Sound Classification using K-nearest Neighbors and Deep Convolutional Neural Networks

Julie Jiang and Ari Brown

December 14, 2017

Abstract

A K-nearest neighbor (kNN) algorithm is an empirically efficient feature clustering method, and the nature of deep convolutional neural networks (CNN) classifier makes them well suited to learn discriminative spectro-temporal patterns in audio [1]. For this study, we compare the two starkly different classification methods, CNN and kNN, on the problem of sound classification. We explore their difference in performance on an annotated musical instrument dataset and examine the results to report further directions for research. In these experiments, Mel-Frequency Cepstral Coefficients (MFCCs) were extracted from the audio samples as features We achieved a 79% accuracy with the kNN model on the validation set and a 93% accuracy with the CNN model.

1 Introduction

The problem of sound classification has been studied in depth and has multiple applications related to identity discrimination, enhanced hearing aids, robotics, and music technology. There are two approaches to the problem of sound classification can be approached. The first method is empirical and requires a database of sounds to learn from through feature extraction. The second method is top-down, by using predefined feature rules to make classification decisions. In this project, we take the developmental approach and use learning mechanisms to enable a robot to gain meaningful information about its environment through sound.

As with any supervised machine learning method, the problem of feature engineering is fundamental to the success of machine learning model. We leverage our domain-specific knowledge on sound to craft feature vectors based on sound. Motivated by the way humans perceive and discern between sounds, we use MFCCs as our features in both models in this study. As a cepstrum based feature, MFCCs effectively encapsulates a rich variety of information in the envelope of short time power spectrum. It is heavily used in many speech recognition and sound classification tasks [2, 3, 4, 5].

The first model we experimented on is kNN. The kNN algorithm is a niave classifier which uses the nearest neighbors in the parameter space to classify new data points, measured by euclidian distance. The kNN is somewhat limited in that it cannot conventionally "learn" on its training data, but it is an "online" algorithm, in that new labeled data can always be introduced while in runtime. The euclidian distance measurements are costly when the dataset is large and has a high number of dimensions. However, the kNN algorithm can be optimized by augmenting the data and cutting down on the time complexity of search.

The second model we delved into is CNN. CNNs are commonly used in classification and recognition problems but are primarily applied to computer vision problems. Recent advancements in sound classification show that CNNs perform superiorly to traditional methods of classification [1, 3]. The convolutional architecture takes advantage of the data locality present in temporal based data that found naturally in sound.

The rest of the paper is structured as follows. In section 2 we briefly survey related literature in sound classification. We discuss the dataset and libraries used in section 3. We present the method of feature extraction and engineering in section 4. In section 5 and 6 we explore the architecture, experimental setup and results of the kNN model and CNN model, respectively. We conclude with a discussion of our study in section 7. A reference implementation of the models is available publicly on Github: `https://github.com/julie-jiang/music-classification-using-knn-cnn`.

2 Related Work

To date, a variety of signal processing and machine learning techniques have been applied to the problem of sound classification. Including non-negative matrix factorization [6], unsupervised dictionary learning using spherical k-means [7], and Gabor wavelet filterbanks with Hidden Markov models [8]. More recently researchers are tapping into the effectiveness of deep neural networks [9] and deep convolutional neural networks [1].

A number of them research the application of sound classifiers in urban and environmental settings [1, 2, 3, 7]. Other works evaluate classifiers that detect events in real life [6, 8]. Such problems are exceptionally challenging due to the scarcity of large-scale and comprehensive annotated dataset. The quality of the datasets is also easily compromised by the mask of noises.

3 Technical Details

3.1 Dataset

To evaluate the performance of our models, we use the NSynth dataset made available by Google Inc. [10]. The NSynth dataset is a high-quality, large-scale dataset designed for data-driven machine learning in sound. It contains monaural 16kHz 4-second recordings of 10 different instruments in every possible pitch, timbre, velocities, and source of production (acoustic, electric and synthetic). The musical notes are held for the first three seconds and are allowed to decay in the final second.

As feature extraction takes unreasonably long, we ran our preliminary experiments on only a subset of the NSynth dataset. We chose recordings of 5 instruments from the training set, and the same 5 instruments from the validation set. There are 55,263 training samples and 3,586 validation samples, resulting in a training and testing split of about 9:1.

3.2 Tools and Libraries

The kNN models are written with Scikit-Learn [11], and the neural networks models are written in Keras with Tensorflow at its core[12, 13]. Features are extracted using Librosa, an audio processing python library [14].

4 Feature Engineering

The raw audio signal is not well suited for feature learning due to its volatility and high dimensionality. A common approach to sound-related learning is to transform the data into a time-frequency representation, a popular choice being the Mel-Frequency Cepstral Coefficients [4, 5, 8, 9].

Mel-Frequency Cepstral Coefficients can be obtained through the following algorithm:

1. Get the spectrum from the signal using the Fourier Transform, in a defined window of samples.

2. Map the Mel scale to the frequency space and calculate weighted averages of the spectrum's magnitude in the local areas around different points on the mel scale. Another way to visualize this is by creating many bandpass filters around the mel points. Use the same amount of filters as the number of coefficients desired. The Mel scale is a perceptual scale based on human ratings of how equidistant pitches are from one another.

3. Take the log of these average energies.

4. Interpret the log-energies as another signal and take the Discrete Cosine Transform (DCT) of these log-energies. The DCT will result in the MFCC coefficients, and these coefficients encode the weighting of cosine waves needed to resynthesize the spectral shape of the signal.

MFCC coefficients are computed for every window size of 2048 and a hop length of 512. We compared and contrasted the performance of the models on feature sets computed from 14 coefficients and 40 coefficients.

5 K-Nearest Neighbors

The K-Nearest Neighbors (kNN) algorithm uses euclidian distance to classify new data points based on a pre-existing dataset. The kNN algorithm plots all training data points within their high dimensional feature space. For a new point to be classified, the kNN algorithm plots the unlabeled data point in the same feature space, and finds the closest K neighbors to the unlabeled point, using Euclidean distance. The local neighbors of the unlabeled point are returned, and then a voting system can be used to determine the label. The kNN algorithm can also work where K=1, and classify the data point to its nearest labeled data point. We used the MFCC's as the dimensions of our training data points.

5.1 Results

We ran the kNN algorithm and used k=1, k=3, k=5, and k=10 as the number of local neighbors in the voting system. We found that k=1 resulted in the best success rate for our dataset. We suspect that this is due to ambiguity in region borders, and more neighbors create more uncertainty during classification.

K	Success
1	79.3
3	77.4
5	76.8
10	73.9

6 Deep and Convolutional Neural Networks

Deep neural networks (DNNs) and convolutional neural networks (CNNs) are both artificial neural networks (ANNs). ANNs are characterized by layers stacked together, where each layer consists of a set of neurons. An ANN has an input layer, an output layer, and one or more hidden layers sandwiched in between. In our experiments, we first explored the effectiveness of using a DNNS. We moved on to CNNs when our DNN could no longer perform any better.

6.1 DNN

Due to the simplicity of a DNN in comparison to a CNN, we begin our experiments with DNNs. DNN mainly consists of many, as such, it is named *deep*, densely connected layers. Every input neuron in a dense layer is involved in the computation of every output neuron. Dense layers are also feedforward, which means the neurons in layer l determines the neurons in every layer after the l^{th} layer.

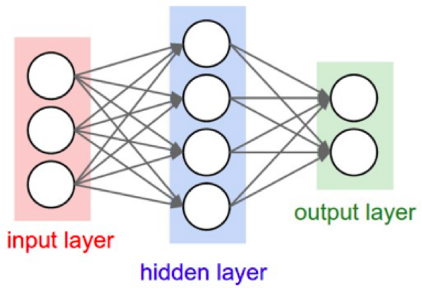

Figure 1: An example of a DNN with a single hidden layer. Image source [15].

6.1.1 DNN Architecture

In order to transform the data into a format compatible with DNNs, We took the average of each MFCC coefficient across time, so every audio sample is transformed into a 14×1 or 40×1 sized feature vector. Empirically, we found that the dataset with 40 coefficients performed marginally better.

Our DNN model is comprised of 6 densely connected hidden layers. Let us denote the set of input neurons as $\mathbf{x}$. Then the neurons in the l^{th} hidden layer for some $l > 1$ is recursively defined as follows.

$$h^l(\mathbf{x}) = f_l(\mathbf{W}^l h^{l-1}(\mathbf{x}) + \mathbf{b}^l) \tag{1}$$

where $\mathbf{W}^l$ is the weight matrix and $\mathbf{b}^l$ is the bias vector, both of which are initialized as random Gaussian distributions with mean 0 and standard deviation $\frac{1}{\sqrt{40}}$.

The function $f(\cdot)$ is an nonlinear activation function. We use a sigmoid activation function $sigmoid(x) = \frac{e^x}{e^x+1}$ after the first layer, a rectified linear units (ReLU) activation function $relu(x) = \max\{0, x\}$ after every hidden layer, and a softmax activation function $softmax(x_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$ where $i = 1, 2, ..., K$ for all K classes in the final output layer. The use of the sigmoid as the activation of the first layer is motivated by normalizing the data to be between 0 and 1. ReLU is used after every hidden layer due to its computation efficiency. Research has shown that ReLU results in much faster training for large networks without much sacrifice in accuracy [16]. Finally, softmax is conventionally used as the activation function of the last layer for classification problems. It squashes each unit in the output vector to a real number between 0 and 1 so that the sum over the vector is equal to 1. As such, the softmax function produces a vector equivalent to a categorical probability distribution.

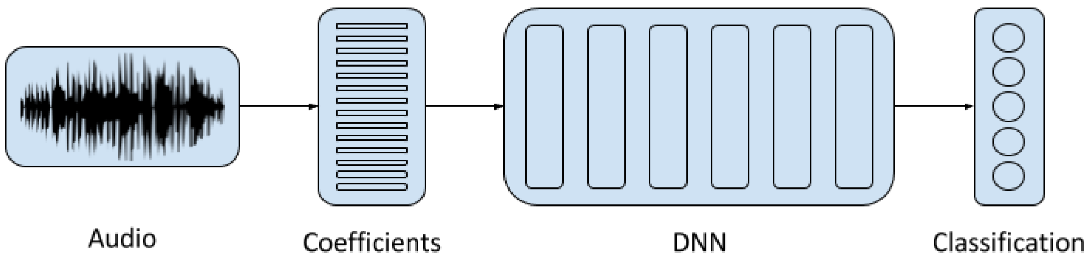

Figure 2: DNN Architecture

The first hidden layer has $20 \times 40 = 800$ neurons, the second hidden layer has $10 \times 40 = 400$ neurons, the third and fourth layers each have 40 neurons, the fifth layer has $40/2 = 20$ neurons, and the final layer has $40/4 = 10$ neurons. The choice of the number of layers and the number of neurons in each layer seems somewhat arbitrary but is in fact empirically determined. The first few layers augments the data so the model can uncover minute details from the input. The last few layers shrinks the data so it can narrow down which classification fits best

Additionally, each dense layer is followed by a dropout of 0.2. Dropout layers are a simple and elegant way to reduce overfitting [17]. It allows the model to randomly forget 20% of the data from the outputs of every hidden layer in every iteration.

The model aims to minimize categorical cross entropy via batch-sized stochastic gradient descent. We use the ADAM optimizer, which is a slight variation of the traditional stochastic gradient descent, as ADAM adaptively decreases the learning rate as training progresses to prevent overshooting [18].

6.1.2 DNN Results

Neural networks optimize its weights via backpropagation after every batch and finish a training epoch after it has seen every training data. The batch sizes we used are 1, 16, 32 and 50. We found that smaller batch sizes enable the model to learn more efficiently and converge faster, but at the cost of slower epochs. A batch size of 16 or 32 is an optimal size considering the tradeoffs between fast convergence and slow training.

While the training accuracy converges asymptotically to 100% and symmetrically the training loss converges asymptotically to 0, it is meaningless to keep it running when the model is overfitting. For this reason, we test the model against the validation set at the end of every training epoch and halts training when the model's performance on the validation set stops improving.

The best DNN model we produced is the one described in section 6.1.1. While the training accuracy was approaching 90%, the validation accuracy peaked after the 35th training epoch at 85%, an 8% improvement on the kNN model in section 5.

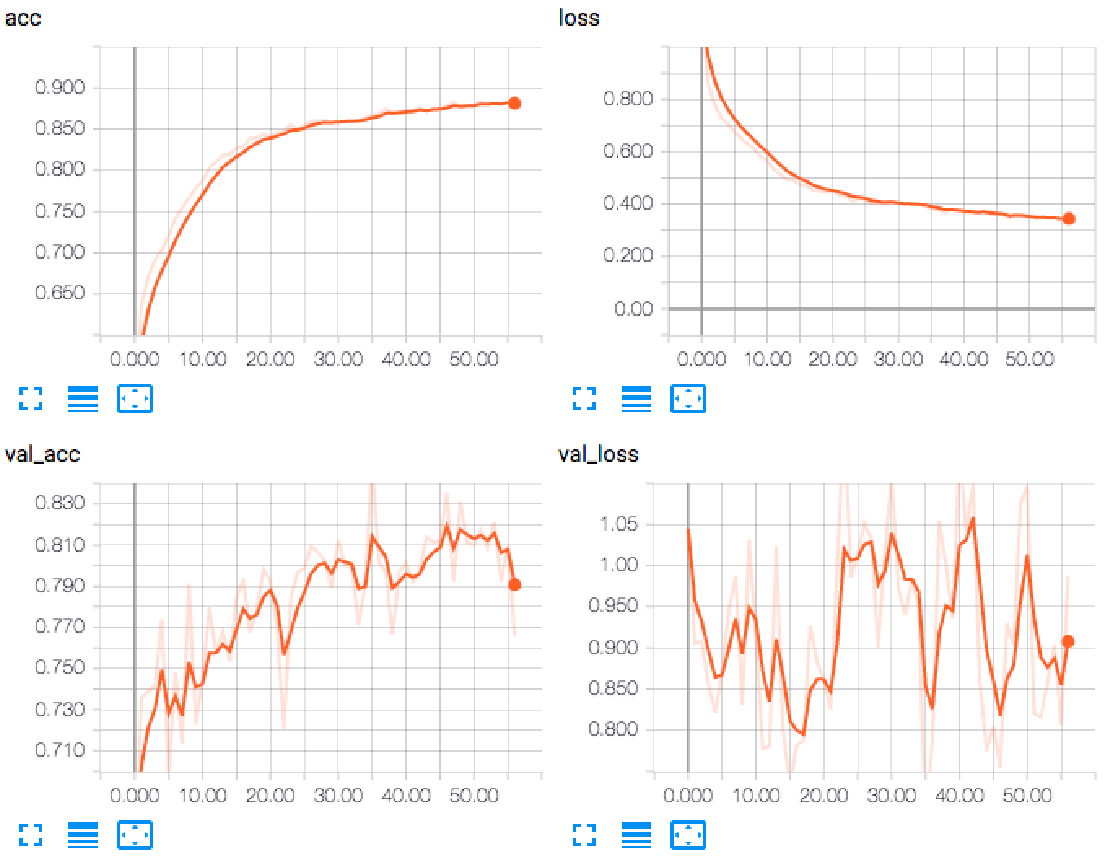

Figure 3: Training accuracy (top left), training loss (top right), validation accuracy (bottom left), and validation loss (bottom right) plots of the CNN model described in section 6.1.1 with a smoothing of 0.6. The x-axis indicates training epoch, and the y-axis indicates either accuracy or loss. Plots generated by TensorBoard.

As indicated in figure 3, while the training accuracy and loss steadily increased or decreased, the validation accuracy and loss tends to be quite volatile, suggesting that the model was having a hard time learning the features that mattered. Later in section 6.2.2 we will see that the CNN model performed much better.

6.2 CNN

Largely motivated by [3], we explored the use of CNNs in hopes that a more sophisticated model can capture the more fine-grained details both the kNNs and DNNs missed. A CNN is most commonly used to analyze an image and is characterized by its convolutional layers that try to learn abstract features of localized regions of the image.

6.2.1 CNN Architecture

One challenge in using a CNN for sound related deep learning is the variable length of every sample of audio clips. CNN requires that inputs are of the same shape. [3] introduced a technique to overcome this problem by sliding a fixed-size window over every sample of an audio clip, essentially breaking one audio sample into more than one.

Fortunately for our purposes, all of our input samples are 4-seconds long in duration, so it is easy to convert every sample into temporal based MFCC matrix of fixed size. Our matrices are 14×173 in size, with 14 being the number of coefficients and 173 corresponding to time. Given this as the input, We apply a 2D convolutional layer with a 3×3 kernel filter. The activation of this convolutional layer is by convention ReLU. Both the kernel weight matrix and the bias vector are initialized with a random Gaussian distribution with mean 0 and standard deviation 0.1. Every neuron is deepened from a depth of 1 to a depth of $14 \times 4 = 56$. Therefore, this resulting feature map is slightly smaller in both length and width but a lot deeper. We then follow up with a 2D max-pooling layer with 2×2 kernel filter, thereby down-sampling the data. This resulting data is then flattened for the dense layer, but not before a dropout layer of 0.5. The final dense layer finishes with a softmax activation function.

The considerable increase in depth in the first convolutional layer is motivated by the empirical results from section 6.1.1, which suggests that the model performs better when the first few layers have a large number of neurons.

Like the DNN, the model is trained with an ADAM optimizer that minimizes categorical cross entropy via batches of 32.

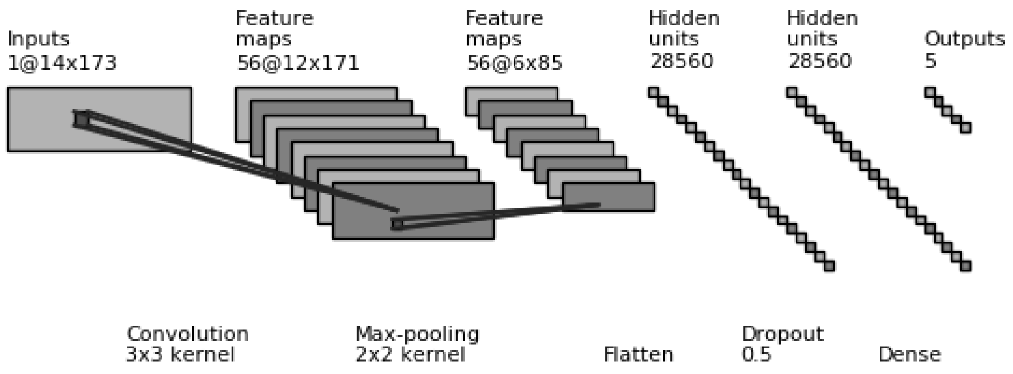

Figure 4: CNN Architecture.[1]

6.2.2 CNN Results

We evaluate the performance of the CNN model similar to that of the DNN model. In only 11 training epochs, the validation accuracy skyrocketed to a 93%. Furthermore,

[1]Figure generated using code adapted from `https://github.com/gwding/draw_convnet`.

we can see from figure 5 that the curves of the validation accuracy and loss go almost hand in hand with the curves of the training accuracy and loss, suggesting that the CNN model was much better at picking up the right things to learn from the features.

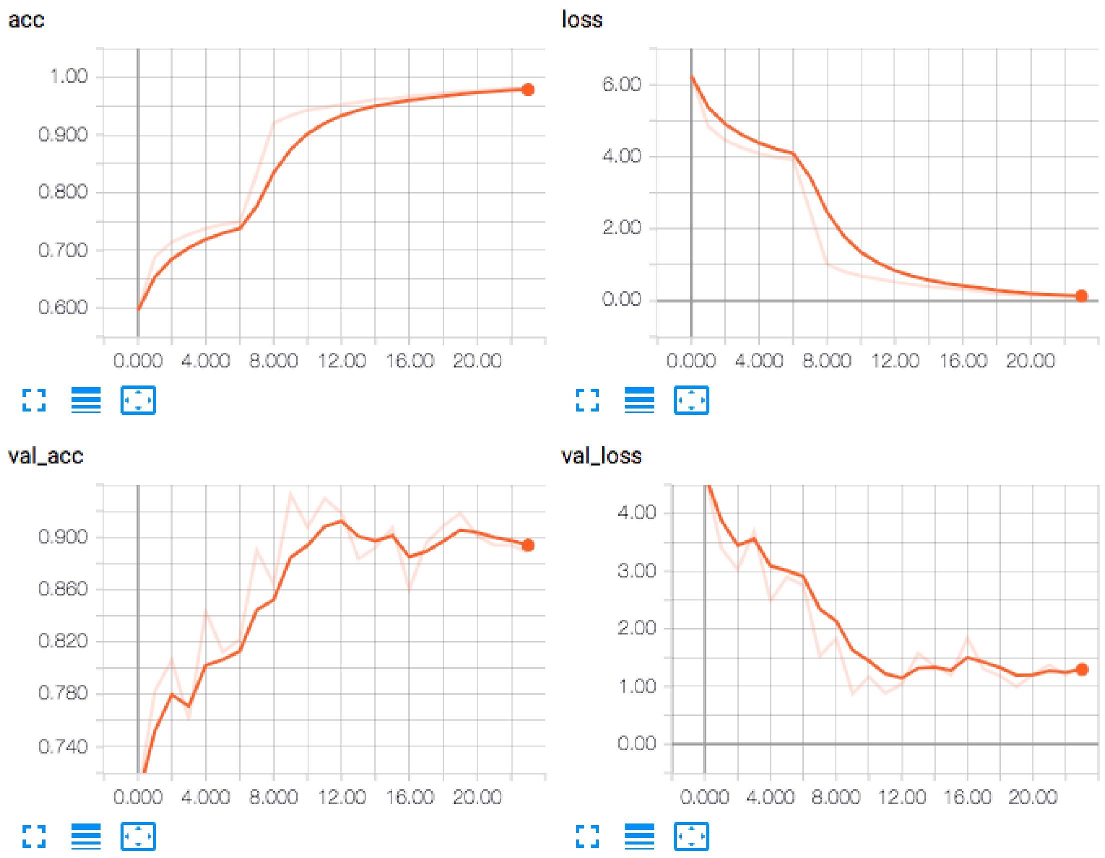

Figure 5: Training accuracy (top left), training loss (top right), validation accuracy (bottom left), and validation loss (bottom right) plots of the CNN model described in section 6.2.1 with a smoothing of 0.6. The x-axis indicates training epoch, and the y-axis indicates either accuracy or loss. Plots generated by TensorBoard.

The confusion matrix and confusion metrics such as precision and recall are particularly useful in classification problems and information retrieval. The confusion matrix and the precision, recall and F-score metrics shown in Table 1 reveals some interesting interpretations of our model. For example, there are a lot of miss classifications regarding keyboards. Piano as an instrument has more strings that match the overtones of the current string that is vibrating, so there is more chance that the overtones of that string will also be more resonant. Because of this, the spectrum (Fourier transform) of the piano will lead to more ambiguous MFCCs as they are based on spectral shapes.

The recall metric, which measures the sensitivity of correctly classifying a label, of vocal instruments is quite low, but that could be explained by the small size of training data. The number of training samples for strings, keyboards, guitars, and brass are 19390, 8068, 11380, and 12605, respectively, but there are only 3820 training samples for vocals. The limited number of data is in part due to difficulty in collecting vocal data in comparison to other instruments, which could be easily synthesized using MIDI.

		Prediction				
		String	Keyboard	Vocal	Guitar	Brass
Actual	String	791	1	0	10	12
	Keyboard	20	275	0	16	10
	Vocal	12	0	71	0	0
	Guitar	24	26	1	1425	6
	Brass	1	7	0	96	782

	String	Keyboard	Vocal	Guitar	Brass
Precision	0.933	0.890	0.986	0.921	0.965
Recall	0.971	0.857	0.855	0.962	0.883
F-Score	0.952	0.873	0.916	0.941	0.922

Table 1: Confusion matrix (top) and precision, recall and f-score matrix (bottom).

7 Conclusion

Our first approach, the kNN algorithm, resulted in a decent success rate of 79 percent. The CNN exceeded the kNN by successfully classifying the instruments 93 percent of the time. One notable difference between the algorithms is their dependency on time. Whereas the kNN classification operates based on average MFCC values across time, the CNN was able to capture the nature of the signal as it changes over time, because convolution captures information in the time domain more accurately. The kNN algorithm is limited in that distance is the only metric that compares data points, but it could be improved by increasing the number of features. The CNN classifier would also benefit from more dimensionality in the feature set, and a limitation is that the CNN is hard to use on audio samples of various length unless durations are all truncated. Also, due to the time constraint, we could only experiment with a very small subset of sound, and further directions to improving our classifiers would be to train on a more robust dataset.

References

[1] Salamon, Justin, and Juan Pablo Bello. "Deep convolutional neural networks and data augmentation for environmental sound classification." *IEEE Signal Processing Letters* 24, no. 3 (2017): 279-283.

[2] Salamon, Justin, and Juan Pablo Bello. "Feature learning with deep scattering for urban sound analysis." In *Signal Processing Conference (EUSIPCO),* 2015 23rd European, pp. 724-728. IEEE, 2015.

[3] Piczak, Karol J. "Environmental sound classification with convolutional neural networks." In *Machine Learning for Signal Processing (MLSP)*, 2015 IEEE 25th International Workshop on, pp. 1-6. IEEE, 2015.

[4] Muda, Lindasalwa, Mumtaj Begam, and Irraivan Elamvazuthi. "Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques." *arXiv preprint arXiv:1003.4083* (2010).

[5] Chen, Chin-Hsing, Wen-Tzeng Huang, Tan-Hsu Tan, Cheng-Chun Chang, and Yuan-Jen Chang. "Using k-nearest neighbor classification to diagnose abnormal lung sounds." *Sensors* 15, no. 6 (2015): 13132-13158.

[6] Mesaros, Annamaria, Toni Heittola, Onur Dikmen, and Tuomas Virtanen. "Sound event detection in real life recordings using coupled matrix factorization of spectral representations and class activity annotations." In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on,* pp. 151-155. IEEE, 2015.

[7] Salamon, Justin, and Juan Pablo Bello. "Unsupervised feature learning for urban sound classification." In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on,* pp. 171-175. IEEE, 2015.

[8] Geiger, Jürgen T., and Karim Helwani. "Improving event detection for audio surveillance using gabor filterbank features." In *Signal Processing Conference (EUSIPCO),* 2015 23rd European, pp. 714-718. IEEE, 2015.

[9] Cakir, Emre, Toni Heittola, Heikki Huttunen, and Tuomas Virtanen. "Polyphonic sound event detection using multi label deep neural networks." In *Neural Networks (IJCNN), 2015 International Joint Conference on,* pp. 1-7. IEEE, 2015.

[10] Engel, Jesse, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders." *arXiv preprint arXiv:1704.01279* (2017).

[11] Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." *Journal of Machine Learning Research* 12, no. Oct (2011): 2825-2830.

[12] Chollet, François and others. Keras, Github, 2015. https://github.com/fchollet/keras.

[13] Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016). Software available from tensorflow.org.

[14] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In *Proceedings of the 14th python in science conference*, pp. 18-25. 2015. DOI: https://zenodo.org/record/293021.

[15] Karpathy, Andrej. "CS231n Convolution Neural Networks for Visual Recognition", December 11 2017. http://cs231n.github.io/neural-networks-1/#add.

[16] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

[17] Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting."*Journal of machine learning research* 15, no. 1 (2014): 1929-1958.

[18] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv*:1412.6980 (2014).

[19] Cover, Thomas, and Peter Hart. "Nearest neighbor pattern classification." *IEEE transactions on information theory* 13, no. 1 (1967): 21-27.