BENCHMARKING SPECTRAL IMAGE GENERATION SCHEMA AND PRE-TRAINED CONVOLUTIONAL NEURAL NETWORK ARCHITECTURES FOR AUDIO CLASSIFICATION TASKS

A Thesis

Presented to the

School of Data Science

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment
of the Requirements for the Degree

Data Science, MS

University of Nebraska at Omaha

by

Kyle J. Hammitt

May 2023

Supervisory Committee

Dr. Dustin White

Dr. Seth Shafer

Dr. Mahbubul Majumder

BENCHMARKING SPECTRAL IMAGE GENERATION SCHEMA AND PRE-TRAINED CONVOLUTIONAL NEURAL NETWORK ARCHITECTURES FOR AUDIO CLASSIFICATION TASKS

Kyle J. Hammitt, MS

University of Nebraska, 2023

Advisor: Dr. Dustin White

Recent advancements in image classification such as Convolutional Neural Networks can be employed to perform audio classification. While there has been work done on benchmarking different networks and image representations, many techniques remain unexplored. Three popular pre-trained image networks (EfficentNet-b0, ResNet50, SqueezeNet) and four different image representations of audio (Cochleagram, Approximate Cochleagram, Linear Gammachirp, and Logarithmic Gammachirp) are benchmarked by performance and preprocessing cost using the UrbanSound8K audio dataset, which contains common sounds that contribute to noise pollution in urban areas. EfficientNet-b0 and ResNet50 achieved comparable performance for each of the four data representations and significantly outperformed SqueezeNet. The Linear Gammachirp and the Cochleagram provided for the highest classification accuracy, with the Linear Gammachirp slightly but not significantly outperforming Cochleagrams. Findings were surprising because the logarithmic gammachirp filter is a much more computationally complex model of the cochlear inner ear than the other three, and yet it yielded the worst results. Findings suggest that more complex cochlear models can achieve but do not guarantee better results out of the box, and that optimal tuning of parameters in each cochlear representation may yield improved results. The limitations of current classification techniques and suggestions for further exploration of different data representations are discussed.

Copyright 2023, Kyle J. Hammitt

Table of Contents

List of Multimedia Objects

Introduction to Audio Classification

Recently, advancements in image and audio feature extraction methods such as deep neural networks have been of much interest in the research community as audio event detection and classification will soon become a key technology. It is already being used to aid medical diagnosis (Sharan et al. 2019), detect adverse emotional states (Salekin et al. 2018), and to build sound event detection systems (Singh et al. 2021). The three main uses for audio feature extraction and classification are sound event detection, human speech recognition, and musical classification tasks such as metadata (genre, source) tagging.

Convolutional neural networks (CNNs) are a popular model choice for image and audio classification tasks. Their architecture typically contains some convolutional and/or pooling layers which are designed to extract features. Those features are then used as inputs in fully connected layers which are designed to classify the input (O'Shea and Nash 2015). Network identified features were shown to outperform previous methods of audio handmade feature extraction such as Mel Frequency Cepstral Coefficients, which tend to perform poorly in the presence of interfering acoustic noise (Zhang et al. 2015, Wang and Hu 2018). Given that sound event detection signals tend to have noise when collected in the field, CNNs have since become one of the most effective methods of audio sound event classification (Piczak 2015, Valenti et al. 2016, Takahashi et al. 2016), human speech recognition tasks (Anvarjon et al. 2021, Mustaqeem and Kwon 2021), and musical classification (Choi et al. 2017, Costa et al. 2017). Their popularity has sparked much research pertaining to the development of suitable image representations of audio.

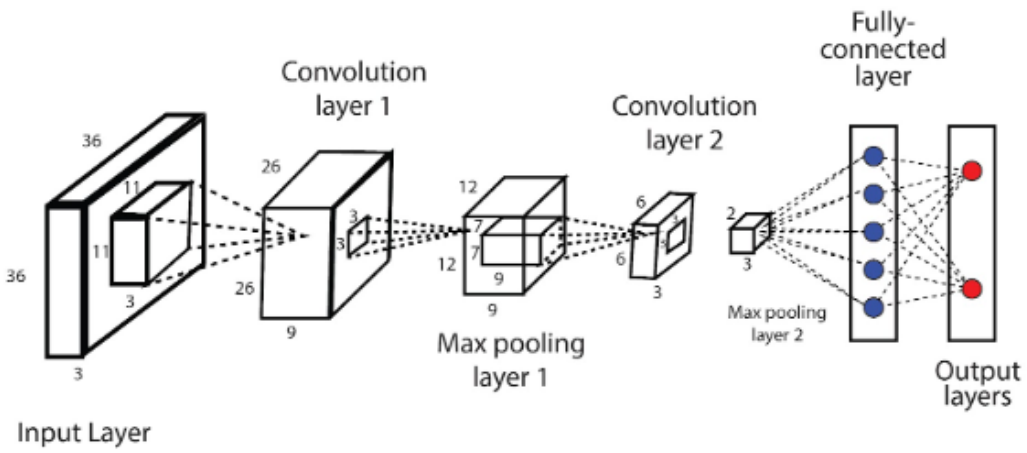

Figure 1: Visualization example of a simple Convolutional Neural Network architecture (Balodi 2019)

There are many ways to represent audio using images, and an abundance of research has been done to determine the most suitable representation of audio data for CNN classification tasks. A conventional spectrogram is a visual time-frequency representation an audio signal, created by applying discrete Fourier transform (DFT) to an audio signal and then taking the log of the absolute value of the resulting DFT values (Allen 1977, Allen 1982). Spectrograms were one of the first data representations to be proposed for use with CNN architecture, and their introduction for audio classification tasks motivated further research to identify other suitable audio data representations for use (Rippel et al. 2015). The conventional spectrogram (formed using short-time Fourier transform, STFT) is still widely used in applications such as speech emotion and spoken digit recognition, but at the time of its inception, using spectrograms proved to be computationally expensive. To create a more effective and less expensive method, filters such as Mel-filters (Davis and Mermelstein 1980) and/or moving average filters (Brown 2004) were implemented to create new versions of the spectrogram by introducing domain knowledge about how humans perceive audio. The Mel-filter (which is used in

computing Mel-frequency cepstral coefficients - MFCCs) has frequency bands equally spaced on the Mel-scale, more accurately resembling the way the human ear perceives sound. Using these filters, the Mel-spectrogram was created and has been presented as an improvement over conventional spectrograms for CNN speech and audio event classification (Zhang et al. 2019, Zhou et al. 2021).

Currently, one of the most popular representations of audio signal data is the cochleagram, which is a spectral image representation of an audio signal which has been based off the frequency selectivity of the human cochlea (Sharan and Moir 2019). This representation is constructed by filtering the audio signal using gammatone filters and adding the energy in the windowed signal for each frequency channel. All previously mentioned representations have a range of 0 Hz to 22,050 Hz (Nyquist frequency), but cochleagrams display the lower frequency range with a finer resolution than conventional spectrograms, without losing spectral information in the upper frequency range (Sharan and Moir 2015). Cochleagrams continue to show promise as the most effective data representation for CNN audio classification tasks in terms of audio event classification accuracy, because audio event data tends to contain more information in the lower frequency range (Sharan and Moir 2019, Sharan et al. 2021).

One caveat to the success of CNNs in classifying audio is that CNN architecture expects a specified input dimensionality, so each image representation of an audio signal must either conform to, or be transformed to, some expected image size (O'Shea and Nash 2015). Given that many audio events have variable durations, the image representation must be resized or down sampled to a certain frequency resolution to match the input dimensionality expected by the CNN. One common approach to solve this issue is to

divide the signal into an equal number of frames, but dividing the signal can result in small frames for short signals, possibly making it difficult to capture unique frequency characteristics. An alternative is to divide the signal into frames of a fixed length, but this results in a different number of frames for different signals. Other techniques used to get an equal sized time-frequency representation include image resizing, zero-padding, and cropping (Sharan et al. 2021).

In further empirical research, many different audio representations (spectrogram, smoothed spectrogram, Mel-spectrogram, cochleagram), image resizing techniques (algorithms such as bilinear, bicubic, and Lanczos), and network information fusion techniques (early, mid, late fusion) were benchmarked to provide empirical evidence of each representation or technique's accuracy in performing audio sound event classification tasks (Sharan et al. 2021). These findings confirmed earlier research done on the effectiveness of cochleagrams for CNN classification: the largest improvements in accuracy over conventional spectrogram representations were achieved by representing the audio with a cochleagram. Popular image resizing techniques, such as Bicubic and Lanczos, (when applied to a conventional spectrogram) showed comparable accuracies to using smoothed spectrograms or Mel-spectrograms but failed to outperform cochleagrams.

Finally, these findings showed that ensemble learning methods (combining spectrograms, Mel-spectrograms, smoothed spectrograms, and cochleagrams) were able to marginally outperform cochleagrams, and demonstrated that late fusion (incorporating all networks' output activations – otherwise known as classification probabilities - into some weighted average of activations) showed the highest improvements to classification accuracy,

although all three methods (early, mid, and late) showed very similar amounts of accuracy gain. Early fusion involves combining inputs of the CNN as additional channels (increasing dimensionality and computation cost), while mid fusion involves freezing and exporting neuron weights of one network with another during the training process. Certain combinations of representations and ensemble methods were able to achieve upwards of 98% accuracy for sound event classification, although it should be noted that such high accuracies often require network architecture and hyperparameter tuning that is specific to the dataset and are only a result of continued efforts by multiple teams to increase benchmarked accuracy over a specific problem domain. Accuracy is often just as data dependent as it is network and hyperparameter dependent.

One caveat to deep learning classification is that these algorithms are typically "trained in separation on a specific feature space and same distribution" **(**Tammina, 2019). The main consequence of this problem is that the network architecture might have to be rebuilt in the case that the feature space it was initially trained on has changed, which can be resource intensive. Given that deep learning networks are used to perform classification tasks for both image and audio classification tasks, it is "nearly impossible to create a machine learning based network for a target domain which consists of very few labelled data for supervised learning" (Tammina, 2019). For example, if we want to train a network to classify cat and dog sounds, we need a very large database of labeled cat and dog sound data to fully train a network from scratch over the target domain. However, because image recognition networks are trained to recognize low-level features in an image such as edges and shapes, networks already trained to recognize these patterns can be re-trained over a specific feature space. The network uses patterns

previously recognized in other training data in addition to what it learns from a smaller set of data collected for the new task.

This process is called transfer learning, and the way we achieve this using a CNN architecture is by freezing weights of the convolution and pooling layers of the network. These pre-trained layers already contain information that allows the network to discern shapes, edges, and other patterns in images at varying levels of abstraction. By reusing weights from another network already trained on massive amounts of data, we are repurposing the network's ability to recognize general patterns and reducing the amount of computation and time needed to train the network. Typically, the only layers being trained during transfer learning are the fully connected layers used for classification, which are retrained for our specific classification task (effectively *repurposing* the pre-trained network for a new task). In most cases the old classification head is removed from the network and replaced with one or more fully connected layers. This enables users to use the previously trained weights in the convolutional base, while training only the classifier head, thus significantly reducing the number of trainable parameters. Previous research has found that a network using frozen layer weights from a popular pre-trained network for image recognition (VGG-16) vastly outperformed a network that was trained only over the sound event of interest (Tammina, 2019). Additionally, if the new task domain is too distinct from the task domain the original network was trained on, we can unfreeze all weights in the network to see if training all network weights yields better fit. However, while this often yields better network performance, it must be acknowledged that this is computationally expensive and there is always a risk the network will destroy its previous learning when training on the new task. Still, in many

cases, training all network weights often provides better network performance and is computationally feasible on newer graphics processing units (GPU).

Immediate Areas of Further Research

Despite all the advancement made in recent years in the field of image classification, there is still much work to be done in the less mature research field of audio signal processing (Zhang et al. 2019). Primarily, the limitations and possibilities for audio classification are still being explored, and experimentation with different combinations of signal processing methods and CNN-based architecture is encouraged (Nanni et al. 2021, Sharan et al. 2021).

As previously mentioned, cochleagrams were shown to be the most effective form of audio representation for audio classification tasks. Most inspiration for using a cochleagram to represent audio data comes from the human cochlea. In effect, a cochleagram shows the auditory nerve firing rate (excitation pattern of the basilar membrane in the inner ear of a human) with respect to time and frequency. However, there have been many cochlear models developed in previous research, and many of these have since been implemented into brian2hears, which is a python library that has been used in previous research to construct cochleagrams and other similar graphical representations of audio signals (Ponomarchuk et al. 2022).

More specifically, brian2hears is an auditory modelling toolbox that is primarily used for generating and manipulating sounds, as well as applying large banks of filters (Stimberg et al. 2019). In essence, brian2hears allows us to generate different data representations of an audio signal. For example, by applying custom filter banks to sound data we can model how auditory nerve fibers are excited by the audio signal. The human cochlea

applies “the equivalent of 3000 auditory filters” which causes a computation cost problem for modelers that brian2hears sets out to solve. This python library allows us to implement models with very large numbers of channels and apply thousands of filters to a sound and uses online computation to solve the computational cost issue caused by high audio sample rates and many audio channels. Instead of computing the entire output of each channel of the filterbank, the output of the filterbanks is only stored for a relatively short amount of time. In effect, we are provided with an auditory sandbox that allows us to create custom data representations for any audio signal. Given that we have a sandbox in which we may explore practically infinite permutations of auditory models by employing filter banks with various inspirations, we now have a powerful tool to explore and create different audio representations. However, this online processing schema has its limitations, which will be considered later.

For the creation of a cochleagram in Sharan et al. (2021), gammatone filters were used as the filter bank (Slaney 1993). However, brian2hears also has other common cochlear models/filterbanks such as the approximate gammatone filter (Hohmann, 2002), linear gammachirp filter (Wagner et al. 2009), and logarithmic gammachirp filter (Unoki et al. 2001). Each filter bank has its own set of parameters specific to its implementation and inspiration. For example, gammachirp filterbanks are filterbanks that are designed to capture the nonlinear response properties of the auditory system (such as cochlear compression) more accurately. It is also important to note that some of the representations take additional arguments to specify “glide slope” and “slope constant” properties of the filters, but these were left at the default values as specified by each filter’s motivating research papers. This will be further discussed in the limitations

section, because due to computational complexity and the irreversible nature of converting audio to spectral image data, there are a few practical consequences to incorporating these parameters as trainable parameters in our networks.

While running thousands of auditory filters through the audio signal provides a user with a spectral image at very high resolution, in audio classification far less filters are typically used. Also, many of these more complex cochlear models require additional and sometimes nonlinear computation, and thus have been sparsely used when compared to the cochleagram. The first reason for using less filters (which will be explained more in depth in the experimental design section) is that the number of filters in the filterbank governs the height of the resulting image data. Most pre trained image networks only accept inputs as large as 224x224 in size, to limit the complexity of the network as well as the VRAM needed to store both the batched data and the network architectures. It was shown that generating cochleagrams with high numbers of filters and then downsizing the image yielded very similar performance to simply generating the cochleagram by setting the number of filters to the desired height of the generated image (Sharan et al. 2021). However, acceptable accuracies have been achieved with shallower networks utilizing smaller input sizes, and adding additional parameters via increasing the size of the network inputs and scaling up network architecture may not necessarily improve network performance.

Lastly, it is important to note that the scope of this research will be deep learning in local environments. Many audio benchmarking datasets are relatively small (<10000 files) which makes it difficult to train a scratch CNN to convergence even if the network is allowed to run for thousands of epochs. Scratch networks can drastically underfit or

overfit to such a small training dataset, which limits the network's ability to generalize to data it has not yet seen. Thus, this research is more interested in analyzing pre-trained network architectures that can leverage previous training for new tasks. This decision was motivated by personal experience – all scratch CNN networks (even ones with relatively low complexity in the convolutional base) were unable to obtain similar accuracy levels to the literature that demonstrates their efficacy. Almost all the tested scratch networks were unable to converge, either severely underfitting or overfitting to the dataset depending on the size and complexity of the network architecture.

Thus, transfer learning will be employed as opposed to creating scratch networks. The most effective solution for a team or individual with low compute resources will almost always require transfer learning, especially if the task or data the network is being trained on requires the network to identify complex patterns to correctly classify the data. It was deemed that it would be most appropriate to benchmark each filterbank representation (cochleagram, logarithmic gammachirp, linear gammachirp, and approximate gammatone) across a few popular pre-trained network architectures (EfficientNet-b0, ResNet50, SqueezeNet) to provide comparison points for different combinations of network architecture and data preprocessing. All networks in this experiment will be run on a consumer level NVIDIA 3080 RTX graphics card, with the hopes that this will make the findings of this research more relevant to deep learning researchers to who do not have the resources to purchase large scale compute solutions from third party vendors such as Google or Amazon.

While it is acknowledged that given enough data and time, there almost always exists a slightly more appropriate organization of layers and hyperparameters for a given task, the

aim of this paper is not to achieve a certain threshold score, but rather to compare the performance of different cochlear models across different network architectures to encourage further empirical conversation about reliable and consistent solution designs that apply to various problem domains. These empirical findings will be reproducible on local machines utilizing less-than-state-of-the-art GPU hardware.

Research Questions

In this research, the following questions will be investigated:

1. Which transfer learning networks provide the best performance in terms of accuracy and F1 score?
    a. Do architectures with more parameters and higher computational complexity provide for better results in terms of accuracy and F1 score?
2. Which spectral image representations of audio provide the best performance in terms of accuracy and F1 score?
    a. Which cochlear models are most computationally expensive in terms of generation time?
    b. Do image representations created by cochlear models with higher computational complexity provide for better performance than simpler pre-processing techniques?

While cochleagrams are currently considered one of the best performing image representations for audio classification tasks (Sharan et al. 2021), it will be useful to see if other cochlear models provide similar results with comparable computational cost. The goal is to identify which network architectures and cochlear models perform best on audio classification tasks, and this research will hopefully motivate audio domain experts

to empirically compare characteristics of each cochlear model to determine *why* some outperform others, and perhaps if some cochlear models represent certain sound classes better than others.

Please note that computational complexity is much more straightforward to measure for pre-trained networks than it is for cochlear model pre-processing. This is because network complexity can be measured in the number of trainable parameters. Thus, there was no need to develop a research question to identify the computational complexity of each pre-trained network.

Initially, it was desired to examine whether changing the parameters of the cochlear models before preprocessing the audio into image data would provide for increased performance. However, the iterative nature of this research question ended up making this difficult to investigate. This is discussed much further in depth in the Limitations section of the paper, and a theoretical approach to identifying optimal parameters is proposed. More specifically, it is suggested that these parameters could be made custom trainable parameters in PyTorch, which may allow the networks to learn the suitable values given enough data.

Experimental Design

Dataset Selection

UrbanSound8k was selected as the dataset for the benchmarking experiment. UrbanSound8k is a dataset that "contains 8732 labeled sound excerpts (<=4s) of urban sounds from 10 classes: air conditioner, car horn, children playing, dog bark, drilling, engine idling, gun shot, jackhammer, siren, and street music" (Salamon et al. 2014). The

dataset was created in response to the scarceness of labelled real-world audio datasets and consists mainly of sounds that are frequently the subject of urban noise complaints.

This dataset was selected based on a few criteria. Firstly, the dataset needed to be complex enough that no existing network has achieved near-perfect accuracy, indicating that there is still room for improvement. However, it couldn't be so complex that common transfer learning networks would struggle to fit the data. According to the dataset's profile on Papers With Code, the top four 10 fold average benchmarking accuracies (85.5%, 88.1%, 89%, and 90%) have all been achieved by end-to-end network frameworks, and only one of these networks was trained from scratch solely on UrbanSound8k. End to end networks map the waveform directly to an output and sometimes make use of different architectures than CNNs (although some recent end-to-end networks make use of CNNs in their frameworks). While they achieve state of the art accuracies on many datasets, each one is often highly specialized to the task and dataset at hand and may become more computationally expensive than typical pre-trained image networks. Thus, it was found that UrbanSound8k had sufficient complexity, and that there was room for improvement for non-end-to-end CNN frameworks for this dataset. Additionally, the clips were selected to be a maximum of 4 seconds because of adjacent research that found that 4 seconds of audio was sufficient for subjects to classify environmental sounds with 82% accuracy (Chu & Kuo 2009). Thus, it should be noted that human audio classification accuracy, especially for classes of environmental or background sounds, is far from perfect. This provides an interesting comparison point for our benchmarking, although it is important to note that the subjects were not tested for accuracy on UrbanSound8k itself.

One last important consideration that needs to be made about UrbanSound8k is that the data is provided to the user via 10 pre-split folds. The folds were selected based on "correlation-based attribute selection to avoid overfitting the training data". This training and evaluation process is performed because multiple audio files in the dataset are from the same recordings. Thus, this experiment will use the 10 provided folds as intended by the team that gathered that data because if the data is reshuffled there is a risk that files from the same recording will be used in both the training and validation datasets, which will result in inflated accuracy scores when compared to how the network generalizes to new data once deployed.

Please note that while the dataset contains 8732 data points, 9 data points had to be removed due to invalid/corrupted PCM encoding, which prohibited them from being loaded into the python packages brian2hears and Librosa. Many methods were employed in attempts to convert these WAV files with the correct encoding and format, to no avail. While this disproportionately affected the 8th fold and the children_playing sound class, the number of affected datapoints was so small that it was deemed acceptable to remove these from the dataset because it would not significantly disrupt the overall class balance for any fold.

	slice_file_name	fsID	start	end	salience	fold	classID	class
4803	19007-4-0-0.wav	19007	0.0	4.000000	2	5	4	drilling
6246	36429-2-0-13.wav	36429	6.5	10.500000	2	8	2	children_playing
6247	36429-2-0-14.wav	36429	7.0	11.000000	2	8	2	children_playing
6248	36429-2-0-15.wav	36429	7.5	11.500000	2	8	2	children_playing
6249	36429-2-0-18.wav	36429	9.0	13.000000	2	8	2	children_playing
6250	36429-2-0-23.wav	36429	11.5	15.500000	2	8	2	children_playing
6251	36429-2-0-6.wav	36429	3.0	7.000000	2	8	2	children_playing
6252	36429-2-0-7.wav	36429	3.5	7.500000	2	8	2	children_playing
8338	88466-7-0-0.wav	88466	0.0	1.662041	1	1	7	jackhammer

Table 1: Metadata for the 9 removed data points (due to incorrect or corrupted PCM encoding)

Benchmarked Inputs: Data Representations

General Process

To preprocess each audio file into suitable inputs for a classification network, the same process was utilized to remain consistent across all data representations. All spectral representations will be created in the previously mentioned Python auditory modelling library, brian2hears. Using the Python library Librosa, the audio files were converted from stereo (2 channels) into mono (1 channel) and zero padded to 4 seconds so that all audio files in the dataset have the same duration. Once in this format, the files were processed by the corresponding filterbank functions in brian2hears. For every filterbank, 224 center frequencies and therefore 224 filters are applied to the audio signal, which will guarantee that image will have a height of 224 (this will be explained more in depth in the following sections). Then, each file is split into 224 windows of equal length, using Librosa to apply windowing and a hamming function. The hamming function is necessary because there is significant autocorrelation between adjacent samples in an audio signal. After summing over each frame, the result is a 224x224 greyscale image object, containing the signal energies of the sound at each frequency and time bin. In the

following sections, the four representations benchmarked in this experiment (cochleagram, linear gammachirp, logarithmic gammachirp, approximate gammatone) will be introduced and defined. Exact code and implementation can be found on the GitHub repository.

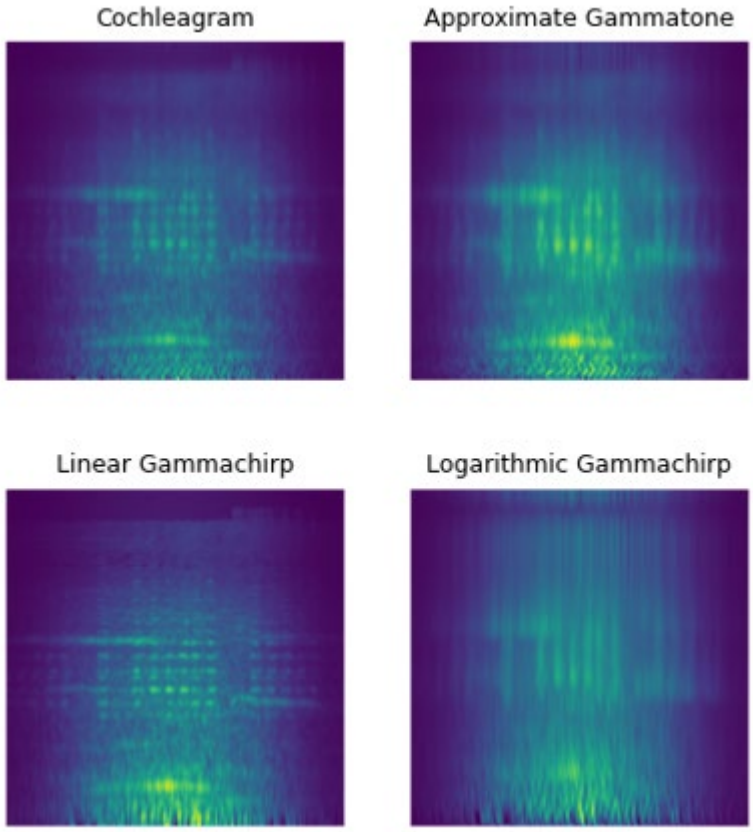

Figure 2: Spectral image representations of a car horn audio signal, as generated in Python library brian2hears

Cochleagram (Gammatone)

Formally presented by (Patterson 1992), cochleagrams are image representations of audio signal that are generated by processing audio signal data using a bank of filters named gammatone filters, which were filters introduced by (Aertsen & Johannesma 1980). These filters were created to mimic the impulse response of the human cochlea and are

commonly used in digital signal processing applications such as aiding the hearing impaired (Soltani-Farani 1998), creating electronic cochlear devices (Leong et al. 2003), and more recently providing inputs for audio classification networks (Sharan 2015, Das et al. 2019, Maka 2019). They are currently the most popular input for classification networks as they are computationally cheaper than other more complex cochlear models and were shown to provide state of the art accuracies when used as inputs to these networks (Sharan et al. 2021).

They are implemented "as cascades of four 2nd-order IIR filters (this 8th-order digital filter corresponds to a 4th order gammatone filter)". This is an important distinction to make, as all representations utilized in this experiment will have equivalent orders for comparison's sake (and because this is how brian2hears has set default arguments for their processing functions). As defined in (Slaney 1993) and implemented in brian2hears, if f is the center frequency in hertz, the approximated impulse response IR and equivalent rectangular bandwidth of the filter are defined as follows:

$$IR(t) = t^3 \exp(-2\pi b ERB(f) t) \cos(2\pi f t)$$
$$ERB(f) = 24.7 + 0.108 f [Hz]$$

An array of center frequencies, f, is constructed using an ERBSpace from 20Hz to 20000Hz, where 224 filters were used to achieve an image height of 224. The parameter that determines the bandwidth of the filters and the duration of the impulse response, b, will be set to its default value of 1.019 as implemented in brian2hears. To compute the exact ERB bandwidth for a given center frequency, parameters ear_order, ear_Q, and min_bw were used as follows:

$$ERB = ((cf/ear\_Q)^{erb\_order} + min\_bw^{erb\_order})^{(1/erb\_order)}$$

These three parameters will remain set at their default values as found in the literature (erb_order = 1, ear_q = 9.26449, min_bw = 24.7). Essentially, cochleagrams have a fixed frequency resolution, that utilizes a constant Q factor to calculate bandwidth at each frequency level. While certain parameters may represent certain sounds more effectively, these comparisons are outside the scope of this research, as the main intent is to simply compare different established representations of audio data.

Once the raw audio signal has been processed, the resulting signal data is then split into 224 frames. The sum of the absolute values is taken for the resulting values in each frame. A negligibly small floor value of 1e-12 is then added to each remaining value for computational purposes, which is common in deep learning and digital signal processing to prevent values of infinity and negative infinity in our gradient computations. Finally, because there is typically autocorrelation present in audio data (i.e. the spectral content in two adjacent frames is usually highly correlated in audio signal data), each of the resulting frames are smoothed with a hamming window with a size of 224. The resulting data object is a 224x224 numpy array, which represents the 1 channel (greyscale) image representation of the processed signal energy values. Exact implementation and a copy of the code can be found on the GitHub repository.

Approximate Gammatone

The approximate gammatone filter is different than the conventional gammatone filter in both mathematical formulation and computational complexity. Approximate gammatone filterbanks have reduced parameterization when compared to gammatone filterbanks, as they estimate the bandwidth of each filter, yielding a more approximate filter shape. This

filterbank sacrifices frequency resolution for quicker computation. Originally presented as a computationally efficient method to approximate the response of the gammatone filter (Hohmann V. 2002), the approximate gammatone filter uses a sum of complex sinusoids with different amplitudes and phases. This is derived from the complex analog gammatone impulse response:

$$g_\gamma(t) = t^{\gamma-1}(\lambda e^{i\eta t})^\gamma$$

Gamma represents the order and therefore the number of approximate IIR gammatone filters applied to the signal. To remain consistent with the cochleagram, we will use a default order value of 4 for the experiment. η corresponds the center frequency and λ is the bandwidth parameter, which is estimated as follows:

$$\lambda = 10^{(0.037+0.785log_{10}(center\_frequencies/Hz))}$$

Like the cochleagram, an array, center_frequencies, is constructed using an ERBSpace from 20Hz to 20000Hz. These frequencies are then used to calculate the bandwidth at each of the 224 center frequencies, which results in 224 filters, and thus, an image height of 224.

Once the sound signal has been processed by the above filterbank, the resulting spectral data is then windowed and summed using the exact processes described in the Cochleagram section. The final output is a 224x224 greyscale image data object. Exact implementation and a copy of the code can be found on the GitHub repository.

Logarithmic Gammachirp

Logarithmic Gammachirp filterbanks are another tool that can be used to create spectral image representations of audio data. The logarithmic gammachirp filter was originally

presented as an improvement over the IIR asymmetric compensation gammachirp filter (Unoki et al. 2001), which was a filter that utilized a feedback loop to change its frequency response to match asymmetric frequency responses of the human cochlea. However, the IIR asymmetric compensation gammachirp filter was limited in that the filter is not able to accurately capture the cochlear response to low-frequency sounds. To approximate the nonlinear frequency response of the cochlea, a logarithmic spacing of the filter's center frequencies is used to allow for more accurate comparisons of filter outputs at different frequencies. It is implemented as "a cascade of 4 2nd-order IIR gammatone filters followed by a cascade of ncascades 2nd-order asymmetric compensation filters", where ncascades is typically set to 4. This is like the cochleagram which was a cascade of four 2nd-order IIR filters but is more computationally expensive and requires additional computation in its processing chain.

As defined in (Unoki et al. 2001) and implemented in brian2hears, if f is the filter's center frequency, b is a parameter that determines the duration of the impulse response, and c is the glide slope or sweep rate in Hz/second, then the filter's impulse response IR and equivalent rectangular bandwidth is defined as follows:

$$IR(t) = t^3 e^{-2\pi b ERB(f) t} cos(2\pi(ft + c \cdot ln(t))$$

$$ERB(f) = 24.7 + 0.108f[Hz]$$

$$f_{instantaneous} = f + c/t$$

An array of center frequencies, f (to be used in calculation of f_instantaneous), was constructed using an ERBSpace from 20Hz to 20000Hz, where 224 filters were used to achieve an image height of 224. This is like the cochleagram, but in this case the center

frequencies from the provided ERBSpace are summed with the glide slope divided by time t. For the experiment, a default glide slope of c = -2.96 will be used to remain consistent with prior literature defining the logarithmic gammachirp filterbank (Unoki et al. 2003) as well as online examples from the brian2hears reference documentation. Thus, f_instantaneous will show a "downchirp" because c<0. Similarly, the parameter that determines the duration of the impulse response, b, will be kept at the default value of 1.81. Thus, this filterbank is made up of nonconstant time varying filters and has an impulse response that varies at a nonconstant rate over time. This nonconstant rate is governed by glide slope.

Once the sound signal has been processed by the above filterbank, the resulting spectral data is then windowed and summed using the exact processes described in the Cochleagram section. The final output is a 224x224 greyscale image data object. Exact implementation and a copy of the code can be found on the GitHub repository.

Linear Gammachirp

Linear Gammachirp filterbanks can also be used to create an image representation of the spectral content of audio. Originally inspired by the auditory responses in the Barn Owl's Nucleus Laminaris (Wagner et al. 2009), these are implemented "as FIR filters using truncated time representations of gammachirp functions as the impulse response".

Because the filters are spaced linearly in frequency, impulse response has the same length for every channel and a duration of 15 times the biggest time constant, sigma. This filterbank differs from the gammatone filterbank in that by introducing a chirp function, the filters are shaped like gammatone filters that have been modified with a chirp function. This introduces a time varying frequency response that has been found in

mammal's auditory system and can provide highly accurate representations of the spectral content of a sound signal at different time intervals. This may allow networks to learn more accurately on both temporal and spectral properties of the sound.

As defined in (Wagner et al. 2009) and implemented in brian2hears, the impulse response is defined as follows:

$$IR(t) = t^3 e^{-t/\sigma} cos(2\pi(ft + c/2t^2) + \varphi$$

In the above impulse response, sigma is the time constant (which determines the duration of the envelope and length of the impulse response IR), c is the glide slope, phi is the phase shift of the carrier, and f is an array of the sweep starting frequencies as expressed below:

$$f_{instantaneous} = f + ct$$

An array of center frequencies, f (to be used in computing f_instantaneous), is constructed using an ERBSpace from 20Hz to 20000Hz, and 224 filters were used to achieve an image height of 224. The glide slope will be kept at the default value of c=0, and thus the f_instantaneous array is always equal to the f array. The time constant sigma is defined by an array of 224 linearly spaced time constant values between 0.3 and 3, as recommended by the literature and the default implementation for brian2hears. Thus, the linear frequency sweep performed by the chirp function is turned off, which reduces the parameterization of the filterbank. This simplifies the filterbank and causes the impulse response to vary at a constant rate over time. Because of this, the linear gammachirp

filterbank is more like the gammatone filterbank than it is the logarithmic gammachirp. However, it is still a time varying filter due to the linearly spaced time variance constants.

Once the sound signal has been processed by the filterbank, the resulting spectral data is then windowed and summed using the exact processes described in the Cochleagram section. The final output is a 224x224 greyscale image data object. Exact implementation and a copy of the code can be found on the GitHub repository.

Benchmarked Networks: Pre-Trained Image Networks

Network Selection

Three popular CNNs trained on image recognition tasks (ResNet-50, EfficientNet-b0, SqueezeNet) were selected to be benchmarked for this experiment. ResNet-50 is a very deep neural network that makes use of residual connections to achieve convergence with many layers. EfficientNet-b0 is the smallest network in a family of networks inspired by resource constraint functions which provide researchers with an intelligent framework under which to scale up their networks in size for additional accuracy. SqueezeNet is a less complex network that was created in order to achieve performance that is comparable to state-of-the-art networks, with far fewer parameters. An added benefit of using these three networks is that they each accept the same input size (224x224x3), while the distinct nature of each architecture provides innate reference points for comparison and thus were deemed suitable for the experiment.

In transfer learning, pre-trained weights are loaded in as starting points for the training process. In this experiment, the NVIDIA 3080 RTX provided enough computational power to train all parameters of each network in a feasible duration, and thus the convolutional base and the classification head were trained in concert and no parameters were frozen in any of the networks. Utilizing the NVIDIA 3080 RTX, EfficientNet and

SqueezeNet takes roughly 30-40 seconds per trained epoch, while ResNet-50 takes roughly 90 seconds per trained epoch. This is to be expected given the size and complexity of the three networks. These will now be discussed more in depth on a technical level.

ResNet-50

Early deep neural networks were often difficult to train, and in studying the effect of adding layers to improve accuracy, it was found that adding additional layers to a suitably deep network leads to higher training error (Srivastava et al. 2015), despite theoretical examples that indicated that a deeper network should produce no higher training error than a shallower network, if it was constructed from said shallower network. This incongruence indicated that current optimization algorithms are commonly unable to find solutions that are comparably good or better to the initial shallow network.

A deep residual learning framework was proposed which resulted in the ResNet network architectures. This framework proposed skip connections, which allow for direct transfer of information from one layer to another. This is performed by residual blocks, which learn residual functions that capture the difference between the input and output of a layer. In a residual block, an input is typically passed through one convolutional layer, then through a nonlinear activation function. This output is passed into a second convolutional layer and is then added to the input (which is passed using a skip connection). This final output is then sent through another nonlinear activation function before moving into another residual block. These skip connections were found to circumvent the accuracy degradation effect of adding more layers, and enabled

researchers to use deeper networks to solve problems. ResNet-50 is an iteration of the ResNet family of networks, which utilizes a total of 50 layers.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 2: The architectures for the ResNet family of networks (He et al. 2015).

For this experiment, the 1000-d fully connected layer was removed from the network and replaced by a fully connected layer with 10 output features, which served as the classification head. Default pre trained weights were used for the convolutional base.

EfficientNet-b0

When Convolutional Neural Networks are developed, this is typically done on a fixed budget and then "scaled up" to achieve higher accuracy/performance. The most common ways to scale up CNNs are by their width, depth, or image resolution (He et al. 2015, Zagoruyko & Komodakis 2016, Huang et al. 2018), but in most cases only one of these dimensions was increased at a time and adjusting these dimensions in concert proved time consuming and sub-optimal. A compound scaling method (Tan & Le 2019) was proposed to provide an empirical roadmap for scaling a CNN to achieve better accuracy:

$$depth = \alpha^{\varphi}$$

$$width = \beta^{\varphi}$$

$$resolution = \gamma^{\varphi}$$

$$s.t.\ \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

Alpha, beta, and gamma are all parameters that can be determined by a grid search. Phi is a coefficient that controls how many resources are available. In the above example, the constraint of 2 signifies that for any phi, the total FLOPS will approximately increase by (2^phi). Using this constraint, the EfficientNet architectures were developed to provide an efficient set of networks to choose from when scaling up the network for a given problem. For instance, EfficientNet-b0 contains only 11 million parameters, while EfficientNet-b7 has roughly 66 million parameters. This provides deep learning researchers with a wide range of possible parameterizations, which can be utilized for problems of differing complexity, both from scratch and using pre-trained weights. The convolutional base consists of all layers in the below diagram except the final fully connected layer in stage 9:

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Table 3: A tabular representation of the EfficientNet-b0 network (Tan & Le 2019). For each stage, the number of channels and layers are listed.

After an initial Conv3x3 input layer, the data is then passed forward through 7 inverted residual blocks with different dimensions. Previously introduced in the ResNet architecture, residual blocks allow direct transfer of information from one layer to another using skip connections, which significantly reduces the computational complexity of the network. These blocks also perform depthwise separable convolution, further reducing the number of parameters required in the network by separating the spatial and channel-wise convolution. While residual blocks use skip connections to link wide layers, inverted residual blocks were introduced in the MobileNet architecture (Howard et al. 2017) and connect narrow layers using skip connections.

Figure 3: A graphical representation of the EfficientNet-b0 network architecture (Tan & Le 2019)

For this experiment, the fully connected layer in stage 9 (FC) was replaced by a fully connected untrained layer with 10 output features, which served as the classification head. For all layers except the new fully connected layer, default pre-trained weights were loaded as a starting point for training.

SqueezeNet

For all problems, there exist both deeper and shallower networks that achieve the same accuracy. Given equivalent accuracy, it was suggested that the network with fewer parameters is advantageous because it provides for more efficient training, reduces overhead when distributing networks, and provides feasibility for FPGA and embedded deployment. This suggestion inspired the SqueezeNet architecture, which was a network architecture that was created to achieve comparable levels of accuracy to state-of-the-art architectures, with many fewer parameters (Iandola et al. 2016).

The SqueezeNet architecture makes use of many network compression techniques, such as reducing the size of the convolutional kernels, decreasing the number of input channels, and downsampling late in the network to provide larger activation maps. The network also makes use of something called a fire module, which is comprised of a squeeze convolution layer (with only 1x1 sized kernels) that feeds into an expand layer which has a mix of 1x1 and 3x3 kernels. In total, the network is comprised of a convolutional layer, followed by 8 fire modules, and finished with another convolutional layer.

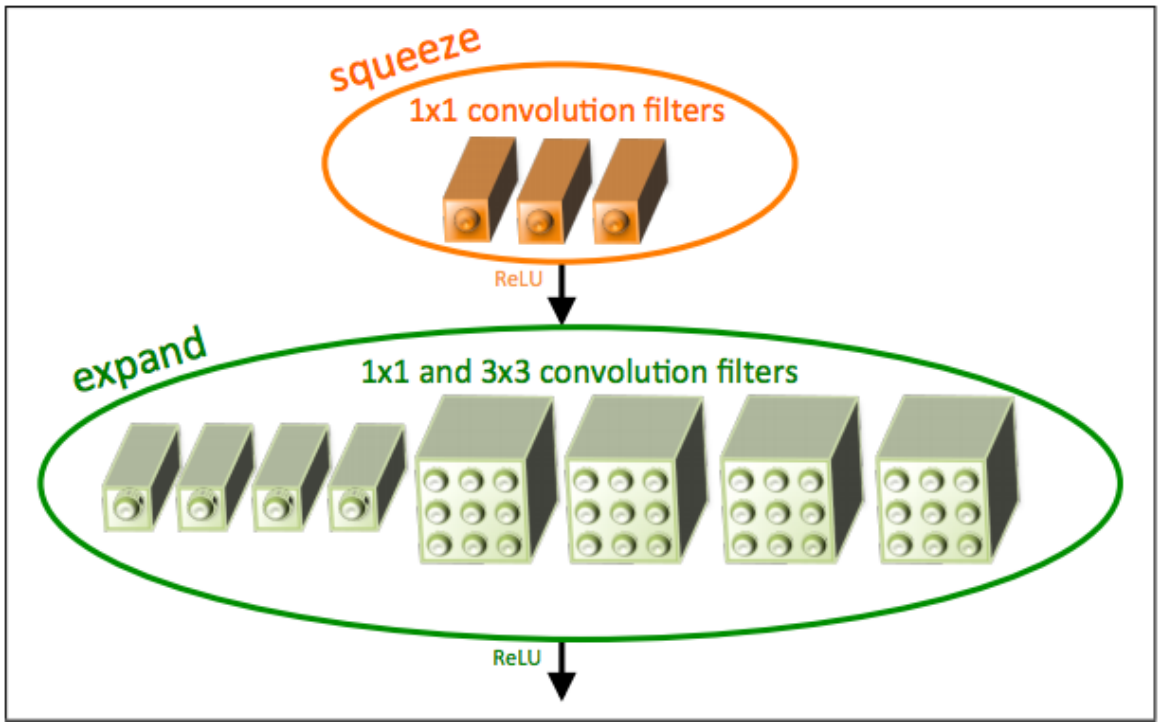

Figure 4: A graphical representation of SqueezeNet's fire modules, made up of a squeeze convolution layer followed by an expand layer (Iandola et al. 2016)

Given that the SqueezeNet architecture was designed to provide a network architecture with low computational complexity, it was selected for this experiment to illuminate how a network with very low complexity performs for an audio classification task. For example, the final classification layer is a convolutional layer with kernel size of 1x1, which differs from the other architectures that make use of linear layers. This convolutional layer provides very low dimensionality and computational complexity for the classification head. Please note that it could be replaced by a linear layer in attempts to achieve higher accuracy, but that doing so will increase the complexity of the network and seems to go against the inspiration for the network itself. The convolutional layer after the dropout layer in the classification head is replaced for this experiment with a similar convolutional layer with 10 output features. For all layers except the

convolutional classification layer, default pre-trained weights were downloaded and used as starting points for the training process.

Data Augmentation

Given that our dataset is comprised of only about 800-900 data points for each of the 10 classes, it is likely that most networks trained from scratch will overfit to the training data and generalize poorly. Accentuating the potential for overfitting is the architecture of the networks commonly used for transfer learning, which are often comprised of millions of parameters and tens if not hundreds of layers. This overfitting problem exists both in image and audio classification, and to combat this a common practice known as Data Augmentation can be employed. Instead of generating synthetic data, the already existing dataset can be randomly modified or transformed in realistic ways to provide the network different contexts under which to learn patterns for the same data points. These transformations include but are not limited to translation (shifting the image by a certain difference over a certain axis), converting color images to greyscale, rotating the image by a certain number of degrees, adding gaussian blur or other artificial noise, and down sampling the image's resolution (Krizhevsky et al. 2012).

It is important to note that many image transformation techniques may not make intuitive sense for an image representation of audio signals. For instance, because our data inputs are the equivalent of a greyscale image object duplicated across 3 channels to simulate a 3-channel input, transforming it to greyscale would do nothing to transform the data. Another example is vertical translation, which if performed on a spectral image of audio, would rearrange the spectral content of a sound, and potentially place signal energies that were originally present in high frequencies in the low frequency area of the image plot.

Conversely, small amounts of horizontal translation would simply have the consequence of shifting the image in the time domain, which is a much more realistic transformation for audio data that preserves spectral content.

Please note that it is up for debate if vertical translation would truly be inappropriate, and that these transformations were chosen due to the intuition behind them with respect to the audio domain, and not raw iterative decisions based on performance. For example, given how a Convolutional Neural Network learns using kernels to filter small sections of the data input, vertical translation may not actually disrupt the network's ability to learn patterns in the data, due to a concept called spatial invariance (LeCun et al. 1998). Because a CNN can learn patterns in the spectral representation regardless of where the patterns occur spatially, vertical translation could be an appropriate choice to combat overfitting. However, as previously stated, decisions for these transformations were made intuitively to err on the side of caution, and when considering interpretability of the outputs of the network, it makes sense to keep the frequency spectrum orthogonal and unaltered.

Other possible techniques to augment audio data (which may be more appropriate than simple image transforms) include but are not limited to adding convolutional reverb to the audio signal to simulate the audio in different physical spaces, compressing the dynamic range of the audio, adding equalization to the audio to remove extraneous low or high frequency information, and using bandpass filters to only allow signal from certain frequency ranges to be represented. While there exist python libraries such as Spotify's Pedalboard (https://spotify.github.io/pedalboard/) to perform this exact function, there exists a major practical problem to performing random audio data augmentation. The

problems mainly center around the computational complexity of creating the image representations combined with the irreversible nature of converting audio to these image formats. For these reasons, audio data augmentation was not used in this experiment, and this will be further discussed in the Limitations section.

The only data augmentation method selected for this experiment was Horizontal Translation. Horizontal Translation is randomly applied with a probability of 50%. Please note that the Horizontal Translation is only applied to the training batches, and the testing batches remained untransformed.

Horizontal Translation (Right Shift)

Horizontal translation was applied to the data to provide each network with a realistic duration transformation of the audio. For instance, an audio file that has heavy spectral content in the first second may be shifted to the right, essentially simulating a data point where the same audio content was simply delayed by some duration of time. Because this is intuitively one of the more realistic translations for many data domains, the maximum possible amount of translation is set to 90% of the image width to allow for a wide range of possible translation amounts. Allowing the right shift amount to be as large as 90% also had the side effect of potentially presenting each network with a partial data point, as some of the tail end of the spectral content would be pushed outside of the image domain (in effect truncating the sound). With 50% probability, a random amount of horizontal translation in between 0% and 90% is applied. Please note that because most audio signals tended to display spectral content in the beginning of the 4 second window and many sounds' durations were extended via padding each sound to 4 seconds in length, only right shift was utilized for this experiment because a left shift of 90% may

push the entirety of a sound's spectral content outside the plot domain. While a left shift could have been employed in smaller amounts, data augmentation was not the major focus of this experiment, and its main purpose was to prevent overfitting. Additional augmentation may have yielded improved results, although this is not guaranteed as it could also be argued that reducing the possible right shift range could have improved results. The iterative nature of this experiment makes it difficult to speculate on the marginal effects of data augmentation.

Standard Hyperparameter Tuning

Hyperparameter Name	Hyperparameter Value
Image Size	224x224x3
Batch Size	32
Training Epochs	20
Loss Function	Cross Entropy Loss
Optimizer	Adam
Learn Rate	0.00005

Table 4: Hyperparameters selected for fine tuning the three pretrained networks.

Hyperparameters were selected for the three transfer learning networks such that they all provided reasonable starting points from previous literature. Because multiple data representation types were benchmarked across multiple network architectures, tuning hyperparameters specifically for each network was avoided in favor of a reasonable set of starting hyperparameters that could be used for all three networks.

An image size of 224x224x3 is used for the input data representations, as this is the specified input structure for all three transfer learning networks used in the experiment. Because data preprocessing generates 224x224x1 greyscale image objects, there are two

possible solutions: one involves replacing or changing the dimensionality of the input layer of each transfer learning network to fit the input size, and the other involves duplicating the greyscale image object across three channels. Because we do not want to destroy the pre-trained weights of the network, especially in the initial layers, the second solution was employed in this experiment.

While typically it has been shown that larger batches help a network learn faster (Krizhevsky 2014, Goyal et al. 2017), there are a few considerations that need to be made about this finding. For one, it was found that there are diminishing returns to batch size, and that eventually a large enough batch size will degrade the network's ability to generalize (Krizhevsky 2014, Keskar et al. 2016, Hoffer et al. 2017). Computationally, batching becomes VRAM intensive quickly with an image size of 224x224x3. Batches larger than 40 were deemed to be unfeasible given the 10 GB VRAM limit, mainly because the networks and their pretrained weights also had to be stored in VRAM simultaneously. Given that the architectures for each transfer learning network were fixed, batch size had to be reduced accordingly. Additionally, because the networks have been trained on massive amounts of data and already have pre-trained weights, we do not need much of a boost from batch size to achieve acceptable accuracy.

Important to note is that network learning is also heavily dependent on the optimizer algorithm selected. This algorithm determines how the network will optimize the loss function. Adam has been shown to provide quick convergence in relatively few epochs (Kingma & Ba 2014), while its widely used alternative Stochastic Gradient Descent (SGD) provides slower convergence but has been shown in some previous research to provide better generalization for a network. It was also shown that if a problem has

multiple global minima (as they often do), different algorithms can and will find different solutions when started from the same starting point (Wilson et al. 2017). While these findings were in support of previous research that proved SGD is uniformly stable for strongly convex loss functions and thus has more optimal generalization potential (Hardt et al. 2016), this topic has been of much debate in recent years. Recent findings challenge these criticisms of adaptive optimization algorithms and assert that more general optimizers should be able to approximate more simple optimizers such as SGD with some set of optimal hyperparameters. They showed that by extending the hyperparameter search space, the adaptive optimizers converged faster and were comparable to SGD in terms of generalization (Choi et al. 2019). One interesting finding from this experiment was that optimal hyperparameters for the optimizers varied largely between datasets, which will be discussed later in the Limitations section.

Because we are training pre-trained networks for relatively few epochs, Adam was the optimizer that was selected for this experiment. Suggested values for learning rate have been found to be anywhere between 0.01 for shallow networks, and 0.0001 for very deep networks (Nakamura et al. 2021, Sharan et al. 2021). In the case that the learning rate is too high, and the optimizer makes large adjustments to layer weights, the network can destroy its learning from previous epochs if left to train too long. It was found, in correspondence with previous literature, that small learning rates provided the best learning for each network. This intuitively makes sense because each of the three networks are relatively deep, and all use pre-trained weights as starting points. Thus, a learning rate of 0.00005 was selected for this experiment, which is consequently half the suggested learning rate for very deep networks. This was done to account for both the

size of the networks and the fact that they have been pre-trained on different tasks, the latter of which was the reason for the additional reduction.

An additional technique that was employed in this experiment was network checkpointing. Essentially, network weights are saved and then overwritten anytime the validation loss decreases to a new minimum value. This is very similar to early stopping, which is a technique that stops a network from further training if a certain metric does not improve after a set number of epochs. Checkpointing differs in that it allows the training process to continue to train for the full 20 epochs because there is a chance that even though the network does not learn for a few epochs, it will breakthrough a local minimum to a more accurate solution. The reasoning is simple – even though the network may de-learn in later epochs, we do not stop it from trying in the case that it does breakthrough a local minimum. We will always be left with final saved network weights that reflect the network at its most learned state, measured by validation loss.

Experimental Data Collection

Each pair of network and data representation was tested using 10-fold validation. In K-Fold validation, the kth fold is selected as the training split, while all other folds are used for the training split. This is performed for all K folds, such that for each network and data representation there will be K trained networks. The performance of these K networks can then be analyzed in concert using support weighted average F1 score, macro average accuracy, and other measures such as approximately how many epochs were needed to achieve the best checkpointed network and graphs of the training and test loss. Thus, after all pairs of data representation type and network are run, the result will be 3*4*10=120 trained networks, which yields a total of 120*20 = 2400 trained epochs.

Additionally, the computation time of each data representation was measured. The generation of each data representation type was timed for Fold 5 to provide a control for comparison. This was replicated 4 times per data representation and will provide insight into the computational complexity of each data representation, as well as intricacies about its specific implementation in brian2hears.

Using these metrics together, we can hope to make some high-level empirical comparisons about useful starting points for deep learning researchers. The limitations section will contain a discussion about the difficulties of making empirical conclusions in iterative fields such as deep learning.

Findings & Results

Preprocessing Time

Data Preprocessing Time (Seconds)				
Replicate	**Cochleagram**	**Linear Gammachirp**	**Logarithmic Gammachirp**	**Approximate Gammatone**
1	3104.70	3977.10	6053.60	3285.40
2	3113.80	3954.30	6110.80	3315.70
3	3109.40	3928.00	6072.10	3338.60
4	3096.20	3983.70	6188.70	3420.10
Number of Sounds (Fold 5)	935	935	935	935
Average Preprocessing Time (Seconds)	3106.03	3960.78	6106.30	3339.95
Per Data Point (Seconds)	3.32	4.24	6.53	3.57
Total Dataset (Seconds)	28977.39	36951.70	56968.19	31159.77
Total Dataset (Hours)	8.05	10.26	15.82	8.66

Table 5: Preprocessing time required to generate each data representation type

After testing each technique on a randomly selected fold of the data (fold 5), the Cochleagram was found to be the least costly data representation, followed by the Approximate Gammatone, Linear Gammachirp, and Logarithmic Gammachirp (in order of increasing complexity). One surprising thing to note is that the approximate gammatone is supposed to be a lower cost representation, but due to actual implementation and application of the filterbanks (which can be found on the GitHub repository) it took slightly longer to pre-process the data using approximate gammatone filterbanks. The Linear Gammachirp filterbank is slightly more costly in terms of generation time, while the Logarithmic Gammachirp is the most expensive pre-processing technique, taking more than twice the time of the Cochleagram and 1.5x more time than the Linear Gammachirp.

It is unknown why the approximate gammatone technique took longer than the cochleagram technique, despite its reputation as being more computationally efficient. This shows that despite what previous literature has shown, different implementations in different programming languages can cause the preprocessing cost to change. Brian2hears is hardcoded in Python, and thus is not compatible with C++ standalone nor the CUDA framework. Also, while the example scripts in the brian2hears documentation suggests that the approximate gammatone takes roughly 0.1 seconds less than the cochleagram for a generated white noise sound of 0.1 seconds, there is no guarantee that either of these preprocessing techniques will scale linearly with audio signal length. The code utilized to perform the preprocessing can be found on the GitHub repository.

In essence, this exposes an area of research that needs further testing – the actual implementation of each preprocessing technique appears to be a main driver in

computational complexity. While utilizing a less complex technique will save a researcher time, this is not guaranteed, especially when the techniques are very similar. If one wanted to reduce preprocessing time for the Cochleagram in this case, it would be more advisable to investigate implementations that are compatible with C++ standalone, or even parallelize the image generation using a Python library like 'multiprocessing'. Utilizing the approximate cochleagram may not be an effective approach, especially if the researcher is using brian2hears to create data representations.

Benchmarking Performance

10-Fold Top 1 Average Accuracy				
Network	**Cochleagram**	**Linear Gammachirp**	**Approximate Gammatone**	**Logarithmic Gammachirp**
EfficientNet-b0	80.32%	80.89%	79.21%	72.22%
ResNet50	80.76%	81.51%	76.61%	70.11%
SqueezeNet	70.75%	75.17%	70.42%	62.94%
10-Fold Top 1 Double Weighted Average F1				
Network	**Cochleagram**	**Linear Gammachirp**	**Approximate Gammatone**	**Logarithmic Gammachirp**
EfficientNet-b0	80.31%	80.83%	79.05%	72.01%
ResNet50	80.61%	81.40%	76.23%	69.78%
SqueezeNet	70.36%	74.83%	70.03%	62.09%

Table 6: Average accuracy and F1 Score, weighted by class and fold support, for each combination of data input type and pre trained network architecture

Firstly, when comparing the performance of the different network architectures, EfficientNet-b0 and ResNet50 achieved very similar results, while SqueezeNet failed to achieve a comparable performance. This confirms previous research that asserts that EfficientNet-b0 provides researchers with a network that achieves similar accuracy to popular networks like ResNet50, despite having 12 million fewer parameters. Interestingly, EfficientNet-b0 showed better results for the data representations that performed most poorly, although this is not a statistically significant difference. Still, its

performance using the Approximate Gammatone data input is interesting, because it appears as though the EfficientNet-b0 architecture may be less sensitive to changes in data input than other networks. Whether this is a function of the architecture, characteristics of the dataset, characteristics of the problem domain, or the pre-trained weights themselves, remains to be investigated.

When comparing the performance of the different data representations, the Cochleagram and Linear Gammachirp filterbanks perform the best across the different pre-trained networks. This is surprising because the cochleagram technique is less computationally complex than both the linear and logarithmic gammachirp filterbanks. This is also not surprising in a way, because the relative simplicity of the cochleagram (when compared to other cochlear models) makes it one of the most accessible spectral representations of audio.

One interesting comparison to note is that the data representations with impulse responses that varied at a constant rate with time were consequently the ones that performed best (Cochleagram, Linear Gammachirp). When comparing the gammatone, linear gammachirp, and logarithmic gammachirp, we see that glide slope has a significant effect on performance, while the time variance constants utilized in the linear gammachirp did not. However, perhaps the default values as found in motivating literature are simply not optimal for every problem in the sound event detection problem domain. This may disproportionately affect representations that take more specific arguments from the researcher, affecting its efficacy out of the box. We see the effects of this when comparing the logarithmic and linear gammachirp filterbanks. For example, setting glide slope to 0 simplified the linear gammachirp and yielded results similar to the even

simpler gammatone filterbank, but utilizing a nonzero glide slope would likely yield different results. Optimal parameterization also likely varies depending on which model architecture is used. This will be further discussed in the limitations section.

There are a few reasons why SqueezeNet may have underperformed when compared to EfficientNet-b0 and ResNet50. For one, SqueezeNet only has roughly 1.25 million parameters, which is roughly ten times as small as EfficientNet and roughly twenty times less than ResNet50. It is possible that audio classification tasks are simply too complex for this network to reach an acceptable solution. Also, it should be noted that when utilizing the Linear Gammachirp technique to create inputs for SqueezeNet, the network was able to achieve 75.17% accuracy, which is very comparable to ResNet50's performance using Approximate Gammatone filterbanks. Thus, if a researcher is attempting to balance performance with cost, it seems that there are many different input-network combinations that could be utilized to achieve comparable results. These results provide evidence that can be leveraged in making decisions on input type and network architecture and may also be used in the case that a researcher wants to reduce or scale up the complexity of the entire solution (input + network).

Limitations and Future Research Opportunities

While benchmarking provides for good empirical starting points when it comes to the selection of network architecture, hyperparameters, and data representation for audio classification, the iterative nature of using advanced networks to solve classification problems severely limits a researcher's ability to make definitive conclusions that generalize to other problem domains. Popular transfer learning networks are often very

deep, with millions of parameters and many layers stacked on top of each other, making stepwise interpretation of the network's learning almost impossible.

Speaking strictly on network architecture, each network has its own inspiration for passing information forward to future layers. For instance, ResNet50 and SqueezeNet make use of skip connections and residual blocks, while EfficientNet makes use of a developed resource constraint to efficiently scale its architecture to handle more complex problems. This is accentuated by the fact that because each network architecture is unique, the theoretical set of hyperparameters for each is also unique. For control's sake, the hyperparameters were kept constant across each pair to provide comparisons for network performance. However, the pairwise comparisons must be interpreted cautiously. While we can compare certain metrics for different pairs of network architectures and data representation, there is always the consideration that perhaps a different set of hyperparameters would have yielded drastically different results. Also, these conclusions only apply to the audio classification problem domain, and thus using a speech database instead of an environmental sound database may have yielded different conclusions. Herein lies the main difficulty of making empirical conclusions about deep learning algorithms and their inputs.

The dataset also has its own limitations. For instance, the team that created UrbanSound8K states that "an urban sound taxonomy should…aim to be as detailed as possible, going down to low-level sound sources such as car horn and jackhammer" (Salamon et al. 2014). Upon closer analysis of the classes, this requirement is fulfilled for most, but not all, classes. The two classes "Children Playing" and "Street Music" appear to violate this idea and expose an interesting abstraction and encoding problem

that occurs with the detection of complex sound events. The sounds of children playing could incorporate many different events including the sound of toys or sports equipment, laughing, talking, breathing, and even clapping. If the children playing are singing, this presents a troublesome overlap with the street music class, which itself is made up of sounds including people singing and playing instruments, people talking in the street, and perhaps even atonal musical chanting. For many folds, it comes as no surprise that the F1 score for children playing and street music classes are relatively low when compared to other classes, across all network architectures, data folds, and data representation types. However, all networks and data representation combinations had even more trouble identifying sounds with significant spectral overlap. These classes include Air Conditioner, Drilling, Jackhammer, and Engine Idling.

This is hypothesized to be partly a side effect of using a general benchmarking dataset instead of a more focused problem domain dataset. For example, if the network was focused solely on the four commonly confused classes and more data was gathered for each class, networks may be able to accurately distinguish between them with more learning. Also, if the classes were encoded differently, or perhaps if the classification network was allowed to give multiple answers instead of just one prediction, we might be able to come up with a schema that more accurately classifies these sounds. While theoretically a large enough network should be able to fit to any problem domain with enough data as number of epochs increases to infinity, it stands to reason that a more focused dataset may eliminate network confusion. Also, there is no guarantee that a larger network will converge to a solution in a reasonable time frame, nor a guarantee that it will ever converge at all.

Practically, unless benchmarking is the goal of the research, a specific problem domain will often yield a more specific database. Consider a problem where you would like to diagnose malfunctioning car parts by the sounds they are emitting. If the dataset also contains classes irrelevant to the problem domain, such as dogs barking, does this reduce the performance of the network in classifying the other relevant sounds? The problem scope of UrbanSound8K is to classify sounds that are most responsible for noise pollution, but perhaps a more limited scope or utilizing or democratizing multiple networks trained on subsets of the same database would yield better solutions. However, it is important to note doing so might reduce your separated datasets to a size that will cause a network to underperform.

The computational complexity of the cochlear models employed in this research is another large limitation that needs to be acknowledged. As was seen in the findings section, the least complex cochlear models took about 3.322 seconds per data point. Thus, it would take roughly 28,977.39 seconds (8.05 hours) to preprocess the entire dataset. In contrast, the most complex model (Logarithmic Gammachirp) takes 6.531 seconds per data point or approximately 56,968.187 seconds (15.82 hours). This is problematic for a few reasons.

Firstly, it implies that any changes to the data preprocessing schema costs a minimum 8.05 hours in preprocessing time. As seen previously, many of the cochlear models have parameters such as glide slope, variance time constants, and the topology of the ERBSpace. Unfortunately, exploring different parameters becomes a slow and painful chore and thus defaults were employed at the suggestion of previous literature. There is no guarantee that these default settings are optimal, or even suitable. One will only find

this out after iterating across the different settings for each representation, and then training the network on these representations to evaluate performance.

The second and perhaps more severe implication of the time cost associated with this preprocessing is that the preprocessing cannot be made a part of the training loop, as doing so would multiplicatively add 8 hours per each training epoch. There are a few reasons why a researcher would and would not want to make preprocessing part of the training loop. From a computation standpoint, it is preferable to preprocess the data once, and save the preprocessed data into a flat data frame for later use in training. Thus, the time cost is only incurred once, and the training and preprocessing loops are independent of each other. However, this is more elastic for image recognition than it is for audio recognition. As mentioned previously, once the audio data has been preprocessed by each cochlear model, the resulting data object *cannot be converted back into audio*, and thus is considered a destructive process. This has a hidden technical consequence – we cannot perform random data augmentation using audio techniques such as reverb, compression, or delay unless the preprocessing is inside the training loop. Any random audio augmentation must be performed before the audio file is processed into image data by the cochlear model.

If computational complexity was not an issue, it would be ideal to put the audio preprocessing inside the training loop of the network. It would allow us to create a custom PyTorch dataloader that would accept an audio file, apply random audio augmentations, then apply cochlear filterbanks, and finally apply random image augmentations before passing the resulting data object through each layer. Because the preprocessing would occur inside the network, this has an exciting implication – the

scalar parameter values for the cochlear models could be converted into custom trainable parameters in PyTorch. This would require porting each cochlear model into PyTorch, and it is unknown if similar generation times would be achieved in PyTorch. An example of what this would look like has already been unofficially implemented for the cochleagram but was not examined in this experiment (Feather et al. 2022). While there is no guarantee this would be successful in improving the network's performance, it is interesting to wonder if we can offload the cost of iteration onto the network itself. Consider a network that utilizes a logarithmic gammachirp filterbank. The glide slope, time constants, and ERBSpace are all hardcoded in this research and based on default values, but perhaps allowing the network to learn suitable values for these parameters will yield higher network performance. It will also yield added interpretability, as we may be able to empirically conclude that certain values for certain parameters work better for different classes of sounds with distinct spectral properties.

Unfortunately, this is currently not feasible because brian2hears does not make use of the CUDA framework and is run solely on the CPU. Upon asking about the possibility of porting it to CUDA on the brian2hears github page, Marcel Stimberg stated that "brian2hears is a bit of an orphan project at the moment, i.e. it works but no one is working on improving it further. Unfortunately, it uses hard-coded Python to implement filters and the like, so it is not compatible with C++ standalone mode, and therefore also not compatible with brian2cuda. We'd definitely like to change this at some point in the future, but I wouldn't expect it to happen anytime soon" (Stimberg 2022 - https://github.com/brian-team/brian2cuda/issues/307). This also raises a key consideration for the audio domain specifically – because the cochlear processing was the

main bottleneck of this experiment, Python may not be the optimal programming language for audio classification tasks. This does not necessarily apply to image recognition tasks, because image preprocessing often requires less computational overhead and image datasets will already contain data points in the desired image format. A more computationally efficient compiled programming language such as C++ or Julia could be used to speed up the data preprocessing and create networks with custom training loops, which may be more desirable than further parallelizing functions in a hardcoded Python library.

While using a different auditory toolbox may be recommended in the case that a researcher wants to use a less complex model such as the spectrogram or mel-spectrogram, some of the cochlear models used in this experiment have not been implemented elsewhere. Notably, brian2hears can simulate an entire auditory nerve fiber model, which is a mathematical model that simulates the response of auditory nerve fibers to sound stimuli. While this is computationally prohibitive (it was found to take roughly one minute per data point), it is exciting to think that there are still more complex cochlear models that exist or will exist in the future. These more complex models could be used as data representations for audio classification if computational power continues to increase, or if implementation of these models becomes more computationally efficient.

Conclusions

In the end, these results can be used to derive suitable starting points for researchers attempting to solve an audio classification problem. For network selection, it is suggested that researchers start with the pre-trained EfficientNet-b0, because it is

virtually tied for best performance with ResNet50, while only containing less than half the parameters of ResNet50. In the case that the network is unable to converge to an acceptable solution, it can be scaled up to more complex iterations of the EfficentNet (b1, b2, …, b7). While the same can be said for the ResNet family of networks, it can be confidently concluded that EfficientNet scales more efficiently, as it makes use of constraint formulas to apply additional resources via a more efficient method of scaling up network architecture. This is supported by its performance, only requiring half the parameters of ResNet50 to achieve the same performance. EfficientNet-b0's performance also appears to be slightly less sensitive to different data input types than ResNet50. For data input selection, it is suggested that researchers continue to use Cochleagrams for a few reasons. While Linear Gammachirp filterbanks provided for comparable (if not slightly better) performance across the different pre-trained networks, it becomes significantly costlier for larger datasets, while the incremental benefits of using Linear Gammachirp filterbanks over the Cochleagram is statistically negligible.

The results of this experiment can also be used to make decisions in the case that a final solution (input + network) needs to be scaled up or down. In the case that an acceptable solution is too resource intensive to officially deploy, one may be able to identify a less complex combination of input and network architecture that provides comparable results while requiring less parameters and lower pre-processing cost. For instance, we saw that EfficientNet-b0 trained on the Approximate Gammatone data representation achieved comparable results to ResNet50 trained on Cochleagrams. In this case, the latter is a more costly solution than the former, but they perform similarly. If the sacrifice in accuracy from utilizing the less complex network and data representation is acceptable in

a certain problem domain, a researcher could attempt to save resources and time by employing the solution with less complexity.

The field of sound event detection continues to be hampered by the iterative nature of deep learning and prohibitively arduous data preprocessing. While benchmarking provides good empirical starting points for audio classification problems, it is still very difficult to make empirical conclusions on how to improve network performance. As computational capacities improve, researchers should continue to automate this iterative process as much as possible to provide a faster exploration of the problem domain. Researchers should also continue to identify, implement, and benchmark different ways to represent audio data for CNNs and other advanced network architectures such as transformer networks or more specific end to end networks to provide a wider base of evidence for more generalized conclusions. Finally, researchers should investigate whether allowing the network to train the parameters of the data representation itself would yield better network performance.

Special Thanks

I want to first thank Dr. Dustin White, Dr. Seth Shafer, and Dr. Mahbubul Majumder for being active members on my committee.

When this project began back in June of 2022, Dr. White encouraged me to pick a topic I was excited about, no matter what the startup prerequisite cost would be and has since held routine meetings with me to track and support my progress. Dr. White acted as my mentor for my years at UNO, fielding my numerous and sometimes downright wacky questions. His courses at UNO instilled in me a love for modelling phenomena of all types, and he has an amazing knack for matching theory with application.

Dr. Shafer was instrumental in answering my audio-based questions and pointing me to great references, and without his expertise in the audio engineering domain I'm sure my analysis and understanding of the subject would be lacking. These references included conference proceedings and presentations that I probably would not have found otherwise.

Dr. Majumder was helpful in identifying key modeling issues in my proposal, primarily centering around how the image representations would represent patterns in my data. It was because of his questions that I decided to go with a zero-padding technique before windowing and downsampling audio, as he identified that I may be underrepresenting spatial patterns in my data by not being more careful to preserve these spatial patterns by padding the files to a constant size first. Interestingly, this intuition was validated, and zero padding provided better results in practice.

I would like to thank Dr. Roneel Sharan for answering some of my conceptual questions regarding his 2021 paper, entitled "Benchmarking audio signal representation techniques for classification with convolutional neural networks", which provided the main inspiration for my research. Especially early in my understanding of the topic, Dr. Sharan took the time to explain the rationale behind a few decisions he made for his paper, which was instrumental in my journey to understand the field myself. Without his personal anecdotes, it would have been much more difficult for me to make sense of concepts like windowing and downsampling, why he used a shallower network and smaller data inputs, and how that related to the dataset used in his research.

I would like to thank the team who develops and maintains the brian2 family of python packages – brian2, brian2hears, and brian2cuda. Dr. Dan Goodman answered a few of

my questions regarding the implementation of cochlear models in brian2hears, and how it compared to other auditory toolboxes. Dr. Marcel Stimberg and PHD Student Denis Alevi engaged with me on the possibilities of porting brian2hears to CUDA frameworks and helped provide some higher-level understanding of the art of the possible regarding brian2hears and audio classification using python packages. I firmly believe their python package is *groundbreaking* in every sense of the word and I can only hope that brian2hears gets its due diligence one day. I can confidently say I didn't even scratch the surface regarding the capabilities of the brian2 family of libraries.

Lastly, I would like to thank my girlfriend Rachel Stack, and my parents Bradley Hammitt and Cynthia Hammitt, for encouraging me to go back to school and providing me with the resources and support to reach my full potential. I love you so much! The time commitment has been rigorous at times, but each of you continued to make sacrifices and concessions at my behest – no questions asked. No matter how many times they didn't understand, they continued to ask about what I was doing and listened while I rambled on and on.

This project is a direct manifestation of my love for learning and the support of everyone previously mentioned. I put my heart and soul into this, and I hope it shows. If something is to be done, you must simply get on with it. Gather the minimum pre-requisites and go! If it is truly important to you, you will make it happen.

References

Aertsen, A., & Johannesma, P.I. (1980). Spectro-temporal receptive fields of auditory neurons in the grassfrog. *Biological Cybernetics, 38*, 223-234.

Allen, J. (1977) Short Term Spectral Analysis, Synthesis, and Modification by Discrete Fourier Transform. IEEE Transactions on Acoustic, Speech and Signal Processing, 25, 235-238.
http://dx.doi.org/10.1109/TASSP.1977.1162950

Allen, J. B. (1982). Applications of the short time Fourier transform to speech processing and spectral analysis. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, *1982-May*, 1012-1015.
[1171703]. https://doi.org/10.1109/ICASSP.1982.1171703

Anvarjon, T., Mustaqeem, Choeh, J.Y., & Kwon, S. (2021). Age and Gender Recognition Using a Convolutional Neural Network with a Specially Designed Multi-Attention Module through Speech Spectrograms. *Sensors (Basel, Switzerland), 21*.

Balodi, T. (n.d.). *Convolutional neural network with python code explanation: Convolutional Layer: Max Pooling in CNN*. Analytics Steps. Retrieved March 24, 2023, from https://www.analyticssteps.com/blogs/convolutional-neural-network-cnn-graphical-visualization-code-explanation

Brown, R.G. Smoothing, Forecasting and Prediction of Discrete Time Series; Dover Publications: Mineola, NY, USA, 2004.

Choi, K., Fazekas, G., Sandler, M.B., & Cho, K. (2017). Convolutional recurrent neural networks for music classification. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2392-2396.

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., & Dahl, G. E. (2019). On empirical comparisons of optimizers for deep learning. arXiv preprint arXiv:1910.05446.

Chu, S.M., Narayanan, S.S., & Kuo, C.J. (2009). Environmental Sound Recognition With Time–Frequency Audio Features. *IEEE Transactions on Audio, Speech, and Language Processing, 17*, 1142-1158.

Costa, Y.M., Oliveira, L., & Silla, C.N. (2017). An evaluation of Convolutional Neural Networks for music classification using spectrograms. *Appl. Soft Comput., 52*, 28-38.

Das, S., Pal, S., & Mitra, M. (2019). Supervised model for Cochleagram feature based fundamental heart sound identification. *Biomed. Signal Process. Control., 52*, 32-40.

Davis, S. and Mermelstein, P. (1980) Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE Transactions on Acoustics, Speech and Signal Processing, 28, 357-366. http://dx.doi.org/10.1109/TASSP.1980.1163420

Feather, J., Leclerc, G., Mądry, A., & McDermott, J. H. (2022). Model metamers illuminate divergences between biological and artificial neural networks. bioRxiv.

Gemmeke, Jort & Ellis, Daniel & Freedman, Dylan & Jansen, Aren & Lawrence, Wade & Moore, R. & Plakal, Manoj & Ritter, Marvin. (2017). Audio Set: An ontology and human-labeled dataset for audio events. 776-780. 10.1109/ICASSP.2017.7952261.

Goyal, P., Dollár, P., Girshick, R.B., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., & He, K. (2017). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *ArXiv, abs/1706.02677*.

Hardt, M., Recht, B., & Singer, Y. (2016, June). Train faster, generalize better: Stability of stochastic gradient descent. In International Conference on Machine Learning (pp. 1225–1234). PMLR.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778.

Hoffer, E., Hubara, I., & Soudry, D. (2017). Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *ArXiv, abs/1705.08741*.

Hohmann, V. (2002). Frequency analysis and synthesis using a Gammatone filterbank. Acta Acustica united with Acustica. 88. 433-442.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv, abs/1704.04861*.

Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q.V., & Chen, Z. (2018). GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. *ArXiv, abs/1811.06965*.

Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W.J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *ArXiv, abs/1602.07360*.

Irino T, Patterson RD. A Dynamic Compressive Gammachirp Auditory Filterbank. IEEE Trans Audio Speech Lang Process. 2006 Nov;14(6):2222-2232. doi: 10.1109/TASL.2006.874669. PMID: 19330044; PMCID: PMC2661063.

Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P.T. (2016). On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *ArXiv, abs/1609.04836*.

Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.

Krizhevsky, A, Sutskever, I, Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012: 1106-1114

Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *ArXiv, abs/1404.5997*.

Lecun, Y, Bottou, L, Bengio, Y, and Haffner, P "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

Lopez-Poveda, Enrique & Meddis, Ray. (2002). A human nonlinear cochlear filterbank. The Journal of the Acoustical Society of America. 110. 3107-18. 10.1121/1.1416197.

Maka, T. (2019). Computationally Efficient Classification of Audio Events Using Binary Masked Cochleagrams. *International Conference on Conceptual Structures*.

Mustaqeem, & Kwon, S. (2021). Optimal feature selection based speech emotion recognition using two-stream deep convolutional neural network. *International Journal of Intelligent Systems, 36*, 5116 - 5135.

Nakamura, K., Derbel, B., Won, K., & Hong, B. (2021). Learning-Rate Annealing Methods for Deep Neural Networks. *Electronics*.

Nanni L, Maguolo G, Brahnam S, Paci M. An Ensemble of Convolutional Neural Networks for Audio Classification. *Applied Sciences*. 2021; 11(13):5796. https://doi.org/10.3390/app11135796

O'Shea, K. & Nash, R. (2015). An Introduction to Convolutional Neural Networks.. *CoRR*, abs/1511.08458.

Patterson, R.D., Robinson, K., Holdsworth, J.L., McKeown, D., Zhang, C.Q., & Allerhand, M. (1992). Complex Sounds and Auditory Images.

Piczak, Karol (2015), "ESC: Dataset for Environmental Sound Classification", https://doi.org/10.7910/DVN/YDEPUT, Harvard Dataverse, V2

Piczak, Karol (2015), "Environmental sound classification with convolutional neural networks," *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2015, pp. 1-6, doi: 10.1109/MLSP.2015.7324337.

Ponomarchuk, A., Burenko, I., Malkin, E., Nazarov, I., Kokh, V., Avetisian, M., & Zhukov, L. (2022). Project Achoo: A Practical Model and Application for COVID-19 Detection From Recordings of Breath, Voice, and Cough. *IEEE journal of selected topics in signal processing*, *16*(2), 175–187.

Rippel, O., Snoek, J., & Adams, R. P. (2015). Spectral representations for convolutional neural networks. *Advances in neural information processing systems*, *28*.

Salamon, J., Jacoby, C., & Bello, J.P. (2014). A Dataset and Taxonomy for Urban Sound Research. *Proceedings of the 22nd ACM international conference on Multimedia*.

Salekin, A.; Eberle, J.W.; Glenn, J.J.; Teachman, B.A.; Stankovic, J.A. A weakly supervised learning framework for detecting social anxiety and depression. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. **2018**, 2, 81.

Sharan, R.V.; Abeyratne, U.R.; Swarnkar, V.R.; Porter, P. Automatic croup diagnosis using cough sound recognition. IEEE Trans. Biomed. Eng. **2019**, 66, 485–495.

Sharan, R. V., & Moir, T. J. (2015). Subband time-frequency image texture features for robust audio surveillance. *IEEE Transactions on Information Forensics and Security*, *10*(12), 2605-2615. [7206602]. https://doi.org/10.1109/TIFS.2015.2469254

Sharan, R. V., & Moir, T. J. (2019). Acoustic event recognition using cochleagram image and convolutional neural networks. *Applied Acoustics*, *148*, 62–66. https://doi.org/10.1016/j.apacoust.2018.12.006

Sharan RV, Xiong H, Berkovsky S. (2021). Benchmarking Audio Signal Representation Techniques for Classification with Convolutional Neural Networks. *Sensors*. 2021; 21(10):3434. https://doi.org/10.3390/s21103434

Sharan, R.V.; Moir, T.J. Time-frequency image resizing using interpolation for acoustic event recognition with convolutional neural networks. In Proceedings of the IEEE International Conference on Signals and Systems (ICSigSys), Bandung, Indonesia, 16–18 July 2019; pp. 8–11.

Singh, B. R., Zhuang, H., & Pawani, J. K. (2021). Data Collection, Modeling, and Classification for Gunshot and Gunshot-like Audio Events: A Case Study. *Sensors (Basel, Switzerland)*, *21*(21), 7320. https://doi.org/10.3390/s21217320

Slaney, M. (1993). An Efficient Implementation of the Patterson-Holdsworth Auditory Filter Bank.

Slaney, M. *Auditory Toolbox for Matlab*; Interval Research Corporation: Palo Alto, CA, USA, 1998; Volume 10.

Soltani-Farani, A. (1998). Sound visualisation as an aid for the deaf, a new approach.

Srivastava, R.K., Greff, K., & Schmidhuber, J. (2015). Training Very Deep Networks. *NIPS*.

Stimberg M, Brette R, Goodman DFM (2019). Brian 2, an intuitive and efficient neural simulator
eLife 8:e47314. doi: 10.7554/eLife.47314

Tammina, Srikanth. (2019). Transfer learning using VGG-16 with Deep Convolutional Neural Network for Classifying Images. International Journal of Scientific and Research Publications (IJSRP). 9. p9420. 10.29322/IJSRP.9.10.2019.p9420.

Takahashi, N., Gygli, M., Pfister, B., & Gool, L.V. (2016). Deep Convolutional Neural Networks and Data Augmentation for Acoustic Event Detection. arXiv: Sound.

Tan, Q., & Carney, L. H. (2003). A phenomenological model for the responses of auditory-nerve fibers. II. Nonlinear tuning with a frequency glide. *The Journal of the Acoustical Society of America*, *114*(4 Pt 1), 2007–2020. https://doi.org/10.1121/1.1608963

Tan, M., & Le, Q.V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *ArXiv, abs/1905.11946*.

Unoki, M., Irino, T., & Patterson, R.D. (2001). Improvement of an IIR asymmetric compensation gammachirp filter. *Acoustical Science and Technology, 22*, 426-430.

Valenti, M., Diment, A., Parascandolo, G., Squartini, S., & Virtanen, T. (2016). DCASE 2016 Acoustic Scene Classification Using Convolutional Neural Networks. *DCASE*.

Wagner, H., Brill, S., Kempter, R., & Carr, C.E. (2009). Auditory responses in the barn owl's nucleus laminaris to clicks: impulse response and signal analysis of neurophonic potential. *Journal of neurophysiology, 102 2*, 1227-40 .

Wang, Y., & Hu, W. (2018). Speech Emotion Recognition Based on Improved MFCC. *International Conference on Computer Science and Application Engineering*.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., & Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. arXiv preprint arXiv:1705.08292.

Zagoruyko, S., & Komodakis, N. (2016). Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer. *ArXiv, abs/1612.03928*.

Zhang, B., Leitner, J., Thornton, S. (2019). Audio recognition using mel spectrograms and convolution neural networks.

Zhang, H., Mcloughlin, I., & Song, Y. (2015). Robust sound event recognition using convolutional neural networks. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 559-563.

Zhou, Q., Shan, J., Ding, W., Wang, C., Yuan, S., Sun, F., Li, H., & Fang, B. (2021). Cough Recognition Based on Mel-Spectrogram and Convolutional Neural Network. *Frontiers in robotics and AI*, *8*, 580080. https://doi.org/10.3389/frobt.2021.580080

Appendix

Code for the implementation of the networks and data pre-processing are hosted here:

GitHub Repository: https://github.com/slyyyle/Audio-Benchmarking-Graduate-Thesis