

A Three-Fold Machine Learning Approach for Detection of COVID-19 from Audio Data

Nikhil Kumar, Vishal Mittal, and Yashvardhan Sharma(✉)

Birla Institute of Technology and Science, Pilani, RJ 333031, India
{f20170658,f20170080,yash}@pilani.bits-pilani.ac.in

Abstract. Most work on leveraging machine learning techniques has been focused on using chest CT scans or X-ray images. However, this approach requires special machinery, and is not very scalable. Using audio data to perform this task is still relatively nascent and there is much room for exploration. In this paper, we explore using breath and cough audio samples as a means of detecting the presence of COVID-19, in an attempt to reduce the need for close contact required by current techniques. We apply a three-fold approach of using traditional machine learning models using handcrafted features, convolutional neural networks on spectrograms and recurrent neural networks on instantaneous audio features, to perform a binary classification of whether a person is COVID-positive or not. We provide a description of the preprocessing techniques, feature extraction pipeline, model building and a summary of the performance of each of the three approaches. The traditional machine learning model approaches state-of-the-art metrics using fewer features as compared to similar work in this domain.

Keywords: Audio processing · Machine learning · Classification · Convolutional neural networks · Recurrent neural networks · COVID-19

1 Introduction

Our aim is to detect the presence of COVID-19, given an audio sample of a person either breathing or coughing. This is essentially a binary classification task (`normal`: 0, `covid`: 1). There are several features that can be extracted from an audio sample and fed into machine learning (ML) models to perform the classification task. This has been the traditional method of feeding audio data into ML models, i.e., by extracting features by hand.

Alternatively, neural networks have been shown to perform well on classification tasks [7], while also not requiring hand-crafted features. Convolutional neural networks (CNNs) have been around since the 1990s [6], and have been used for face and object detection tasks. The biggest advantage of using CNNs is that filters need not be handcrafted, as they are learnt by the network. Interest was rekindled upon seeing their success at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [5].

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12951, pp. 497–511, 2021.
https://doi.org/10.1007/978-3-030-86970-0_35

Another class of neural networks is the recurrent neural network (RNN), which has seen success with sequence data, such as text processing, speech processing and time-series forecasting [7]. Long short-term memory networks (LSTMs) are a special kind of RNN, containing memory cells to keep track of information over a sequence [3]. We explore the use of LSTMs to process breath and cough audio sequences in this project, owing to their success on sequence data.

2 Related Work

2.1 COVID-19 Sounds Project at the University of Cambridge

Brown et al. developed an application to crowdsource audio samples of COVID-positive and COVID-negative individuals [1]. Two modalities were used – breath and cough audio samples. The data was divided broadly into individuals with asthma (but not COVID-positive), COVID-positive individuals and healthy individuals. The COVID-negative class was subdivided into individuals who reported having cough as a symptom and those who did not.

Two kinds of models were applied – traditional machine learning models on handcrafted features and a convolutional model based on VGG-16, made for audio data called VGGish. 477 features were extracted using a combination of global and instantaneous features, along with their statistics, and fed into traditional ML models. Spectrograms of the audio samples were extracted and fed into VGGish.

Three separate binary classification tasks were performed, aimed at differentiating between different combinations of COVID-positive and COVID-negative individuals, taking other meta-information (such as whether the patient reported having a cough or asthma) into account. However, our work will focus on the plain vanilla task of distinguishing COVID-positive and normal samples from the raw audio data itself without using the metadata.

2.2 Coswara Project at the Indian Institute of Science

Researchers at the Indian Institute of Science worked on collecting a database of cough, breath and voice samples called Coswara [15]. Several standard traditional audio processing features were used, and traditional machine learning models were fitted. Their paper discusses the results they obtained on a random forest classifier on a 9-way classification problem.

2.3 Approach Using MFCCs and CNNs at the University of Manchester

Dunne et al. built a CNN classifier using the Mel-cepstral frequency coefficients, which gave an accuracy of 97.5% on a small dataset and a 2-way classification problem of COVID-positive vs. normal [2].

However, we will not be considering accuracy as the only metric in our project, given the highly unbalanced nature of the dataset we are using. Precision and recall are considered more reliable metrics in disease-detection tasks, as compared to raw accuracy scores.

2.4 What Is New in This Project?

Given the sequential nature of the audio data, we leveraged recurrent neural networks. To the best of our knowledge, there has been very little to no work done in using RNNs for this task (one of the few works is [13], which uses a combination of transformers and RNNs).

3 Dataset Used

We obtained all the data used in this project from the University of Cambridge's COVID-19 Sounds Project [1] under an academic license. The dataset contained 1134 breath samples and 1135 cough samples. Each audio type was further divided into:

- Asthma samples (COVID-negative), all samples reporting having a cough
- COVID-positive samples, reporting having a cough
- COVID-positive samples, reporting not having a cough
- Healthy samples, reporting having a cough
- Healthy samples, reporting not having a cough

The samples were collected by [1] from two sources – an Android application and a web application. We merged the two sources together using some simple Python scripts. We also combined the COVID with cough and COVID without cough samples together, and the healthy with cough and healthy without cough together, since our goal is binary classification of `covid` vs. `normal`, and not obtaining the predictions given whether we know someone reports having cough or not. The task at hand (i.e., binary classification into `covid` and `normal`) does not require the class `asthma`, so we discarded those samples.

The distribution of `asthma`:`covid`:`normal` samples in both breath and cough data was roughly 15 : 15 : 70. We maintained a similar distribution in the dataset split into train, validation and test sets (more on this in Sect. 5.1).

4 Proposed Technique

We use a three-fold approach to detect COVID-19 using:

- Traditional ML model using support vector machine (SVM)
- CNN on mel-spectrograms
- LSTM on sequential audio features

We used 108 features for the machine learning models. All features for these models were handcrafted, and are the typical set of features that are used in audio processing. We used the Librosa [9] library in Python for the signal processing tasks. A detailed description of the features used is given in Sect. 6.2.

For the CNN model, we brought all audio samples to the same length and then generated the log-scaled-spectrograms and mel-spectrograms for the audio samples. More information about this procedure is given in Sect. 5.2.

For the LSTM model, we split the audio sample into frames and extracted features across each frame, giving us an array of feature sequences, with shape [`num_timesteps`, `num_features`].

5 Data Preprocessing

5.1 Partitioning the Data

For the ML model, we performed an 80 : 20 split of the data into train and test. We did not keep a hold-out validation set, but performed a 10-fold cross validation, owing to the small size of the dataset. For the CNN and LSTM models, we performed an 70 : 15 : 15 split of the data into train, validation and test. In both cases, we ensured that the original distribution of each class (`asthma`, `covid` and `normal`) was maintained while performing the split (Tables 1, 2).

Table 1. Train-test split (80 : 20) for ML model)

Class	Breath		Cough	
	Train	Test	Train	Test
`asthma`	133	34	134	34
`covid`	137	35	142	36
`normal`	636	159	631	158

Table 2. Train-validation-test split (70 : 15 : 15) for CNN and LSTM models

Class	Breath			Cough		
	Train	Validation	Test	Train	Validation	Test
`asthma`	133	17	17	134	17	17
`covid`	137	17	18	142	18	18
`normal`	636	79	80	631	79	79

Note that in each column, `asthma`:`covid`:`normal` is approximately 15 : 15 : 70, which is the same as for the original distribution.

5.2 Loopback and Clipping

The audio samples all have varying lengths (as shown in Fig. 1). This does not cause a problem for the traditional machine learning models because we consider aggregate features for the whole audio sample, so there is no time sensitivity. However, it causes problems for the convolutional and recurrent models as:

- All the spectrograms have the same dimension in pixels. A spectrogram has time along the x-axis. However, we cannot have different scales on the x-axis for different images (If the audio samples have different lengths, and the length of the x-axis is the same for all images, the scale converting from x-axis units to time in seconds will be different for each image).
- Each audio sequence must have the same length as LSTMs require input of shape [`num_samples`, `num_timesteps`, `num_features`].

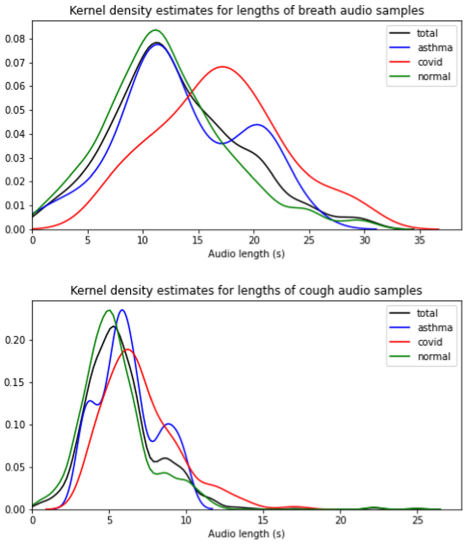


Fig. 1. Distribution of lengths of audio samples

6 Feature Extraction

6.1 Types of Features

An audio signal is an analog signal. However, to store an audio signal as a file on a computer, it needs to be converted into a digital signal through sampling and quantization. Sampling is the discretization of time and quantization is the discretization of the audio intensity.

- The number of samples taken in a unit interval of time is the sampling rate. Alternatively, the duration of each sample is the reciprocal of the sampling rate. A common value of sampling rate is 44100Hz. Human hearing has an upper bound at around 22050Hz. The Nyquist-Shannon theorem [14] tells us that to extract reliably a maximum frequency f from the signal and to avoid aliasing, we need to sample it with a minimum frequency of $2f$.
- An n-bit audio clip uses 8 bits used for each audio level. So an 8-bit audio clip has $2^8 = 256$ levels. Higher the value of n, higher the resolution of audio levels.

We used a sampling rate of 16000 Hz (= 2 × 8000 Hz), because the exploratory data analysis revealed that most of the significant frequencies were found up to 8000 Hz. Researchers working on [1] used the same sampling rate, on the same dataset. There are two kinds of features that are used in audio signal processing [12] – instantaneous and global.

Instantaneous Features. The audio sample is divided into frames. Frames can overlap with each other. Frame size is the number of samples within each frame. The distance between consecutive frames is called the hop length. The ends of the audio clip are padded to allow for framing over the edges. Audio features are then calculated over these frames, giving a sequence of features over the timesteps.

The relation between number of samples in the audio (`audio_len`), number of samples in the frame (`frame_len`), padding used on each side (`pad_len`), hop length (`hop_len`) and the length of the output vector (`output_len`) is

$$\texttt{output_len} = \left\lfloor \frac{\texttt{audio_len} + 2 \times \texttt{pad_len} - \texttt{frame_len}}{\texttt{hop_len}} \right\rfloor + 1 \tag{1}$$

Librosa [9] by default uses:

$$\texttt{pad_len} = \left\lfloor \frac{\texttt{frame_len}}{2} \right\rfloor \tag{2}$$

Values of the above parameters we used in our work are $\texttt{frame_len} = 256$, $\texttt{pad_len} = \lfloor \frac{256}{2} \rfloor = 128$ and $\texttt{hop_len} = 64$.

Instantaneous features are perfect for LSTMs. CNNs too can use these features in the form of spectrograms, which are calculated using an algorithm called the short-time Fourier transform (STFT). STFT uses the same idea of calculating over frames.

We did not use instantaneous features for the traditional machine learning models for two reasons:

- The mean audio length is 13.04s for the breath samples and 5.82s for the cough samples. Approximating the mean audio length as 10s, using sampling rate as 16000Hz and the parameters defined in Sect. 6.1, a simple back-of-the-envelope calculation using Eq. 2 yields an output sequence of length 2000+ timesteps. These are too many features to pass into a traditional ML model (curse of dimensionality).

– Passing timestep features to a traditional ML model is not useful as it cannot pick up on the sequencing.

Global Features. Global features are features that are calculated over the complete audio sample, i.e., no framing. These are well-suited for traditional machine learning models. We have not used global features for our models.

Aggregate Instantaneous Features. Since we cannot pass the raw instantaneous feature sequences extracted from the audio samples into the traditional ML models, we calculated statistics across the sequence and used these as input (since we get a single value for each audio sample that summarizes the sequence for us).

We used 18 instantaneous features summarized using mean, median, root mean square value, maximum, minimum and RMS energy weighted mean, giving us $18 \times 6 = 108$ features.

6.2 Features Used

We shortlisted commonly used features in audio processing tasks ands extracted them from the data:

Root Mean Squared Energy (RMSE). It gives an idea of how much energy is contained in the signal [9]. RMSE for a frame is given by

$$\text{RMSE}_{\text{frame}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2} \tag{3}$$

where i iterates over the samples in the frame, n is the number of samples in the frame and y_i is the signal value for the ith sample in the frame.

Zero Crossing Rate (ZCR). It is the number of times a signal crosses the y-axis in a time interval. A high ZCR indicates noisy data [12]. ZCR for a frame is given by

$$\text{ZCR}_{\text{frame}} = \frac{1}{2}\sum_{i=0}^{n-1} \mid \text{sgn}(y_{i+1}) - \text{sgn}(y_i) \mid \tag{4}$$

where the indices are the same as for Eq. 3 and sgn is the signum function.

Spectral Centroid (SC). It gives an idea about what value of frequency f most of the frequencies are distributed [12]. One can think of it as a measure of central tendency or the centre of mass of the frequencies present in the Fourier transform of the signal. SC for a frame is given by

$$\mathrm{SC}_{\mathrm{frame}} = \frac{\sum_{j=1}^{m} M_j f_j}{\sum_{j=1}^{m} M_j} \tag{5}$$

where j iterates over the frequencies in the frame, m is the number of frequencies in the frame, M_j is the magnitude corresponding to the jth frequency f_j in the frame.

Spectral Bandwidth (SB). It gives an idea about the spread of the frequencies around a mean value [9], like a measure of dispersion such as variance or standard deviation. We used the 2nd order spectral bandwidth ($p = 2$), which is the full width at half-maximum of the plot of power vs. frequency. SB of order 2 for a frame is given by

$$\mathrm{SB2}_{\mathrm{frame}} = \frac{1}{2} \sum_{j=1}^{m} M_i (f_i - \mathrm{SC}_{\mathrm{frame}})^2 \tag{6}$$

Spectral Rolloff (SR). It is the frequency below which $p\%$ of energy of the spectrum for a frame lies [1]. p is a threshold, typical values for p are 85 and 95 [12]. We used $p = 85$ for our work.

Mel-Frequency Cepstral Coefficients (MFCCs). MFCCs are obtained by taking the discrete cosine transform (DCT) of the mel-spectrogram. MFCCs are one of the most popular features in audio processing [1]. Typically, the first 13 coefficients are used.

6.3 Spectrograms

The audio signal $y(t)$ in the time domain does not directly give us information about its frequency content. To obtain the frequency and their magnitudes, one has to take the Fourier transform of $y(t)$. This is represented by $y(\omega) = \mathcal{F}[y(t)]$. However, in this process, we lose the time information as the Fourier transform is calculated over the complete audio signal.

A solution to maintain both information related to time and frequency is to calculate the Fourier transform over frames of the audio signal (as described in Sect. 6.1).

This gives us the magnitude of the frequencies as a function of time $M(f, t)$. A spectrogram is a heatmap of function M, usually with time along the x-axis and frequency along the y-axis.

The Fourier transform assumes that the signal tapers off and goes to zero at the edges. However, this is not true for the individual frames, so we apply a windowing function (Hann window) that makes this condition true. Keeping

a hop length less than frame length (i.e., having overlapping frames) helps to avoid the loss of information at the edges of a frame due to windowing.

After performing the STFT on the audio signal, we changed:

- The frequency scale to the mel scale
- The magnitude of the frequencies into a logarithmic scale (decibels).

This is because a lot of the information is concentrated at the lower frequencies, so the magnitudes of the higher frequencies appear almost 0. The scaling helps visualize the magnitudes of the lower frequencies clearly.

The mel scale attempts to model human pitch perception, as the variation between perceived pitch and frequency is not linear. Rather, it is logarithmic. One of many empirical formulae to convert frequency from Hz to mels is as follows [10]:

$$m = 2595 \log_{10} \left(1 + \frac{f}{700}\right) \tag{7}$$

6.4 Input Vectors to LSTMs

Inputs to LSTMs have the shape [`num_samples`, `num_timesteps`, `num_features`]. We have considered only instantaneous features as inputs to the LSTM, because we want the time-ordering to be preserved.

7 Experiments and Results

The models[1] we used are described in detail below, organized by the audio type (breath or cough).

7.1 SVM Model

We used Scikit-learn [11] to train the SVM.

Procedure:

1. Applied principal component analysis (PCA) on the data to reduce the dimensionality of the data, using 20 components (which explained 84% of the variance).
2. Passed the output from the PCA into a support vector machine classifier with a radial basis function (RBF) kernel.
3. Performed grid search with 10-fold cross validation using the hyperparameters below:
   - C: L2 regularization parameter for SVM. Lower values of C give higher regularization.
   - γ: Coefficient for the kernel.
4. Refitted model on all 10 folds using the best hyperparameters found.

Results on test set are detailed below. We have taken `covid` as the positive class to calculate metrics (Table 3).

[1] Source code can be found at www.github.com/nikhilkmr300/covid-audio and www.github.com/vismit2000/covid-audio.

Table 3. Performance (SVM)

		P	
		normal (0)	covid (1)
T	normal (0)	157	2
	covid (1)	6	29

(a) Breath

		P	
		normal (0)	covid (1)
T	normal (0)	153	4
	covid (1)	1	34

(b) Cough

Metric	Breath	Cough
Precision	0.935	0.895
Recall	0.828	0.971
F1	0.879	0.931
Accuracy	0.959	0.976

(c) Metrics

T = True **P** = Predicted

7.2 CNN Model

We were able to obtain excellent results using an SVM by directly using the imbalanced dataset, however the CNN model was highly sensitive to the class imbalance. Deep neural networks are also prone to overfitting on small datasets, as is in this case, so we used a relatively simple architecture.

The model, however, learned to predict all samples as `normal` because of the heavy class imbalance, leading to an accuracy equal to the percentage of `normal` samples in the dataset and a recall of 0. We tried the following approaches to remedy the situation:

- Giving a higher weight to the minority class.
- Undersampling the majority class.

neither of which caused any significant improvement.

The poor performance of the CNN can be attributed to:

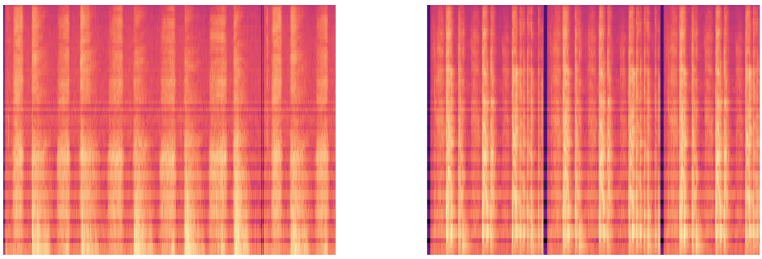
- Tiny dataset consisting of 1000 samples. Augmentation can help only to a certain extent.
- Large class imbalance between `covid` and `normal` samples.

both of which point to more data being required.

Another reason could also be the fact that the mel spectrograms of the `covid` and `normal` classes, at least to the human eye, look quite similar (refer Fig. 2). While it could be argued that the model can capture hidden features that the human eye misses, after some trials it does not seem that the CNN is capable of extracting from the raw spectrograms significant information to discern between the classes.

The poor performance of the CNN can be understood in the context of [1]. Brown et al. [1] used the pretrained VGGish model to extract 256 features from a spectrogram. They also mention that since VGGish takes a spectrogram as input, there might be a loss of temporal information, and hence they use the VGGish features along with 477 handcrafted features. In our case, with only a

raw spectrogram as input and a CNN with a straightforward architecture, post factum, it almost seems unreasonable to expect it to perform well. It appears that until more data is available, tweaking the CNN model is going to yield marginal results at best.



(a) `covid` (b) `normal`

Fig. 2. Mel spectrograms of a COVID-positive vs. a normal breath audio sample

The procedure we followed is detailed below:

1. Resized images to $(128, 128)$ using bilinear interpolation to reduce computational cost.
2. Performed augmentation on already generated mel spectrograms, shifting it upto a range of 0.2 times the width. This only affects the time axis, which preserves semantics, because we can shift the audio forward and backward without affecting the class of the sample or meaning of the spectrogram. However, if we shift the y-axis, we would be changing the frequency values on the y-axis, and then the spectrograms would not be comparable. Rotating, zooming and cropping the spectrogram are not meaningful transformations.

Table 4. CNN architecture

Layer	Activation	Layer information
Convolution 2D	relu	$f_c = 16, f_s = (5, 5)$
Max pooling 2D	None	$f_s = (2, 2)$
Convolution 2D	relu	$f_c = 32, f_s = (5, 5)$
Max pooling 2D	None	$f_s = (2, 2)$
Convolution 2D	relu	$f_c = 64, f_s = (3, 3)$
Max pooling 2D	None	$f_s = (2, 2)$
Flatten	None	None
Fully connected	relu	$n = 32$
Dropout	None	$p = 0.5$
Fully connected	sigmoid	$n = 1$

f_c = Filter count, f_s = Filter/pool size
p = Dropout probability, n = Number of neurons

3. Trained the model using the architecture detailed below with early stopping, with the Adam optimizer [4] with an initial learning rate of 1×10^{-4}, decreasing it by a factor of 0.05 on plateauing of validation loss (Table 4).

7.3 LSTM Model

The LSTM model, like the CNN model, was highly sensitive to the imbalanced nature of the data, and was incentivized to predict all samples as being of class `normal`.

Considering that deep networks are highly prone to overfitting with small datasets, we used relatively simple architectures. However, the downsampling of the majority class caused a performance hit, which is possibly the reason why the LSTM is unable to perform as well as the SVM (Tables 5, 6).

Metrics on the breath data have very noisy curves when plotted against epoch. Currently the LSTM model is performing, extremely poorly, slightly better than random chance (AUC = 0.569). It is probably due to the small size of the dataset, rendered even smaller because of downsampling.

The LSTM model performs significantly better on the cough data than on the breath data.

Table 5. LSTM architecture

Layer	Activation
LSTM ($u = 32$)	tanh
LSTM ($u = 32$)	tanh
FC ($n = 32$)	relu
FC ($n = 1$)	sigmoid

(a) Breath

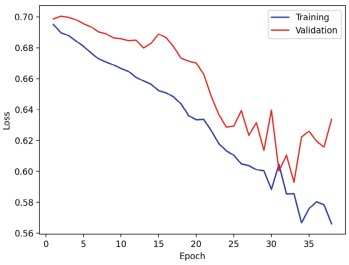
Layer	Activation
LSTM ($u = 64$)	tanh
LSTM ($u = 64$)	tanh
FC ($n = 64$)	relu
FC ($n = 1$)	sigmoid

(b) Cough

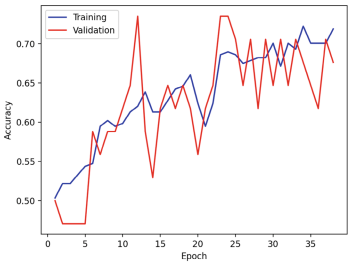
u = Number of hidden units, n = Number of neurons FC = Fully connected layer

Table 6. Test metrics (LSTM)

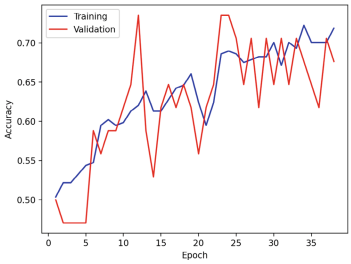
Metric	Breath	Cough
Precision	0.600	0.750
Recall	0.667	0.833
AUC	0.569	0.802
Accuracy	0.611	0.778



(a) Loss

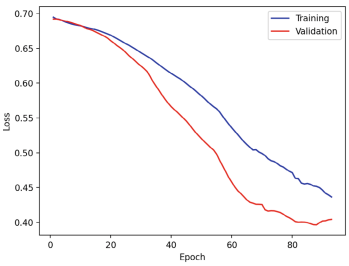


(b) Accuracy

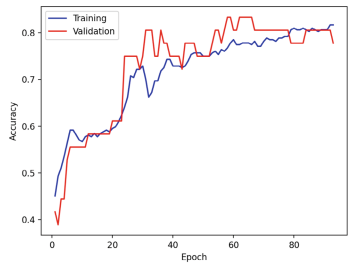


(c) AUC

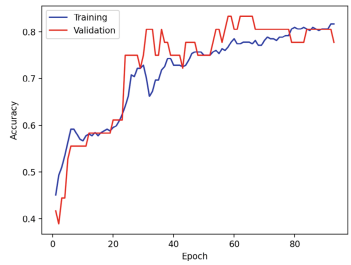
Fig. 3. Training and validation metrics for breath data



(a) Loss



(b) Accuracy



(c) AUC

Fig. 4. Training and validation metrics for breath data

8 Conclusion

8.1 Major Takeaways

The major takeaways from this work are as follows:

- The traditional machine learning model shows promise, scoring well on precision and recall, which are better indicators of performance in medical applications than the usual accuracy.
- Feeding models partially processed data in the form of aggregate metrics over raw time sequences could be more useful, as evidenced by the fact that the traditional ML model far outperforms the LSTM models using raw instantaneous features. One could think of this as the model being able to process partially *metabolized* data over the raw data.
- The CNN is unable to differentiate the classes from the raw spectrograms, an attestation to the previous point.

8.2 Comparison to Work by Brown et al.

Brown et al. [1] used a combination of 256 features extracted using VGGish, 477 handcrafted features and other combined features to get a total of 773 features, which were then reduced to a smaller set of features by PCA. Task 1 in [1] is in line with the work done in this paper. They obtained a precision of 0.72 and a recall of 0.69 on task 1. It is, however, to be noted that they used only 298 breath samples and 141 cough samples, as opposed to the 773 breath samples and 773 cough samples used by us for training. We used a larger amount of data, with fewer features (our 108 vs. their 773 before PCA), and obtained improved a precision metric of 0.935 (breath) and 0.895 (cough) and a recall of 0.828 (breath) and 0.971 (cough) (refer Sect. 7.1).

8.3 Future Work

The performance of each of the three models can be improved as more data becomes available. Once the models approach similar performance to each other, they can be combined together to create an ensemble. Ensembling has shown promise in the past and can be used to obtain performance better than any of the constituent models [8].

Acknowledgements. The authors would like to convey their sincere thanks to the Department of Science and Technology (ICPS Division), New Delhi, India, for providing financial assistance under the Data Science (DS) Research of Interdisciplinary Cyber Physical Systems (ICPS) Programme [DST/ICPS/CLUSTER DataScience/2018/Proposal-16:(T-856)] at the Department of Computer Science, Birla Institute of Technology and Science, Pilani, India. The authors are also thankful to the authorities of Birla Institute of Technology and Science, Pilani, for providing basic infrastructure facilities during the preparation of the paper.

References

1. Brown, C., et al.: Exploring automatic diagnosis of COVID-19 from crowdsourced respiratory sound data. CoRR abs/2006.05919 (2020). https://arxiv.org/abs/2006.05919
2. Dunne, R., Morris, T., Harper, S.: High accuracy classification of COVID-19 coughs using mel-frequency cepstral coefficients and a convolutional neural network with a use case for smart home devices (August 2020). https://doi.org/10.21203/rs.3.rs-63796/v1
3. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
4. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015, Conference Track Proceedings (2015). http://arxiv.org/abs/1412.6980
5. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017)
6. LeCun, Y., et al.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**(4), 541–551 (1989). https://doi.org/10.1162/neco.1989.1.4.541
7. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
8. Maclin, R., Opitz, D.W.: Popular ensemble methods: an empirical study. CoRR abs/1106.0257 (2011). http://arxiv.org/abs/1106.0257
9. McFee, B., et al.: librosa: Audio and music signal analysis in python. In: Proceedings of the 14th Python in Science Conference, vol. 8, pp. 18–25 (2015)
10. O'Shaughnessy, D.: Speech Communication: Human and Machine. Addison-Wesley series in electrical engineering, Addison-Wesley Publishing Company, Boston (1987). https://books.google.co.in/books?id=mHFQAAAAMAAJ
11. Pedregosa, F., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**(85), 2825–2830 (2011). http://jmlr.org/papers/v12/pedregosa11a.html
12. Peeters, G.: A large set of audio features for sound description (similarity and classification) in the cuidado project. CUIDADO IST Proj. Rep. **54**, 1–25 (2004)
13. Pinkas, G., Karny, Y., Malachi, A., Barkai, G., Bachar, G., Aharonson, V.: Sars-cov-2 detection from voice. IEEE Open J. Eng. Med. Biol. **1**, 268–274 (2020). https://doi.org/10.1109/OJEMB.2020.3026468
14. Shannon, C.E.: Communication in the presence of noise. Proc. IRE **37**(1), 10–21 (1949). https://doi.org/10.1109/JRPROC.1949.232969
15. Sharma, N., et al.: Coswara - a database of breathing, cough, and voice sounds for COVID-19 diagnosis. Interspeech 2020 (October 2020). https://doi.org/10.21437/interspeech.2020-2768. http://dx.doi.org/10.21437/Interspeech.2020-2768