Degree Project Pre-study

Helmer Nylén
helmern@kth.se

2020–05–29

In this report a brief overview of the methods used in noise classification and, to some extent, audio quality assessment is presented. This is complemented by the implementation of five classifier types and tests on additive noise in a speech signal. The goal of the literature study was to find appropriate features and classifier types for the experimental part, and thus focused on papers related to noise classification, environment or scene recognition, and classification of general audio data. The goal of the experimental part is to find the best classifier type for noisy speech, which will later be used for other noise types than independent, additive noise.

1 Background – Noise classification

The noise in a speech signal is in most cases unwanted and much work has been done to find ways to filter out and remove it. There are, however, examples of when noise itself can provide meaningful information. Qi et al. [1] use the additive background noise in a recording to identify the model of the recording mobile device, and Kotropoulos and Samaras [2] use the device's frequency response to the same end. The proposed application of these techniques is in forensics. A commercial application is to use noise to classify what kind of environment a user is currently in, as done by Leman and Faure [3] and Ma, Milner, and Smith [4]. This bears some resemblance to classification of audio data as done by Li et al. [5], where TV or radio sound is segmented and assigned one of seven classes. Another application is in noise monitoring systems, often used at construction sites or mining facilities, where noise events can be classified to determine a likely origin [6, 7]. This is similar to audio event detection which is done by e.g. Cakir et al. [8].

1.1 Feature Extraction

Different researchers use different methods for everything from preprocessing to feature extraction and classification, and we will first look at methods to extract relevant features from the audio data. Common approaches, especially in older work, are often based on describing the Fourier spectra of signals using cepstral coefficients or similar. These include Mel-frequency cepstral coefficients (MFCC), which have a history of successful applications in speech recognition [4]. A number of the reports in this study primarily use MFCCs or test them against other features, occasionally appending MFCC-Δ[1] or -$\Delta\Delta$ as well [2, 4–6, 8–11]. Couvreur et al. [7] use the cepstral coefficients acquired from Linear Predictive Coding (LPC) analysis for noise event classification, citing their use in speech analysis, but note that longer analysis windows are needed for this purpose compared to speech recognition. Similarly, El-Maleh, Samouelian, and Kabal [12] use Line Spectral Frequencies (LSF) for background noise classification.

Studies in which the performance of different features are measured place MFCCs among the best for scene recognition, general audio classification and audio quality assessment [5, 11, 13]. However, with the rise of neural networks in the last decade some studies have taken a different approach. For example, Qi et al. [1] use existing software to first get a denoised speech signal, then subtract this from the original to get only a noise signal. A Fast Fourier Transform (FFT) is then applied and the resulting coefficient histogram is directly input into the classifier. Lee

[1] The difference over time, which can be used to more easily detect changes in the spectral envelope.

et al. [14] input the time-domain waveform directly into a CNN, and Hershey et al. [15] perform a Short-time Fourier Transform, group the resulting spectrogram into mel-spaced bins, and input the log-magnitude into their CNN (thus not using the cepstral coefficients).

Leman and Faure [3] use a decision tree to select the most relevant features (sound pressure level variation and spectral flux) for classifying noise as *Intelligible*, *Environmental*, *Hissing* or *Crackling*, though they look only at low-level features suited for real-time computation in a mobile phone. Abreu, Duarte, and Villarreal [16] use the dual-tree complex wavelet transform. Ultimately, MFCC features were selected for use in the experimental part of this study as they have been successful in numerous related tasks and are easy to implement (and libraries for computing them already exist).

It is worth mentioning that MFCCs are not optimal in all cases. Ma, Milner, and Smith [4] note that the sounds in a database they used were not clearly distinguishable in the MFCC domain, and that classes used were "based on criteria other than acoustic similarity". Furthermore, in one of the studies MFCCs were found to be less suited for audio quality ranking than low-level and psychoacoustic features [10]. This was speculated to be because MFCCs are designed to mimic human hearing and thus carry more information about the content than the noise. Lastly, Avila et al. [11] do not only use MFCC features but add a pitch estimate, log-power energy, voice activity detector (VAD) output, and deltas of these to achieve their results.

Some authors perform preprocessing on the signal to filter out irrelevant parts or to segment the signal into smaller, coherent sections. This step is not necessarily based on the same features as used in the subsequent e.g. classification. For instance, Leman and Faure [3] use a VAD to retrieve only the non-voiced timeframes of a signal, as do Abreu, Duarte, and Villarreal [16]. Li et al. [5] use an intricate segmentation algorithm which is distinct from the classification of the segments.

1.2 Classifiers

Hidden Markov models (HMM) are commonly used to model the time-dependence of signals and can also be used in classification tasks, which is done by reports in this study [4, 7]. Ma, Milner, and Smith [4] found that a left-to-right HMM performed better than an ergodic HMM. Commonly the output probability distributions in the HMMs are either discrete (using vector quantization as a preprocessing step) [7] or Gaussian Mixture Models (GMM) [4], of which a single Gaussian can be considered a special case. Some studies ignore the time dependence and gather statistics about their features, fit a GMM for each label to these statistics, and classify based on which GMM has the highest probability of yielding the observation [5, 6, 12, 13] (used as baseline in [9]). Kotropoulos and Samaras [2] train GMMs on MFCC features extracted from a signal, then use the parameters of the GMM as input into a different classifier. A similar approach is part of one of several tested features by Avila et al. [11].

Other traditional classification methods include Support Vector Machines (SVM) [2], Linear Least Squares [12], Decision Tree [3, 12], and Nearest Neighbor [2, 12, 13]. Furthermore, there are a number of methods based on neural networks, including Multi-layer Perceptrons (MLP, often referred to as fully connected deep neural networks, or DNN) [2, 6, 9, 11, 16], Convolutional Neural Networks (CNN) [1, 8, 9, 11, 14, 15], Recurrent Neural Networks (RNN) [8], as well as combinations thereof. Liu et al. [17] propose an HMM–neural network hybrid called GenHMM which is described in more detail in section 2.3.2.

In the experimental part of the study five classifiers were selected for use: a HMM using GMMs, GenHMM, a type of RNN called Long Short-Term Memory (LSTM), SVM, and CNN. The specifics of these are described in section 2.3. These were selected both to give a mixture of state-of-the-art machine learning methods and traditional signal processing methods and to be relatively easy to implement.

1.3 Audio Quality Assessment

Some articles on audio quality assessment were considered in the literature study, but no experiments were conducted on the topic. It can be considered relevant for noise classification as

feature extraction and overall model architectures are often similar. One difference is that noise classification tasks in general use "clean" data, where the speech signal (if at all present) has been removed during preprocessing either by VAD or through some other means, while audio quality assessment by its nature requires a complete, degraded signal. This is sometimes accompanied by a clean reference signal (in an intrusive algorithm) or a pseudoreference signal (in certain non-intrusive algorithms) [18]. The literature study focused on non-intrusive (or non-reference) quality assessment algorithms.

While a speech recognition algorithm is ideally robust to varying noise (in the acoustic sense), an audio quality assessment algorithm is ideally robust to the content of the signal itself, be that music [10] or speech [11, 19]. Ma, Milner, and Smith [4] note that a difference between speech recognition and acoustic environment classification is that assumptions can be made on the nature of speech (originating from a single source and with certain limitations), which one cannot in the latter case. A similar distinction is made in quality assessment and different algorithms exist for speech and for general sound.

ITU-T recommendation P.563 [19] is a non-intrusive algorithm for speech quality assessment. Simplified, it performs vocal tract analysis to classify the speech as originating from a male, female, or "strongly robotizied" voice, and then proceeds to identify issues with the recording. It also detects additive noise, unnatural silence, and clipping, among others. The overall score is then based on what issues were identified in the analysis and their severity.

Avila et al. [11] uses the recommendation as a baseline for comparison with their method, and note that as it was developed to test speech codecs it is outperformed by other algorithms in reverberant speech. They make no attempt to identify specific signal issues, instead training a network to predict the mean opinion score (MOS) that a human listener would assign a given sample. The same is true for Li et al. [10], who do not estimate the score directly but instead have users and classifiers rank differently degraded versions of the same content.

Rix et al. [18] describe the history of speech quality assessment up until 2006. Early approaches in non-intrusive assessment were based on estimating the distance from a given signal to the space of ideal speech signals. This was later augmented by HMM and GMM techniques, and vocal tract analysis to identify sounds and effects not caused by a human voice.

2 Experiment

In the experimental part of this pre-study, five classifier types were tested against each other to determine which are the most accurate and reliable when classifying additive noise. The noise was further assumed to be independent of the speech. The accuracy and reliability were evaluated on different datasets by repeated trials of each classifier.

In the first phase of the study the papers described above where sought out and read while consulting with the supervisors. Papers were located either through Google Scholar keyword searches, recommendations from supervisors, or through the citations of already-read papers. Basic dataset generation was then implemented using a Python version of the Audio Degradation Toolbox [20] by Hanssian [21] but later ported to use the original MATLAB implementation due to functionality concerns. Classifiers were chosen and implemented, as were tools to train and test them, after which the hyperparameter search was written and conducted. Lastly, the repeated classifier evaluation was implemented and conducted as the literature study section was written.

As the project included testing the GenHMM implemented in Python using PyTorch by Liu et al. [17], the rest of the classifiers and surrounding code was also written in Python. Dataset generation, however, also requires MATLAB.

In section 2.1 the data used during training and testing is described, and the features extracted are outlined in section 2.2. In section 2.3 the types of classifiers and the hyperparameter settings used are described. Finally, in section 2.4 results are presented and discussed.

2.1 Datasets

Six datasets were used to evaluate performance: `snr20`, `snr20_pad1`, `snr20_pad1_alt`, `snr20_pad1_white`, `snr40_pad1`, and `snr40_pad1_alt`. These were generated by splitting the recordings in TIMIT randomly into a testing and a training set, ignoring the existing partition. This means that a certain speaker may occur in both the training and testing set, as may a certain phrase, but never the same phrase uttered by the same speaker. Similarly, the recordings of noise in a certain category are split into training and testing sets. The noise categories considered in this experiment were air conditioning noise acquired from Freesound, electrical hum noise acquired from Freesound and from Sten Ternström's recordings, and white noise generated using the Audio Degradation Toolbox (ADT) [20]. A total of 15 air conditioning recordings (ca. 12.5 minutes) and 19 electric hum recordings (ca. 8.5 minutes) were used.

Each utterance was assigned a noise category and a random sample in that category was selected and added using the ADT (including *No noise added*, for which the utterance was taken as-is). Training noise samples were selected for training utterances and likewise for testing, thus no certain noise recording appears in both the training and the testing set. All samples in a dataset are normalized to avoid clipping and to ensure that an increased volume itself is not an indicator that noise is present. However, when an added noise sample is shorter than an utterance the noise sample is repeated, which for certain recordings causes an audible transient.[2]

The number following `snr` in the dataset names is the signal-to-noise ratio in dB, where the speech is considered signal. `pad1` indicates that 1 s of silence was appended to the beginning and end of a TIMIT utterance before adding the noise. The datasets which end with `alt` are generated with the same settings as those which do not, and only differ in the partitioning into training and testing sets. White noise was not included in any dataset except `snr20_pad1_white`.

The `snr20_pad1` dataset was also used during the hyperparameter search.

2.2 Features

The MFCC features were extracted using Kaldi [22] with the default settings of `compute-mfcc-feats`. The most notable parameter settings are shown in table 1. A VAD [23] was considered to be included in the feature extraction, and may be in future work, but it was not used during this experiment.

Parameter	Value	Note
Frame length	25 ms	
Frame shift	10 ms	
Window type	Povey	Similar to Hamming
Frequency range	20 Hz – 8 kHz	The sample rate was 16 kHz
Number of bins	23	
Number of cepstra	13	

Table 1: Settings used for feature extraction.

2.3 Classifier Types

A hyperparameter search was conducted to find the best parameter settings for each classifier. This was done by considering the default values in either related reports (GMMHMM, GenHMM, CNN) or documentation for the frameworks in which the classifiers were implemented (LSTM, SVM), and then increasing and decreasing the parameter. For instance, the number of states in the GMMHMM was varied between 2 and 6. Ideally every compatible combination of parameters would be repeatedly considered on multiple datasets, but time constraints would prevent some combinations from being tested. Thus random combinations were tested, and the search was left to run for as long as possible.

[2]This issue was later fixed with Sten Ternström's help, but as the evaluation was already in progress by then the old datasets were used for consistency.

The best performing parameter setting for each classifier was then used in the training detailed in section 2.4. During the hyperparameter search each combination was only evaluated once on a single dataset (`snr20_pad1`), and as the initializations of all models except the SVM[3] have some element of randomness this does not necessarily mean the chosen settings are guaranteed to be optimal in repeat trials. It can however be argued that they are the best guesses available.

2.3.1 Gaussian Mixture Model HMM (GMMHMM)

The GMMHMM models the time-dependence in the signal using a hidden Markov model with 3 states, and the probability distributions of MFCC vectors using 2 Gaussian mixture components with a diagonal covariance matrix. The use of diagonal covariance matrices in conjunction with MFCC features is motivated by Ma, Milner, and Smith [4], since the coefficients acquired after performing the Discrete Cosine Transform during feature extraction are uncorrelated. They also note that increasing the number of mixture components increases the amount of data needed for training, and settle for a single mixture component for their own purposes.

The GMMHMM is implemented by Liu et al. [17], whose implementation in turn is based upon the version in `hmmlearn`.

2.3.2 Generative Model HMM (GenHMM)

The GenHMM works similarly to the GMMHMM, with 3 states and 2 probability components, but the probability distributions are instead modeled by so-called generators to increase the modeling capacity. Let s denote the state and k a component. Each combination s, k has its own probability distribution which is realized by transforming a standard normal distribution. The transform is designed to be invertible, so that the model is easy to train and can compute the loglikelihood exactly.

Let $Z \sim \mathcal{N}(0, I)$ be a sample from the multivariate standard normal distribution in $\mathbb{R}^D$. To apply the transform we begin by partitioning Z into two parts Z_a, Z_b using a mask. Z_a is used as input into a neural network consisting of two hidden layers of dimension H with LeakyReLU activations and an output layer of dimension $2|Z_b|$. This yields a vector which is split into s_b and t_b, which we in turn use to calculate

$$Z'_b = (Z_b + t_b) \odot e^{s_b}, \tag{1}$$

where $\odot$ denotes element-wise multiplication. Trivially, (1) can be inverted via $Z_b = Z'_b \odot e^{-s_b} - t_b$. This operation is repeated by using Z'_b as input into an equivalent network to get s_a and t_a, which are used to calculate Z'_a in the same way. We now merge Z'_a and Z'_b using the same mask as during the partitioning to get Z'. This entire transform is denoted $Z' = g(Z)$, and the inverse is $Z = g^{-1}(Z') = f(Z')$. The theory behind these transforms, called *coupling layers*, are described in [24].

In the GenHMM we have n_{chain} different g per state s and component k, and the total transform is

$$X = (g_1 \circ g_2 \cdots \circ g_{n_{\text{chain}}})(Z), \tag{2}$$

where $\circ$ denotes function composition. For classification we do not need to sample using the GenHMM in this fashion, instead applying the inverse transform during training. The parameters we train in the GenHMM are both the regular HMM parameters and the neural network parameters in each transform. As we increase the number of parameters, however, we also increase the amount of data needed to properly train a classifier. As early as in 1995 Morgan and Bourlard [25] noted that hybrid HMM-neural network models require more data and computing power than discrete HMMs, which may be a limitation in this case as the data available is somewhat restricted (see section 2.1 on dataset generation).

The hyperparameters used in the experiments can be found in table 2.

[3] And, due to a bug at the time, the GMMHMM. This issue was resolved before the training detailed in section 2.4.

Parameter	Value	Note
H	24	Hidden dimension in g
D	12	
Mask type	cross	
n_{chain}	4	
p_{drop}	0	Dropout probabilities in g
`em_skip`	4	
`startprob_type`	first	
`transmat_type`	triangular	

Table 2: GenHMM hyperparameters. 3 states and 2 mixture components were used.

2.3.3 Long Short-term Memory (LSTM)

Like the HMMs, an LSTM layer can process variable-length input. A hidden state is initialized in the beginning of a sequence whereupon each MFCC vector in the sequence is input into the LSTM cell, updating the hidden state and giving an output vector. The last output vector of the sequence can then be used with a network which requires fixed-length input.

The LSTM is implemented in PyTorch and uses two layers with a hidden dimension of 20, followed by a fully connected $20 \times N_{\text{classes}}$ layer. `log_softmax` is then applied. The LSTM is trained using the Adam optimizer with a constant learning rate of 0.005, using Negative Log Likelihood (NLL) loss.

2.3.4 Convolutional Neural Network (CNN)

In this implementation of the CNN we use no HMM, RNN or other method which allows for sequences of varying length. We thus need to convert our signal into fixed-size objects to classify, which is done as follows. Each MFCC vector contains $M = 13$ elements, and for the CNN we accumulate $N = 30$ consecutive vectors and arrange these in a single $N \times M$ matrix. Thus for a given signal converted into an MFCC sequence we get a number of non-overlapping[4] MFCC matrices. Each such matrix, containing information about 315 ms of the signal, is labeled with the same class as the full sequence and fed into the CNN for training. During testing the signal is processed the same way and each matrix is classified individually. The most common class within a sequence's matrices is taken as the sequence's class.

The CNN is also implemented in PyTorch, and the idea for the design is based on Cakir et al. [8]. Two convolutional layers are used: one which is 1×3 channels and another which is 3×6 channels, both with a 4×3 size kernel and a zero-padding of 2 in each dimension. After each convolutional layer we apply a ReLU-nonlinearity followed by a 3×3 max pooling operation. The channels are stacked and the entire vector is fed into a fully connected network with ReLU-activations, a single hidden layer of size 32 and an output dimension equal to the number of classes. `log_softmax` is then applied. As the convolutional layers themselves allow for variable-sized input one can combine them with e.g. an LSTM, which would remove the need to split the signal into frames, though this was not tested.

The CNN is trained using the Adam optimizer with a constant learning rate of 0.03, using NLL loss.

2.3.5 Support Vector Machine (SVM)

The MFCCs are preprocessed in a similar way to the CNN, except instead of arranging them into a matrix we concatenate them into a single $N \cdot M$ vector. Here, $N = 20$ and the frames do overlap as we use a stride of 15 MFCC vectors. As with the CNN the chosen class of a signal during testing is the class which is most common among its frames. Before being passed to the SVM they

[4]Overlapping was tested during the hyperparameter search but not found to be required.

are normalized to have a mean of 0 and variance of 1 in each dimension. The same normalization (learned from the training set) is applied during testing.

The SVM classifier is a wrapper for the `sklearn` SVC implementation, and uses radial basis functions, a regularization parameter $C = 1.0$, and a kernel coefficient $\gamma = \frac{1}{NM}$. The hyperparameter search for the SVM was cut short as training times escalated, and classification itself was so slow that it would not be usable in a practical setting (exceeding 1 s for ∼5 s signals on an Intel i5-4690K 3.50GHz CPU, whereas other models achieved at most about 100 ms and can use a GPU).

2.4 Performance on different datasets

Once the classifiers had been implemented and hyperparameters chosen the evaluation itself was conducted. For each classifier type (except SVM) and each dataset, ten[5] instances were randomly initialized. These were subsequently trained for as many epochs as needed until the increase in mean F1-score on the test set for the ten instances was lower than 0.1 percentage points. The number of iterations per epoch $n_{\text{iter}} = 4$ for each classifier. A single SVM classifier was used for each dataset, as the SVM is only fit once for given data and does not (in this implementation) have any element of randomness.

The results are summarized in figure 1. As we have three classes in all datasets except `snr20_pad1_white`, which has four, we expect random guessing to yield an accuracy of 33% and 25%, respectively. Interestingly, the accuracy instead increases by more than 10 percentage points for the GMMHMM, LSTM and SVM on `snr20_pad1_white` when compared to `snr20_pad1` and `snr20_pad1_alt`. The overall increase can be explained by the fact that the statistics of white noise are very easy to model, and as it increases all frequencies equally it could most likely be detected by thresholding on the first dimension in the MFCCs.

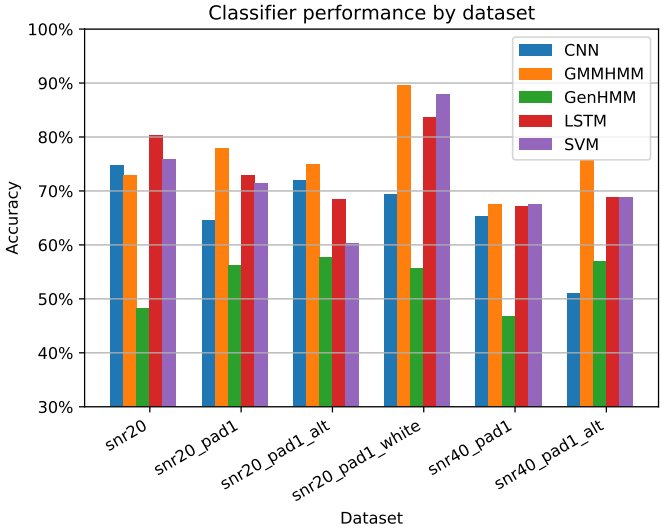
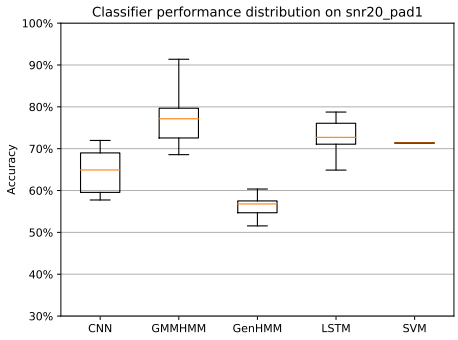


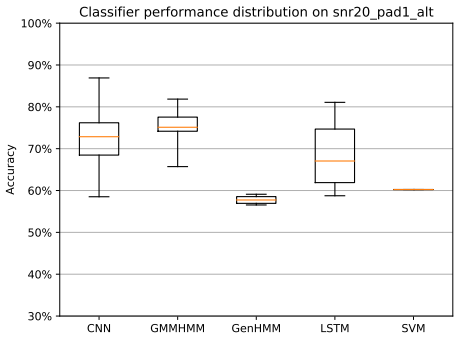
Figure 1: Average accuracy per dataset for each classifier type. The scores are evaluated at the second to last epoch.

In figures 2 and 3 the distributions of the accuracies on test sets are evaluated for different classifiers and datasets. We note the large spread of CNN accuracy on multiple datasets, the comparatively poor performance of GenHMM on all datasets, and the peak performance of one GMMHMM instance at ca. 97.5% on `snr20_pad1_white`.

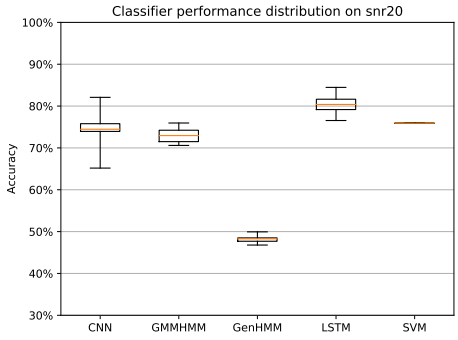
[5]Due to time constraints not all GenHMM instances had finished training on all datasets at the time of writing. `snr20` had ten completed instances while all other datasets had at least seven.



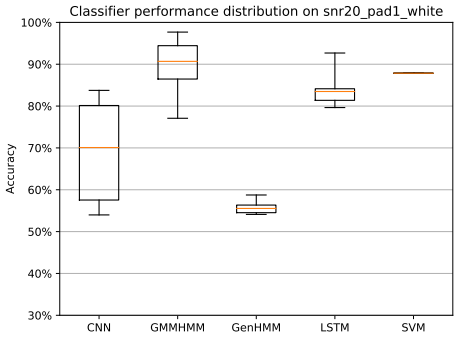
(a) Dataset `snr20_pad1`



(b) Dataset `snr20_pad1_alt`

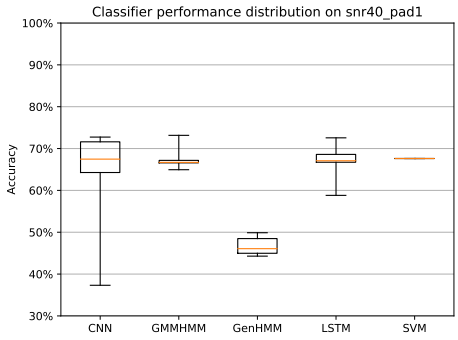


(c) Dataset `snr20`

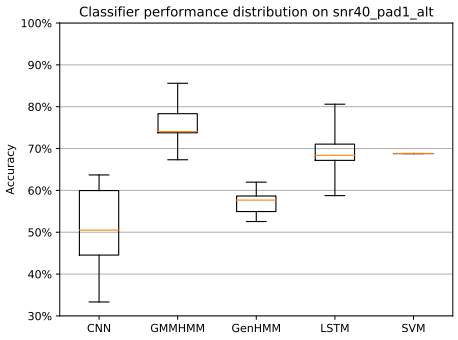


(d) Dataset `snr20_pad1_white`

Figure 2: Accuracy distribution of 10 classifiers[5] of each type (except in the SVM case), on the datasets with SNR = 20 dB. The whiskers show the minimum and maximum score in and the box shows the interquartile range, with an orange line at the median. The scores are evaluated at the second to last epoch.



(a) Dataset `snr40_pad1`

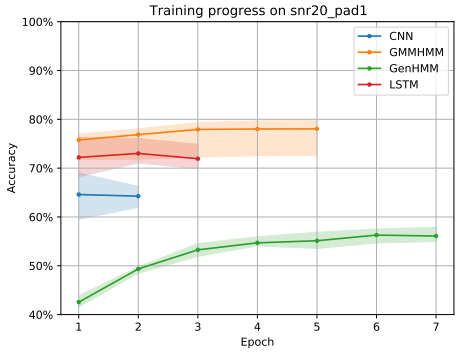


(b) Dataset `snr40_pad1_alt`

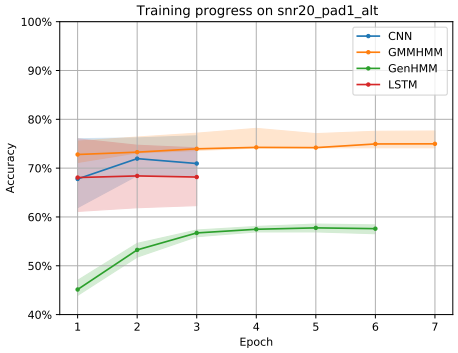
Figure 3: Accuracy distribution of 10 classifiers[5] of each type (except in the SVM case), on the datasets with SNR = 40 dB. The whiskers show the minimum and maximum score and the box shows the interquartile range, with an orange line at the median. The scores are evaluated at the second to last epoch.

In figures 4 and 5 we see how the accuracy on the testing set progresses during training for each classifier type. Note that this is not a comparison of performance over total training time as the time needed to train one epoch varies by classifier type. The SVM is not shown as its training

cannot be segmented in the same manner. We note that the LSTM, GMMHMM and CNN often converge within a small number of epochs, and a reduced learning rate may have been beneficial.

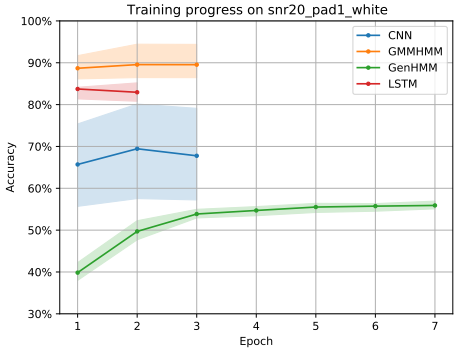


(a) Dataset `snr20_pad1`



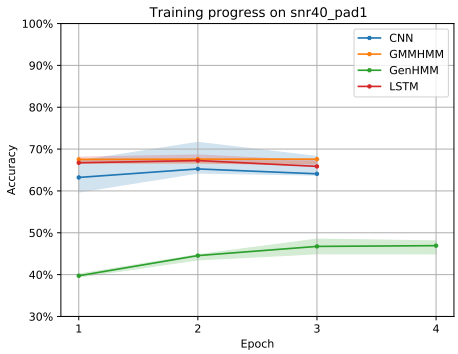
(b) Dataset `snr20_pad1_alt`

(c) Dataset `snr20`

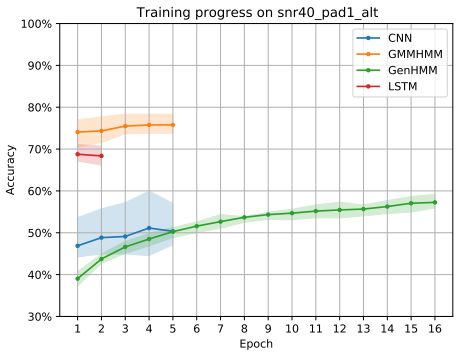


(d) Dataset `snr20_pad1_white`

Figure 4: Training progress of 10 classifiers[5] of each type except SVM, on the datasets with SNR = 20 dB. The line shows the mean accuracy and the surrounding coloring represents the interquartile range.



(a) Dataset `snr40_pad1`



(b) Dataset `snr40_pad1_alt`

Figure 5: Training progress of 10 classifiers[5] of each type except SVM, on the datasets with SNR = 40 dB. The line shows the mean accuracy and the surrounding coloring represents the interquartile range.

2.5 Conclusion

Both the LSTM and, in particular, the GMMHMM performed well in the experiment with both decent averages and spread. The CNN, while in some cases on par with the others, has a much bigger difference between its extremes and performs considerably worse on `snr40_pad1_alt`. The performance of the SVM is comparable with the GMMHMM and LSTM but it suffers from debilitating training and classification times. The GenHMM has a considerably lower accuracy than the rest of the models and is likely suffering from lack of variation in the training data, as it has a lot of parameters to train and much of the data is irrelevant. Thus future work in this project will focus on GMMHMM classifiers.

VAD filtering as a preprocessing step would be an interesting next iteration of this specific experiment. It should be noted that the results for all classifiers could be improved by tailoring feature extraction and hyperparameter settings more. The current settings used are based on empirical testing, but the reliability of this testing may be impacted by its lack of repetition.

References

[1] S. Qi et al. "Audio Recording Device Identification Based on Deep Learning". In: *2016 IEEE International Conference on Signal and Image Processing (ICSIP)*. 2016, pp. 426–431.

[2] Constantine Kotropoulos and Stamatios Samaras. "Mobile Phone Identification Using Recorded Speech Signals". In: *2014 19th International Conference on Digital Signal Processing*. IEEE. 2014, pp. 586–591.

[3] Adrien Leman and Julien Faure. "Method and Device for Classifying Background Noise Contained in an Audio Signal". US8972255B2. Orange SA. 2015-03. URL: `https://patents.google.com/patent/US8972255B2/en` (visited on 2020-04-06).

[4] Ling Ma, Ben Milner, and Dan Smith. "Acoustic Environment Classification". In: *ACM Trans. Speech Lang. Process.* 3.2 (2006-07), pp. 1–22. ISSN: 1550-4875. DOI: `10.1145/1149290.1149292`.

[5] Dongge Li et al. "Classification of General Audio Data for Content-based Retrieval". In: *Pattern Recognition Letters* 22.5 (2001), pp. 533–544. ISSN: 0167-8655.

[6] Panu Maijala et al. "Environmental Noise Monitoring Using Source Classification in Sensors". eng. In: *Applied Acoustics* 129 (2018), pp. 258–267. ISSN: 0003-682X.

[7] Christophe Couvreur et al. "Automatic Classification of Environmental Noise Events by Hidden Markov Models". eng. In: *Applied Acoustics* 54.3 (1998), pp. 187–206. ISSN: 0003-682X.

[8] Emre Cakir et al. "Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection". eng. In: *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)* 25.6 (2017), pp. 1291–1303. ISSN: 23299290.

[9] Md. Shamim Hussain and Mohammad Ariful Haque. "SwishNet: A Fast Convolutional Neural Network for Speech, Music and Noise Classification and Segmentation". In: (2018). arXiv: `1812.00149`.

[10] Zhonghua Li et al. "Non-reference Audio Quality Assessment for Online Live Music Recordings". In: *Proceedings of the 21st ACM international conference on Multimedia*. 2013, pp. 63–72.

[11] A. R. Avila et al. "Non-intrusive Speech Quality Assessment Using Neural Networks". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 631–635.

[12] K El-Maleh, A Samouelian, and P Kabal. "Frame Level Noise Classification in Mobile Environments". eng. In: *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*. Vol. 1. IEEE, 1999, 237–240 vol.1. ISBN: 0780350413.

[13] V. Peltonen et al. "Computational auditory scene recognition". In: *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 2. 2002, pp. II-1941-II–1944.

[14] Jongpil Lee et al. "Raw Waveform-based Audio Classification Using Sample-level CNN Architectures". In: *CoRR* abs/1712.00866 (2017). arXiv: 1712.00866. URL: http://arxiv.org/abs/1712.00866.

[15] Shawn Hershey et al. "CNN Architectures for Large-Scale Audio Classification". In: *CoRR* abs/1609.09430 (2016). arXiv: 1609.09430. URL: http://arxiv.org/abs/1609.09430.

[16] Caio Cesar Enside de Abreu, Marco Aparecido Queiroz Duarte, and Francisco Villarreal. "An Immunological Approach Based on the Negative Selection Algorithm for Real Noise Classification in Speech Signals". eng. In: *AEUE - International Journal of Electronics and Communications* 72 (2017), pp. 125–133. ISSN: 1434-8411.

[17] Dong Liu et al. "Powering Hidden Markov Model by Neural Network based Generative Models". In: *ECAI 2020* (2019).

[18] A.W Rix et al. "Objective Assessment of Speech and Audio Quality–Technology and Applications". eng. In: *IEEE Transactions on Audio, Speech, and Language Processing* 14.6 (2006), pp. 1890–1901. ISSN: 1558-7916.

[19] ITU-T. *Single-ended Method for Objective Speech Quality Assessment in Narrow-band Telephony Applications*. Tech. rep. Series P: Telephone Transmission Quality, Telephone Installations, Local Line Networks, 2004.

[20] Matthias Mauch, Sebastian Ewert, et al. "The Audio Degradation Toolbox and its Application to Robustness Evaluation". In: *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR 2013)*. 2013.

[21] Sevag Hanssian. *audio-degradation-toolbox*. GitHub repository. 2019-11-19. URL: https://github.com/sevagh/audio-degradation-toolbox.

[22] Daniel Povey et al. "The Kaldi Speech Recognition Toolkit". In: *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Catalog No.: CFP11SRW-USB. Hilton Waikoloa Village, Big Island, Hawaii, US: IEEE Signal Processing Society, 2011-12.

[23] John Wiseman. *py-webrtcvad*. GitHub repository. 2019-05-17. URL: https://github.com/wiseman/py-webrtcvad.

[24] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density Estimation Using Real NVP". In: (2016).

[25] N Morgan and H.A Bourlard. "Neural Networks for Statistical Recognition of Continuous Speech". eng. In: *Proceedings of the IEEE* 83.5 (1995), pp. 742–772. ISSN: 0018-9219.