Deep Neural Networks for Urban Sound Classification

Jhonny Velasquez
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0002
jhonnyv19@vt.edu

Abstract

Audio classification is important in many scenarios and is employed in a lot of modern-day applications today, such as virtual assistants, music/speech recognition, and text-to-speech transcription software. This paper explores different techniques for both audio processing and deep learning architectures for performance in audio classification, specifically for urban sound classification. It was found that treating audio classification as an image classification problem on each audio sample's spectrogram provided superior results to other methods attempted in terms of both loss and accuracy.

1. Introduction

This project attempted to explore different methods for audio classification, specifically for classifying ten classes of urban sounds that can be heard in urban environments. A detailed review of the labeled *UrbanSound8K* dataset that was exclusively used for training and testing can be found in the paper published by Salamon, Jacoby, and Bello [2]. The dataset consists of 8732 labeled audio excerpts which are all less than or equal to 4 seconds in duration. All models which were to be trained would take as a training example some features extracted from an audio clip along with the training label, which was a one-hot encoded array.

The main objective of this project was to achieve a level of accuracy that is within a close margin (~5% to ~10%) to the average accuracy performance of a human, which for a 4-second audio clip was found to be 82% [2].

1.1 Previous and Related Works

Currently, audio classification tasks are generally done by first extracting some features from the audio clip, such as Mel-frequency Cepstral Coefficients, which are commonly used for speech classification purposes. Typically, extracting these types of features reduces the overall amount of data that is used for training and can provide each audio sample with a consistent training shape for the training model. One common issue that arises when training with audio data is that most training examples end up not having consistent dimensions, this can be due to many reasons but usually is due to differences in sampling rate, audio excerpt duration, and differences in the number of audio channels. Training a model with a dataset with these types of inconsistencies can pose a lot of challenges during preprocessing. Another challenge that comes with training directly on the time-series data is the amount of memory that must be available. Audio that was sampled at 44.1 kHz, which is common for music files, will have 44,100 data points for every second of audio on every channel. So, for a stereo audio file with a sampling rate of 44.1 kHz which is 4 seconds in duration, one would have approximately 353,000 data points to feed into the model without any preprocessing done. This can pose difficulties when training on systems with limited memory and can also result in training models with lots of parameters that would take a long time to train even when using hardware acceleration. Thus, most audio classification tasks involve a preprocessing step, which can involve downsampling data, extracting frequency domain information, and artificial shortening and elongating of audio excerpts. All of these methods.

1.2 Background and Motivation

Accurate and automatic classification of environmental sounds can be used in many helpful applications. For instance, people with hearing disabilities could be once again given the ability to observe sounds if a diverse audio classifier that they could carry with them was integrated with a device that converted audio into another sense they could observe. Alternatively, in the field of robotics, an autonomous robot operating in an urban environment could utilize accurate audio classification to collect information about its whereabouts that may not be immediately visible using computer vision, which is a technology popular in the field of autonomous robotics that is limited to the information in front of the camera. Although this paper only addresses a very small subset of all the possible urban environment sounds that are too small for innovative applications, any contributions in the field from the research community will ultimately yield a net positive outcome.

2. Approach

2.1. Audio Classification

As discussed in the introduction, audio classification can be done in many and this is primarily due to the many ways that features can be extracted from the data. A common strategy often employed in audio classification is the use of Mel-frequency cepstral coefficients (MFCCs) which are coefficients derived from the Mel-frequency cepstrum. A description of how this strategy was used for this task with this dataset can be seen in the paper by Li et al., 2019 [8]. The Mel scale is a scale of pitches, which are related to frequency, that attempts to replicate how humans perceive pitches to be spaced in frequency [6]. With this adjusted frequency scale, a cepstrum can be derived for any given audio clip that will describe certain properties of the audio clip on a frequency scale that mimics how people would observe the sound. The MFCCs obtained from the Mel-frequency cepstrum is then used to classify different classes of audio since the MFCCs for two audio clips sharing similar qualities will be relatively similar. These coefficients work well in tasks involving music and speech and are most likely due to the coefficients being derived from a frequency scale that represents human hearing. Both speech and music are in a way designed to vary in time and frequency that another human can expect them to. However, when it comes to classifying classes of audio that are not exactly natural to the human ear, such as gunshots and air-conditioning units, one can argue that using MFCCs is not the most effective tool for classification given the wide bandwidth of sounds commonly heard in urban environments.

All audio was imported from the *UrbanSounds8K* dataset using the Librosa python library's *load* function, with default parameters, which included the downsampling of all audio data down to the default sampling rate of 22.05 kHz [9]. This was done to reduce the total amount of data per training example upon importing into memory, while also preserving most of the important bandwidth of each audio clip. This was also a way to reduce the total amount of data associated with each training example since all the information needed to appropriately classify each audio clip for the purposes of this project could be stored in just a single channel.

To explore this issue, audio classification for one of the experiments was done using MFCCs while the others were done on data from different methods of audio clip preprocessing. The MFCCs were generated using the Librosa Python library's *mfcc* function, from which 40 MFCCs were generated from each audio clip [9]. These MFCCs were then averaged across time for each respective frequency using an arithmetic mean and then input into a feedforward neural network with 1 hidden layer with 48 units. The architecture of which can be found in the diagram below, generated using the NN-SVG tool developed by A. Lenail [5]. As a regularization technique, a dropout layer with a rate of 0.5 was included for the hidden layer. Note that colors associated with the weights are not representative of the actual weights used during training purposes and are purely for illustrative purposes. All neural networks discussed in this paper were developed and trained using the TensorFlow machine learning framework [7].

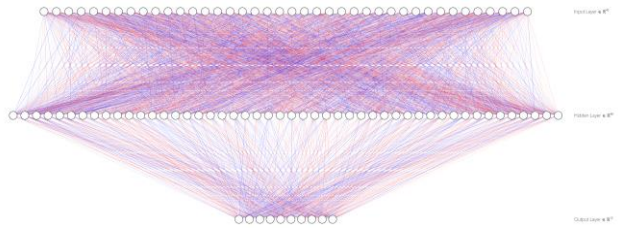

Figure 1: Feed Forward Model Architecture

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1, 48)             1968
_________________________________________________________________
dropout (Dropout)            (None, 1, 48)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1, 10)             490
=================================================================
Total params: 2,458
Trainable params: 2,458
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Feed Forward Model Parameters

The main alternative method that was used for data processing was the derivation of the spectrogram for every audio clip using the Matplotlib Python library's *specgram* function. [4]. A spectrogram is a 2-dimensional plot of a visual representation of the spectrum of frequencies of a given signal across time, using the short-time Fourier transform (STFT). With the spectrogram data for each of the audio clips, standard image classification techniques were used to classify the spectrograms into the ten classes present in the dataset. This method was thought to yield better performance than features generated by using the Mel frequency scale, for the same line of reasoning described above. Although the frequency scale shows good results for tasks relating to speech and music, this same heuristic may not be directly applicable to the classes of audio found in the dataset. Therefore, a standard spectrogram was chosen as the feature set to be extracted for every audio clip. It should be noted that most audio classification tasks solved using deep learning employ a similar procedure to this method by first extracting features

that can be represented by image data, such as spectrograms and chromagrams, and then treating the problem as an image classification task for which there are well-developed techniques for. A spectrogram for an audio clip in the dataset is included as a sample below.

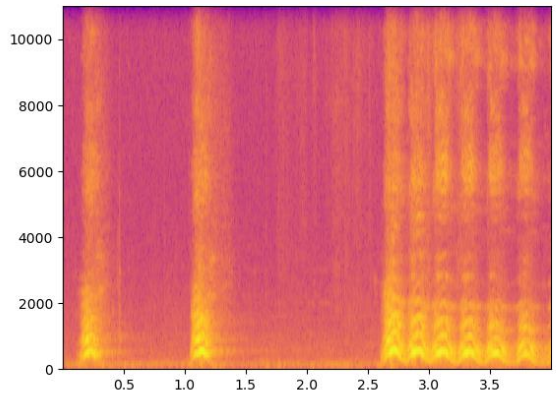

Figure 3: Spectrogram for 101415-3-0-2.wav in UrbanSounds8K dataset

Audio clips in the dataset are all a maximum of four seconds in length but there are many audio clips that are as short as half a second in duration. Since the width of the spectrogram is directly dependent on the duration, all spectrograms which were generated were zero-padded to the right up to the maximum length of the spectrogram which could be generated with the longest audio clip in the dataset as part of the data preprocessing. This was done because the image classifier that was designed for this project required images of fixed size. Simply truncating the longest audio clips in the data would most likely lead to the loss of important data that could be used during the training process. Most image classifiers are designed to perform optimally with square images, that is, having equal pixel height and width dimensions. However, as discussed above, this could not be done without some loss of data, so the rectangular spectrogram image dimensions were preserved and a Convolutional Neural Network (CNN) was designed to better handle the rectangular dimensions. This was done by adjusting filter dimensions and Maxpooling layers to account for the relative differences between height and width of the spectrogram, the specifications of the model that was used during training can be seen in the figure below.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 129, 700, 32)	4032
activation (Activation)	(None, 129, 700, 32)	0
conv2d_1 (Conv2D)	(None, 129, 700, 32)	128032
activation_1 (Activation)	(None, 129, 700, 32)	0
max_pooling2d (MaxPooling2D)	(None, 32, 29, 32)	0
dropout (Dropout)	(None, 32, 29, 32)	0
conv2d_2 (Conv2D)	(None, 32, 29, 64)	256064
activation_2 (Activation)	(None, 32, 29, 64)	0
conv2d_3 (Conv2D)	(None, 32, 29, 64)	512064
activation_3 (Activation)	(None, 32, 29, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 8, 1, 64)	0
dropout_1 (Dropout)	(None, 8, 1, 64)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 256)	131328
activation_4 (Activation)	(None, 256)	0
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
activation_5 (Activation)	(None, 128)	0
dropout_3 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290
activation_6 (Activation)	(None, 10)	0

Total params: 1,065,706
Trainable params: 1,065,706
Non-trainable params: 0

Figure 4: CNN Model Architecture and Parameters

Prior to training either model, it was anticipated that the feedforward neural network would perform worse than the CNN regarding accuracy. This was because the feedforward network dealt with only 40 coefficients as opposed to a more comprehensive summary of the audio clip's properties such as the spectrogram described above, which the CNN was designed to train on. However, because these models vary drastically in the amount of data they train on and the number of trainable parameters they each have, it was expected that the feedforward neural network would take a lot faster to train than the CNN.

During the training process, the runtime hypothesis was proven to be correct as the CNN took roughly 3 hours to train using Google Colab hardware acceleration on an NVIDIA® Tesla® P100 GPU, which made running repeated trials with modified network parameters difficult to perform. This runtime contrasts with the feedforward neural network that took less than 5 minutes to train on an Intel Core i7 10th Gen CPU. This made model performance changes with respect to changes in model parameters easier

to observe.

2.2 Feed Forward Neural Network

With regards to the feedforward network, specific model parameters that were adjusted included the number of hidden layers, the length of the hidden layers, and the dropout rate. It was originally believed that accuracy from the MFCCs would increase as the number of hidden layers in the model also increased. Similarly, it was also believed that longer hidden layers would also yield better performance since more trainable parameters would be created. However, this was not proven to be true since trial runs indicated that more than one hidden layer in the feedforward network at the same lengths specified in the diagram above made it more difficult for the model to learn the data's classes. A single hidden layer seemed to provide optimal results regarding both loss and accuracy. Additionally, it was found that a dropout rate of 0.5 was optimal for results. Any dropout rate lower or higher resulted in the model overfitting to the training data or not being able to generalize to even the training data, leading to high loss and low accuracy performance.

2.3 Convolutional Neural Network

The CNN had slightly more parameters that could be adjusted and included convolutional layers, filter quantity, filter dimensions, max-pooling layers, max-pooling dimensions, dropout rate, padding type, and feed-forward parameters at the end of the convolutional part of the entire neural network. The overall structure of the CNN for this project was inspired by the CNN used in Homework 4 for Virginia Tech's CS 4824: Machine Learning course, which was designed for image classification on the widely popular CIFAR 10 dataset [3]. Initial trials were attempted using the same neural network structure without any adjustments. The results that were to be collected from this were not expected to be optimal since the CNN was originally designed for images from the CIFAR-10 dataset which were 32x32 in pixel dimension, thus the important aspects of the CNN such as the filters and pooling layers would most likely not perform as well on the spectrograms images from the audio clips which had dimensions of 129x700. Optimizations to the original CNN included adjusting the original filter size from 3x3 to 5x25 to adjust for the image having a much longer width than height. For the MaxPooling2D layers, pool sizes were increased from 2x2 to 4x24 so. Since filtering and pooling are what extract a lot of the features to be used for classification in a CNN, it was important that the dimensions of both respective elements be proportional to the rectangular dimensions of the image they would work on. As the final modification to the CNN, the feedforward network at the end of the original CNN was changed to have two layers, with 256 and 128 neurons respectively, before the final output neurons instead of the original single layer with 512 neurons before the output layer. The main motivation for the change in the feed-forward network was memory limitations in the runtime environment since a single layer with 512 layers resulted in too many trainable parameters.

3. Experiments and Results

Accuracy was the main method of evaluation of a model's success for this project. Specifically, 10-fold cross-validation using the predefined folds that come with the *UrbanSounds8k* dataset, as described in the associated website for the dataset. The dataset comes from 10 folds (as 10 distinct folders), each of which contains a non-equal amount of audio clips or distribution of classes among the audio clips. To obtain final model accuracy, there were ten training iterations done, each time with different training, validation, and test splits. For each iteration, one of the folds was selected as the testing set for that iteration and the rest were all combined to form the training dataset. After training, the model is evaluated on the test set that was chosen. After each of the training iterations and evaluations, the weights of the model were reset so that they did not influence the performance of future iterations. This is the preferred method of testing that the authors of the dataset website suggested using in order to collect consistent results and it was followed for this project.

As part of the training process, for each training iteration, a validation dataset was formed by splitting the training dataset using an 80-20 split. This validation set is separate from the testing set that is chosen at the beginning of each training iteration. Although the model never uses the training examples present in the validation dataset, it provides the training process with a validation accuracy metric that can be used for stopping the training process before the predetermined number of epochs is completed, thus preventing overfitting of the training data. This parameter for training the model is often referred to as early stopping and was set up to trigger once the validation accuracy was observed to consistently decrease for 3 consecutive epochs. All models were trained using the Adam optimizer [1], the categorical cross-entropy loss function, and a batch size of 64 images. All of which is a part of the TensorFlow library and can be passed as parameters during the model compilation process.

3.1 Feed Forward Network Results

After training the feedforward using the hyperparameters and evaluating using the cross-validation technique described above, the following results were obtained.

Test Fold	*Accuracy*	*Loss*
1	0.7723	0.7334

2	0.7030	0.9585
3	0.7260	0.8532
4	0.7734	0.7142
5	0.7635	0.7247
6	0.7023	0.8983
7	0.7631	0.7478
8	0.7528	0.8995
9	0.7614	0.7196
10	0.7840	0.6869
Arithmetic Mean	0.7502	0.7936

Figure 5: Feedforward Network Results

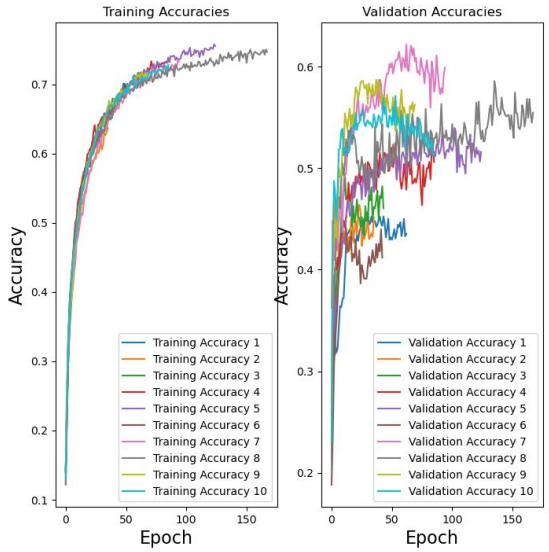

Figure 6: Feedforward Network Accuracy Plots

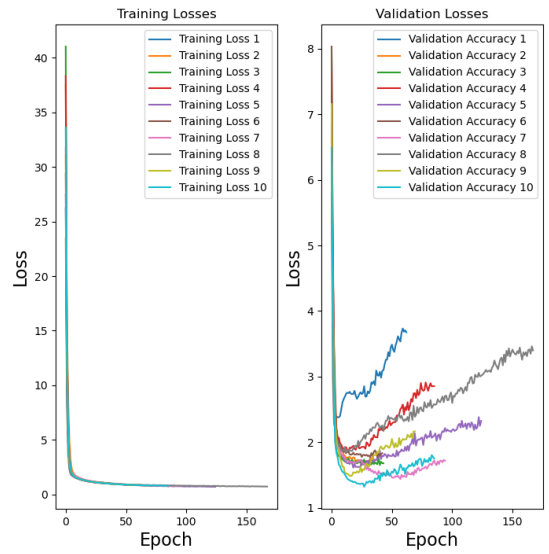

Figure 7: Feedforward Network Loss Plots

3.2 Convolutional Neural Network Results

After training the feedforward using the hyperparameters and evaluating using the cross-validation technique described above, the following results were obtained.

Test Fold	Accuracy	Loss
1	0.8766	0.4473
2	0.8585	0.5631
3	0.8681	0.6928
4	0.8806	0.4662
5	0.8750	0.4563
6	0.8635	0.4911
7	0.8885	0.4300
8	0.8953	0.3956
9	0.8725	0.4800
10	0.8683	0.5918
Arithmetic Mean	0.8747	0.5014

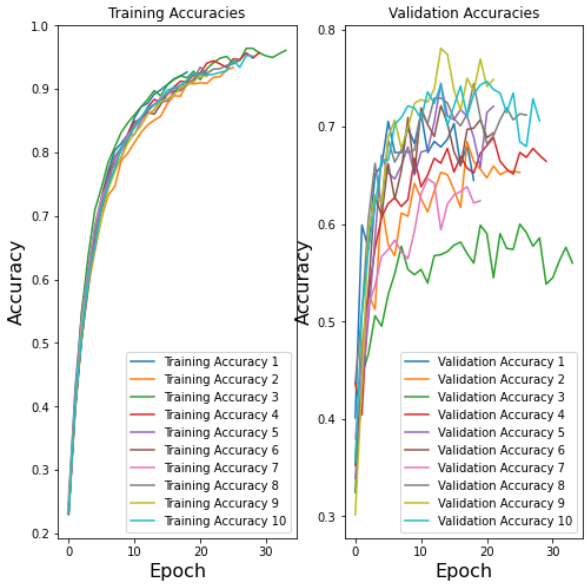

Figure 8: CNN Accuracy Plots

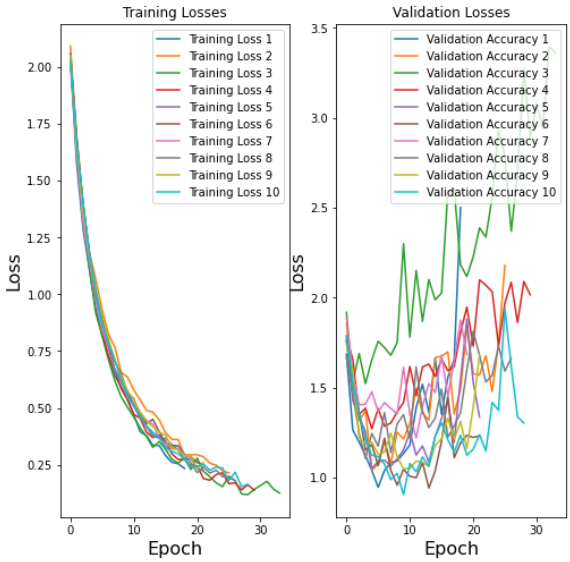

Figure 9: CNN Loss Plots

3.3 Results Interpretation

The results above indicate that the original goal of accomplishing near-human performance on urban sound audio clips of about 4 seconds, which was 82%, was met. The CNN performed, overall, a lot better than the feedforward with respect to both loss and accuracy. The accuracy and loss plots look roughly similar for both the feedforward and CNN. An interesting observation to note is that validation loss would decrease but slowly increase along with validation accuracy.

4. Conclusion

Overall, training an audio classifier with the spectrogram image of an audio clip appears to be better for audio classification than using MFCCs, with the CNN being on average 12% more accurate at classifying audio clips than the feedforward network. Although the exact reason why the CNN performs better is unknown, it is believed to be most likely due to the difference in the amount of information that is present in 40 coefficients versus the entire spectrogram of an audio clip. Though the difference in training time should also be considered when evaluating the overall performance of both models. The CNN took a few hours to fully train and evaluate 10 models, while the feedforward took less than 5 minutes to do the same task.

Future works might seek to explore how Recurrent Neural Networks (RNN) based classifiers perform compared to the results discussed above, as well as how different preprocessing and feature extraction methods change results. Additionally, if memory limitations are not an issue, it would be interesting to see how a classifier performs directly on the time series audio clips with minimal preprocessing done.

A GitHub repository with the python notebook file including the code used to generate the data and results is included below.

https://github.com/jhonnyv19/CS4824_Project_Audio_Classification.git

5. References

[1] Kingma, D.P. and J. Lei Ba, *Adam: A Method For Stochastic Optimization*, in *The International Conference on Learning Representations*. 2015. p. 15.
[2] Salamon, J., C. Jacoby, and J.P. Bello, *A Dataset and Taxonomy for Urban Sound Research*, in *Proceedings of the 22nd ACM International Conference on Multimedia*. 2014, Association for Computing Machinery. p. 1041–1044 , numpages = 4.
[3] Krizhevsky, A., *Learning Multiple Layers of Features from Tiny Images*. 2009. p. 58.
[4] Hunter, J.D., *Matplotlib: A 2D Graphics Environment.* Computing in Science & Engineering, 2007. **9**(3): p. 90-95.
[5] Lenail, A., *NN-SVG: Publication-Ready Neural Network Architecture Schematics.* Journal of Open Source Software, 2019. **4**(33): p. 747.
[6] Stevens, S.S., J. Volkmann, and E.B. Newman, *A Scale for the Measurement of the Psychological Magnitude Pitch.* The Journal of the Acoustical Society of America, 1937. **8**(3): p. 185-190.
[7] Mart\'n~Abadi, et al., *TensorFlow : Large-Scale Machine Learning on Heterogeneous Systems*. 2015.
[8] Li, J., et al., *What's That Sounds? Machine Learning for Urban Sound Classification*. 2019. p. 9
[9] Brian McFee, “librosa/librosa: 0.9.1”. Zenodo, Feb. 15, 2022. doi: 10.5281/zenodo.6097378.