as it did on music tasks, given the large variety of sounds present in ESC versus music.

Hypothesizing that the short sequence length of our first experiment (512) may have resulted in the transformer not be able to have a sufficient view of the code to perform a classification, we attempted a much longer sequence length, using a V100 GPU with 16GB of RAM. Even with a sequence length of 2048, which translates to an effective number of samples of 65,536, or about 1.5 seconds, we did not observe a substantial increase in accuracy, still falling far below other feature extraction methods.

As with the rest of the methods in this work, the first step to increasing accuracy on ESC using VQ-VAE codes is to obtain more data. Training on a much larger corpus

of unlabeled audio is entirely possible in the first step to creating VQ-VAE codes, and may improve the quality of the codes created. Additionally, using techniques such as the ones presented by Nazabal *et al.* [71], to alter the VQ-VAE to enable to it better handle heterogeneous data, may help as well. It may also be of value to perform a pretraining step, either supervised or unsupervised, and finetune on more ESC data. However, even with all such adjustments, it seems unlikely that VQ-VAE codes will exceed MFCCs, Mel spectrograms or raw audio in predictive capability.

2.6.1.3 MFCC, GFCC, CQT, and Chromagram

In experiments #5 and #6, we observe that augmentations make a substantial (5.2%) impact on accuracy. We also see that MFCC's perform better, though only slightly so, than raw amplitudes. These experiments were performed with 128 Mel bands, which resulted in the hidden size H of the model to be 128 as well. These models began to overfit around 50 epochs.

Experiments #7 and #8 showed that adding additional feature extraction methods improved the accuracy of the model beyond only using MFCCs, especially in the non-augmented case. However, when augmented, the model did not show any major improvements, as had occurred with MFCCs. This is different than the results by Sharma *et al.* [55], which had used augmentations to improve accuracy by more than 3%. However, we note that for our purposes – inferring at the edge – the cost of computing features using all four extraction methods becomes prohibitive, and the model would have been unlikely to be of use at the edge, even it it had obtained high accuracy. We also found that extracting these features at training time resulted in extremely slow training, which hindered additional experimentation with these features.

2.6.1.4 Mel Spectrogram and Hyperparameter Search

We trained using a Mel spectrogram in experiments #9 and #10, and obtained accuracy that outperformed any other feature extraction methods. This is particularly advantageous at the edge, since computing a Mel spectrogram is a reasonably inexpensive operation. Of note, as well, is the fact that this was obtained with a smaller sequence length than experiments #7 and #8, due to the 3x downsampling that we performed. We also used AIBERT, and a longer sequence length that other models, which may have contributed to the improved performance. Judging by the performance of BERT-based transformers trained on Office Sounds, however, it seems unlikely that AIBERT would result in a significant performance improvement alone. The impact of number of samples is discussed below.

Since this was our best-performing model, we performed a hyperparameter search to determine the optimal parameters. All training was performed using AIBERT as the base model, with a downsampling rate of 2x. We performed 159 training runs, which are aggregated into the graphs in Figure 2.3.

Some clear improvements result by changing certain parameters. The most obvious is the number of samples that are passed into the Mel spectrogram, which, as it increases, also increases the maximum possible validation accuracy. We chose a peak of 220,500 samples, or 5 seconds of audio, because files in ESC-50 audio have a maximum length of 5 seconds. As can be seen, a model's access to the full file's worth of data improves its ability to classify well.

Another clear result is the importance of using more than 80 Mel bands when creating the Mel spectrogram. This result is particularly important, as many research works make use of 80 Mels or less [13, 14, 46, 64, 72, 73], which likely reduced accuracy in those works.

2.6.1.5 Curve Tokenization

As a first attempt in literature at tokenizing audio based on curves, for the purpose of training a transformer, we find that they provide very little predictive power. There may be several reasons for this, the first of which is the small number of samples over which the model can view a sound. Since every 8 samples is quantized and converted into a token, using a sequence length of 512, the number of effective samples is 4096, which is only 93 milliseconds of audio. This is likely a limiting factor on the predictive ability of the model, and a model able to handle a much longer sequence length would likely perform better. It is also likely that quantizing it reduced the information content of the audio, and further reduced the predictive power.

In the case of absolute curves, we find that augmentations substantially reduce accuracy. This is likely due to the fact that our vocabulary was created on ESC-50 without augmentations, so the curves that appear with augmentations result in many more <UNK> tokens. We see a slight increase in relative curve tokenization with augmentations, but, given the incredibly low accuracy of the model, find it to be of little interest.

Overall, we consider it unlikely that curve tokenization would ever beat out more well-known feature extraction techniques. It removes too much information, such as vital frequency and phase information, which other feature extraction methods allow the transformer to make use of. Nonetheless, we consider it an interesting experiment in possible tokenization techniques for transformers on audio.

2.6.2 Experiments on Office Sounds

After completing our experiments on ESC-50, we trained on the Office Sounds dataset [67]. We used BERT-based models only, with an emphasis on model size, specifically on reducing the model size while maintaining accuracy in order to perform more efficient processing at the edge. We were particularly interested in models which were capable of being run on microcontrollers; in our case, we chose a target model size of 256KB or less – the available SRAM on the Arduino Nano 33 BLE Sense – which meant that the model must be less than 250,000 parameters when quantized. There are, of course, methods to run larger models with less SRAM, such as MCUNet [74], but we left such optimizations to a future work, focusing on the generic case of running a transformer on a microcontroller without any special optimizations.

We made several adjustments between our ESC-50 experiments and our Office Sounds experiments, described in Section 2.4, which enabled us to experiment with vastly different model sizes. We began by choosing a model with parameters similar to our best-performing models from the hyperparameter search. We chose the model seen in experiments #4 and #5 of Table 2.3, based on Mel spectrogram input with a hop length of 512, window size of 1024, number of FFTs of 1024, and Mel bands of 128. It had 8 layers, and a hidden size H larger than we had been able to use in the ESC-50 experiments, of 512, and 8 heads. We also removed downsampling, making the sequence length much longer than before, but still able to fit within the constraints of consumer-grade GPUs while maintaining a reasonable batch size. These models obtained a maximum validation accuracy of 93.75%, with augmentations, and began to overfit after 200-300 epochs.

In order to facilitate accurate comparisons to our previous work [67], we reimplemented and performed training on Office Sounds using MFCCs as input to a CNN,

shown in Table 2.4. Following that work, the CNN was an exact reimplementation of Kumar *et al.*'s model in [27]. We trained the model, non-augmented, on ESC-50 to confirm accurate implementation, and obtained 81.25%, which is very close to the pretrained model accuacy that Kumar *et al.* reported. We performed training of this MFCC-based CNN against Office Sounds, using a random slice of the each audio file in each epoch, and obtained a maximum of 92.97% accuracy, using augmentations on 2.5 seconds of audio. This corresponds to the results obtained in [67], even though the training scheme is slightly different. The model contained 4.5 million parameters, and nearly 500 million multiply-adds. The inference time of this model on a Samsung Galaxy S9 was an average of 57 milliseconds [67], non-augmented and non-quantized, using TensorFlow Lite, shown in Table 2.5. We note that augmentations had a small positive effect on validation accuracy, and that increasing the visible audio window size from 2.5 to 5 seconds had a slightly negative effect.

In comparing the Office Sounds transformers to the Office Sounds CNNs, we find that the transformers outperform the CNN, while being much smaller. A model with 95.2% less parameters, transformer experiment #3, outperformed the CNN by more than 2%. The smallest model that we trained on 5 seconds of audio, experiment #2, 99.85% smaller than the CNN, also outperformed the CNNs. This is unexpected, since CNNs far outperformed transformers on the ESC-50 experiments. Our hypothesis is that the increased number of example data for each class in Office Sounds (200 or more per class, compared to 40 per class in ESC-50), assisted in preventing as rapid overfitting as was observed in ESC-50. This can be tested by reducing the number of samples in Office Sounds and running these experiments again; however, this is costly, and so we leave this for a future work.

We also trained our smallest model on one second of data (experiment #1), and

found that it substantially reduced the accuracy of the model. Interestingly, for some applications, it may be worthwhile to process only one second of audio with reduced accuracy, as it reduces the cost of feature extraction and allows the model to be run more frequently. Also, predictions over each second can be aggregated across a longer time span via majority voting, or something similar, in order to potentially produce more accurate predictions.

2.6.3 Inference at the Edge

Table 2.5 shows feature extraction and inference times on a Samsung Galaxy S9. This model uses a transformer based on a Mel spectrogram, processing 5 seconds of audio data and producing a classification using ESC-50 labels. We observe that, even on a device more than two years old, inference is still fast enough to be performed many times a second. We also find that quantization results in lowered latency (about 21% with dynamic quantization), which further increases the potential model size. Static quantization is likely to reduce latency further, as dynamic quantization does not quantize model activations.

We also observed a substantially decreased inference time from our smallest model, as expected, inferring 93% faster than the 1-million parameter transformer. It was surprising to find that the much larger CNN had faster inference times than the 1-million parameter transformer, however, this may be due to optimizations in TensorFlow Lite that are not present in PyTorch Mobile, or simply that CNN operations are optimized further than transformer operations on edge devices.

2.7 Conclusion and Future Work

Efficient edge processing is a challenging, but critical task which will become increasingly important in the future. To aide in this task, we trained a 6,000-parameter transformer on the Office Sounds dataset that outperforms a CNN more than 700x larger than it. This enables accurate and efficient environmental sound classification of office sounds on edge devices, even on inexpensive microcontrollers, resulting in inference times on a Samsung Galaxy S9 that are 88% faster than a CNN with comparable accuracy. We find that models trained in traditional frameworks (like PyTorch) have relatively little support for conversion to models that can be run at the edge (like on a microcontroller), even with the development of ONNX.

Our ESC-50 transformer models did not outperform CNNs, as they did on Office Sounds. Understanding this, and finding solutions to the problem of training transformers on small audio datasets, is a crucial future work. Solutions may come through large amounts of unsupervised pretraining, through an architectural change, or though improved supervised ESC datasets. In any case, our work provides groundwork upon which these questions can be answered.

The small size and efficiency of the transformer we trained raises questions about the cost of retraining. It may be that, because there are so few operations (less than 6,000) required in a forward pass, that *on-device* retraining becomes possible, similar to what is done on Coral Edge TPUs through the imprinting engine [75]. This would have vast implications on the future of intelligent edge analytics, and even a variety of user applications, and is deserving of future work.

The trend in machine learning and artificial intelligence, in general, is toward larger models and more data. We consider this work to be evidence that accuracy does not

always require scale, and that this is true even for transformers.

Table 2.2: Accuracy on Office Sounds dataset, running under various feature extraction and training schemes.

#	Input	Accuracy	Samples	Layers	Heads	Sequence Len	Batch	Augment	Type
1	Amplitude reshaping	48.96	44100	8	8	256	16	True	BERT
2	Amplitude reshaping (Pretrained)	52.08	44100	16	16	256	16	True	BERT
3	VQ-VAE (32x)	31.77	16384	8	8	512	32	False	BERT
4	VQ-VAE (32x)	34.50	65536	8	8	2048	4	False	BERT
5	MFCC	53.13	44100	8	8	173	32	False	BERT
6	MFCC	58.33	44100	8	8	173	32	True	BERT
7	MFCC, GFCC, CQT, and Chromagram	59.38	88200	8	8	173	32	False	BERT
8	MFCC, GFCC, CQT, and Chromagram	59.90	88200	8	8	173	32	True	BERT
9	Mel spectrogram (Downsampled 3x)	60.45	220500	8	8	143	64	False	AlBERT
10	Mel spectrogram (Optimized)	67.71	220500	16	16	215	16	True	AlBERT
11	Curve Tokenization (Relative)	7.81	4096	8	8	512	16	False	BERT
12	Curve Tokenization (Relative)	8.85	4096	8	8	512	16	True	BERT
13	Curve Tokenization (Absolute)	19.79	4096	8	8	512	16	False	BERT
14	Curve Tokenization (Absolute)	13.54	4096	8	8	512	16	True	BERT

Table 2.3: Transformer accuracy on Office Sounds dataset for various models, ordered by number of paramaters. All models were based on BERT, and had the feed-forward layer size set to $4H$.

#	Input	Accuracy	Params	Multiply-Adds	Samples	Layers	Hidden	Heads	Sequence Len	Batch	Augment
1	Mel spectrogram	81.48%	5,954	5,638	44100	1	16	2	86	64	True
2	Mel spectrogram	93.21%	6,642	5,982	220500	1	16	2	430	64	True
3	Mel spectrogram	95.31%	213,858	210,414	220500	4	64	4	430	64	True
4	Mel spectrogram	93.75%	25,553,762	25,506,734	220500	8	512	8	430	16	True
5	Mel spectrogram	89.38%	25,553,762	25,506,734	220500	8	512	8	430	16	False

Table 2.4: CNN accuracy on Office Sounds dataset, ordered by accuracy.

#	Input	Accuracy	Params	Multiply-Adds	Samples	Batch	Augment
1	MFCC	92.97%	4,468,022	478,869,984	110250	64	True
2	MFCC	92.19%	4,468,022	478,869,984	110250	64	False
3	MFCC	91.41%	4,468,022	956,052,832	220500	16	False

Table 2.5: Inference times of selected models on an edge device. Models are run on a Samsung Galaxy S9 using PyTorch Mobile, except for the first, which was run with TensorFlow Lite. Results are averaged over 10 runs. Quantization is PyTorch dynamic quantization.

Experiment	**Params**	**Mult-Adds**	**Latency (ms)**
MFCC CNN from [67]	4,468,022	478,869,984	57
ESC-50 #10	958,146	948,888	111
ESC-50 #10, Quant.	958,146	948,888	88
Office Sounds, Trans. #2	6,642	5,982	7

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[2] Nicholas D. Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. Squeezing Deep Learning into Mobile and Embedded Devices. *IEEE Pervasive Computing*, 16(3):82–88, 2017.

[3] George Plastiras, Maria Terzi, Christos Kyrkou, and Theocharis Theocharidcs. Edge intelligence: Challenges and opportunities of near-sensor machine learning applications. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–7. IEEE, 2018.

[4] Shikhar Verma, Yuichi Kawamoto, Zubair Md Fadlullah, Hiroki Nishiyama, and Nei Kato. A survey on network methodologies for real-time analytics of massive iot data and open research issues. *IEEE Communications Surveys & Tutorials*, 19(3):1457–1477, 2017.

[5] Zachary K Pecenak, Jan Kleissl, and Eric Lam. Detection of a surface detonated nuclear weapon using a photovoltaic rich distribution grid. In *2018 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2018.

[6] Yang Cai, Eric Lam, Todd Howlett, and Alan Cai. Spatiotemporal analysis of "jello effect" in drone videos. In *International Conference on Applied Human Factors and Ergonomics*, pages 197–207. Springer, 2019.

[7] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[8] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[9] Charles Leech, Yordan P Raykov, Emre Ozer, and Geoff V Merrett. Real-time room occupancy estimation with bayesian machine learning using a single pir sensor and microcontroller. In *2017 IEEE Sensors Applications Symposium (SAS)*, pages 1–6. IEEE, 2017.

[10] Aly Metwaly, Jorge Peña Queralta, Victor Kathan Sarker, Tuan Nguyen Gia, Omar Nasir, and Tomi Westerlund. Edge computing with embedded ai: Thermal image analysis for occupancy estimation in intelligent buildings. *INTelligent Embedded Systems Architectures and Applications, INTESA@ ESWEEK*, 2019.

[11] Dalibor Mitrović, Matthias Zeppelzauer, and Christian Breiteneder. Features for content-based audio retrieval. In *Advances in computers*, volume 78, pages 71–150. Elsevier, 2010.

[12] Venkatesh Boddapati, Andrej Petef, Jim Rasmusson, and Lars Lundberg. Classifying environmental sounds using image recognition networks. *Procedia Computer*

Science, 112:2048–2056, 2017.

[13] Karol J. Piczak. Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. ISSN: 2378-928X.

[14] Juncheng Li, Wei Dai, Florian Metze, Shuhui Qu, and Samarjit Das. A comparison of Deep Learning methods for environmental sound detection. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 126–130, New Orleans, LA, March 2017. IEEE.

[15] Rishabh N. Tak, Dharmesh M. Agrawal, and Hemant A. Patil. Novel Phase Encoded Mel Filterbank Energies for Environmental Sound Classification. In B. Uma Shankar, Kuntal Ghosh, Deba Prasad Mandal, Shubhra Sankar Ray, David Zhang, and Sankar K. Pal, editors, *Pattern Recognition and Machine Intelligence*, volume 10597, pages 317–325. Springer International Publishing, Cham, 2017.

[16] Dharmesh M. Agrawal, Hardik B. Sailor, Meet H. Soni, and Hemant A. Patil. Novel TEO-based Gammatone features for environmental sound classification. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 1809–1813, Kos, Greece, August 2017. IEEE.

[17] Reona Mogi and Hiroyuki Kasai. Noise-Robust environmental sound classification method based on combination of ICA and MP features. *Artificial Intelligence Research*, 2(1):p107, November 2012.

[18] Sachin Chachada and C.-C. Jay Kuo. Environmental sound recognition: a survey. *APSIPA Transactions on Signal and Information Processing*, 3:e14, 2014.

[19] Zhichao Zhang, Shugong Xu, Shunqing Zhang, Tianhao Qiao, and Shan Cao. Learning Attentive Representations for Environmental Sound Classification. *IEEE*

Access, 7:130327–130339, 2019.

[20] Xiaohu Zhang, Yuexian Zou, and Wei Shi. Dilated convolution neural network with LeakyReLU for environmental sound classification. In *2017 22nd International Conference on Digital Signal Processing (DSP)*, pages 1–5. ISSN: 2165-3577.

[21] Dading Chong, Yuexian Zou, and Wenwu Wang. Multi-channel Convolutional Neural Networks with Multi-level Feature Fusion for Environmental Sound Classification. In Ioannis Kompatsiaris, Benoit Huet, Vasileios Mezaris, Cathal Gurrin, Wen-Huang Cheng, and Stefanos Vrochidis, editors, *MultiMedia Modeling*, volume 11296, pages 157–168. Springer International Publishing, Cham, 2019.

[22] Karol J Piczak. Esc: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1015–1018. ACM, 2015.

[23] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1041–1044. ACM, 2014.

[24] Annamaria Mesaros, Aleksandr Diment, Benjamin Elizalde, Toni Heittola, Emmanuel Vincent, Bhiksha Raj, and Tuomas Virtanen. Sound event detection in the dcase 2017 challenge. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 27(6):992–1006, 2019.

[25] Eduardo Fonseca, Manoj Plakal, Frederic Font, Daniel PW Ellis, Xavier Favory, Jordi Pons, and Xavier Serra. General-purpose tagging of freesound audio with audioset labels: Task description, dataset, and baseline. *arXiv preprint arXiv:1807.09902*, 2018.

[26] Eduardo Fonseca, Jordi Pons Puig, Xavier Favory, Frederic Font Corbera, Dmitry

Bogdanov, Andres Ferraro, Sergio Oramas, Alastair Porter, and Xavier Serra. Freesound datasets: a platform for the creation of open audio datasets. In *Hu X, Cunningham SJ, Turnbull D, Duan Z, editors. Proceedings of the 18th ISMIR Conference; 2017 oct 23-27; Suzhou, China.[Canada]: International Society for Music Information Retrieval; 2017. p. 486-93.* International Society for Music Information Retrieval (ISMIR), 2017.

[27] Anurag Kumar, Maksim Khadkevich, and Christian Fugen. Knowledge Transfer from Weakly Labeled Audio Using Convolutional Neural Network for Sound Events and Scenes. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 326–330, Calgary, AB, April 2018. IEEE.

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[29] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, 2015.

[30] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. Very deep convolutional neural networks for raw waveforms. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 421–425. IEEE, 2017.

[31] François Chollet et al. Keras. `https://keras.io`, 2015.

[32] Amadej Trnkoczy. Topic understanding and parameter setting of sta/lta trigger algorithm. *New manual of seismological observatory practice*, 2, 1999.

[33] Guy Barrett Coleman and Harry C Andrews. Image segmentation by clustering. *Proceedings of the IEEE*, 67(5):773–785, 1979.

[34] Michael Cowling and Renate Sitte. Comparison of techniques for environmental sound recognition. *Pattern recognition letters*, 24(15):2895–2907, 2003.

[35] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[36] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

[37] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[38] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

[39] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[40] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.

[41] Mark Chen, Alec Radford, Rewon Child, Jeff Wu, Heewoo Jun, Prafulla Dhariwal, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *Proceedings*

of the 37th International Conference on Machine Learning, 2020.

[42] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[43] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017.

[44] Anonymous. An image is worth 16x16 words: Transformers for image recognition at scale. In *Submitted to International Conference on Learning Representations*, 2021. under review.

[45] Ngoc-Quan Pham, Thai-Son Nguyen, Jan Niehues, Markus Müller, Sebastian Stüker, and Alexander Waibel. Very deep self-attention networks for end-to-end speech recognition.

[46] Ruixiong Zhang, Haiwei Wu, Wubo Li, Dongwei Jiang, Wei Zou, and Xiangang Li. Transformer based unsupervised pre-training for acoustic representation learning. *arXiv:2007.14602 [cs, eess]*, July 2020. arXiv: 2007.14602.

[47] Yangyang Shi, Yongqiang Wang, Chunyang Wu, Christian Fuegen, Frank Zhang, Duc Le, Ching-Feng Yeh, and Michael L. Seltzer. Weak-Attention Suppression For Transformer Based Speech Recognition. *arXiv:2005.09137 [cs, eess]*, May 2020. arXiv: 2005.09137.

[48] Dan Stowell, Dimitrios Giannoulis, Emmanouil Benetos, Mathieu Lagrange, and Mark D. Plumbley. DCASE 2016 Acoustic Scene Classification Using Convolutional Neural Networks. *IEEE Transactions on Multimedia*, 17(10):1733–1746, October 2015.

[49] Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 776–780. IEEE, 2017.

[50] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. Very deep convolutional neural networks for raw waveforms. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 421–425. ISSN: 2379-190X.

[51] Justin Salamon and Juan Pablo Bello. Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification. *IEEE Signal Processing Letters*, 24(3):279–283, March 2017.

[52] Yuji Tokozume and Tatsuya Harada. Learning environmental sounds with end-to-end convolutional neural network. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2721–2725. ISSN: 2379-190X.

[53] Sajjad Abdoli, Patrick Cardinal, and Alessandro Lameiras Koerich. End-to-end environmental sound classification using a 1D convolutional neural network. *Expert Systems with Applications*, 136:252–263, December 2019.

[54] Aditya Khamparia, Deepak Gupta, Nhu Gia Nguyen, Ashish Khanna, Babita Pandey, and Prayag Tiwari. Sound classification using convolutional neural network and tensor deep stacking network. 7:7717–7727. Conference Name: IEEE Access.

[55] Jivitesh Sharma, Ole-Christoffer Granmo, and Morten Goodwin. Environment Sound Classification using Multiple Feature Channels and Attention based Deep Convolutional Neural Network. *arXiv:1908.11219 [cs, eess, stat]*, April 2020. arXiv:

1908.11219.

[56] D. Giannoulis, E. Benetos, D. Stowell, M. Rossignol, M. Lagrange, and M. D. Plumbley. Detection and classification of acoustic scenes and events: An ieee aasp challenge. In *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 1–4, Oct 2013.

[57] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, May 2020. arXiv: 2005.14165.

[58] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108 [cs]*, February 2020. arXiv: 1910.01108.

[59] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for BERT model compression.

[60] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*, 2019.

[61] Christophe Couvreur, Vincent Fontaine, Paul Gaunard, and Corine Ginette Mubikangiey. Automatic classification of environmental noise events by hidden markov models. page 20.

[62] Zohaib Mushtaq, Shun-Feng Su, and Quoc-Viet Tran. Spectral images based environmental sound classification using cnn with meaningful data augmentation. *Applied Acoustics*, 172:107581.

[63] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music.

[64] Koichi Miyazaki, Tatsuya Komatsu, Tomoki Hayashi, Shinji Watanabe, Tomoki Toda, and Kazuya Takeda. Weakly-supervised sound event detection with self-attention. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 66–70. ISSN: 2379-190X.

[65] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *arXiv:1909.11942 [cs]*, February 2020. arXiv: 1909.11942.

[66] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[67] David Elliott, Evan Martino, Carlos E Otero, Anthony Smith, Adrian M Peter, Benjamin Luchterhand, Eric Lam, and Steven Leung. Cyber-physical analytics: Environmental sound classification at the edge. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–6. IEEE, 2020.

[68] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[69] Matthias Sperber, Jan Niehues, Graham Neubig, Sebastian Stüker, and Alex

Waibel. Self-Attentional Acoustic Models. *arXiv:1803.09519 [cs]*, June 2018. arXiv: 1803.09519.

[70] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[71] Alfredo Nazabal, Pablo M Olmos, Zoubin Ghahramani, and Isabel Valera. Handling incomplete heterogeneous data using vaes. *Pattern Recognition*, page 107501, 2020.

[72] Yilun Zhao, Xinda Wu, Yuqing Ye, Jia Guo, and Kejun Zhang. Musicoder: A universal music-acoustic encoder based on transformers. *arXiv preprint arXiv:2008.00781*, 2020.

[73] Yang Jiao. Translate reverberated speech to anechoic ones: Speech dereverberation with bert. *arXiv preprint arXiv:2007.08052*, 2020.

[74] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319*, 2020.

[75] Retrain a classification model on-device with weight imprinting.