A Three-Fold Machine Learning Approach to Detecting COVID-19 from Audio Data

1st Nikhil Kumar

Department of Computer Science & Information Systems
Birla Institute of Technology & Science, Pilani
Pilani, India
f20170658@pilani.bits-pilani.ac.in

2nd Vishal Mittal

Department of Computer Science & Information Systems
Birla Institute of Technology & Science, Pilani
Pilani, India
f20170080@pilani.bits-pilani.ac.in

***Abstract*—This year has brought medical professionals and data scientists the world over in search of a way to tackle the COVID-19 pandemic. A lot of work has been done in detecting the presence of COVID-19 using chest X-ray images. However, this approach requires special machinery, and is not very scalable. Not to mention that it also requires individuals to visit diagnostic centres, which can be a potential hotspot for the spread of the virus. Another common approach is to use thermal images to detect the presence of fever. However this method is too general, as a fever can mean several things, and not just COVID-19. Using audio data to perform this task is still relatively nascent and there is much room for exploration. Here we explore using breath and cough samples as a means of detecting the presence of COVID-19, to do away with the drawbacks of the other two methods. 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.**

***Index Terms*—audio processing, machine learning, convolutional neural networks, recurrent neural networks**

I. Introduction

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

Alternatively, neural networks have been shown to perform well on classification tasks [1], while also not requiring handcrafted features. Convolutional neural networks (CNNs) have been around since the 1990s [2], 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 [3].

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 [1]. Long short-term memory networks (LSTMs) are a special kind of RNN, containing memory cells to keep track of information over a sequence [4]. Gated recurrent units (GRUs) are a type of RNNs proposed in 2014, which offer comparable performance to LSTMs at lower computational cost [5]. Owing to their success on sequence data, we consider using RNNs to process breath and cough audio sequences in this project.

II. Related Work

A. COVID-19 Sounds Project at the University of Cambridge

Mascolo, Cicuta, et al. developed an application to crowdsource audio samples of COVID-positive and COVID-negative individuals [6]. 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. All the asthma samples reported having cough as a symptom. 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. However, our work will only focus on distinguishing COVID-positive and COVID-negative samples.

B. 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 [7]. Several standard traditional audio processing features were used, and traditional machine learning models fitted. Their paper discusses the results they obtained on a random forest classifier on a 9-way classification problem.

C. Approach Using MFCCs and CNNs at the University of Manchester

Dunne, Morris and Harper 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 vs non-COVID [8].

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.

D. What Is New in This Project?

We are working on using recurrent neural network models, given the sequential nature of the data. There has been very little to no work that has been done exploring the possibilities of using RNNs for this task (one of the few works is [9], which uses a combination of transformers and RNNs).

III. Dataset Used

We obtained all the data used in this project from the University of Cambridge's COVID-19 Sounds Project under an academic licence. 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 [6] 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. However, we still maintained the filenames which contain this information (i.e., reporting cough as a symptom or not), and used it

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 the Partitioning the Data section).

IV. Proposed Technique

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

- Traditional machine learning models like support vector machines and random forest
- Convolutional neural networks on mel-spectrograms
- Recurrent neural networks 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 0.8.0` [10] library in Python for all our feature extraction tasks. A detailed description of the features used is given in the Data Preprocessing section.

For the convolutional models, we brought all audio samples to the same length generated the log-scaled-spectrograms and mel-spectrograms for the audio samples. More information about this procedure is given in the Loopback and Clipping subsection.

For the recurrent models, we split the audio sample into frames and extracted features across each frame, giving us an array of feature sequences, with shape $[\texttt{num\_timesteps}, \texttt{num\_features}]$.

V. Data Preprocessing

A. 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 RNN models, we performed an $75 : 15 : 15$ 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.

TABLE I
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 II
Train-validation-test split ($70 : 15 : 15$) for CNN and RNN 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.

B. Loopback and Clipping

The audio samples all have varying lengths (as shown in Figure 1 above.

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:

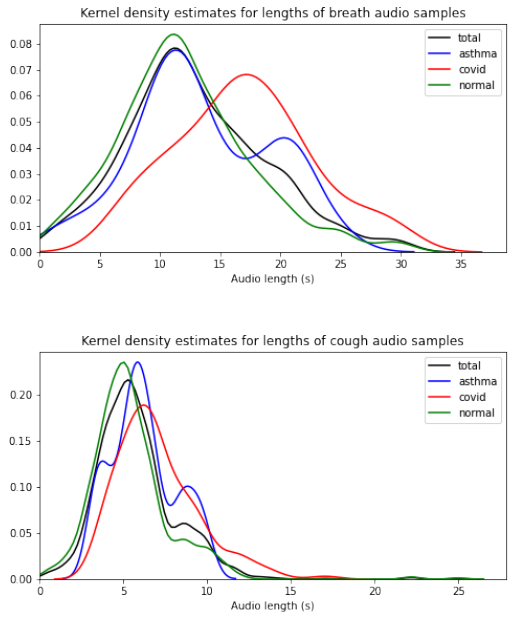


Fig. 1. KDE plots of length of audio samples

- 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).
- RNNs require an input of shape [`num_samples`, `num_timesteps`, `num_features`]. This means that each audio sequence must have the same length.

VI. Feature Extraction

A. Types of Features

Before any discussion of audio processing takes place, it is imperative to know the meaning of sampling rate. 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 44 100 Hz. It just so happens that human hearing has an upper bound at around 22 050 Hz. The Nyquist-Shannon theorem [11] tells us that to avoid aliasing effects and extract reliably a maximum frequency f from the signal, we need to sample it with a minimum frequency of $2f$.
- An n-bit audio clip refers to the number of bits used to store the 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 16 000 Hz (= 2 × 8000 Hz), because on performing EDA, we found that most of the significant frequencies were found up to 8000 Hz. Researchers working on [6] 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.

1) 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 [10] 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:

- $\texttt{frame\_len} = 256$
- $\texttt{pad\_len} = \lfloor \frac{256}{2} \rfloor = 128$
- $\texttt{hop\_len} = 64$

Instantaneous features are perfect for RNNs. 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 13.04 s for the breath samples and 5.82 s for the cough samples. Approximating the mean audio length as 10 s, using sampling rate as 16 000 Hz and the parameters defined in Instantaneous Features, a simple back-of-the-envelope calculation using Equation 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.

2) 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.

3) 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.

An alternative classification of features is:

- Time-domain: Calculated on the raw audio signal $y(t)$
- Frequency-domain: Calculated on the Fourier-transformed audio signal $\mathcal{F}[y(t)] = y(\omega)$.

B. Features Used

We shortlisted the commonly used features in audio processing tasks, extracted them from the data and shortlisted the ones that looked useful:

1) Root Mean Squared Energy (RMSE): It gives an idea of how much energy is contained in the signal [10]. 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 i^{th} sample in the frame.

2) 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:

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

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

3) 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. More specifically, it is the weighted mean of the frequencies in the signal, where the weights are the magnitudes of the frequencies in the Fourier transform. SC for a frame is given by:

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

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 j^{th} frequency f_j in the frame.

4) Spectral Bandwidth (SB): It gives an idea about the spread of the frequencies around a mean value [10], like a measure of dispersion such as variance or standard deviation. We used the 2^{nd} 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:

$$\text{SB2}_{\text{frame}} = \frac{1}{2}\sum_{i=1}^{n} M_i(f_i - \text{SC}_{\text{frame}})^2 \tag{6}$$

5) Spectral Rolloff (SR): It is the frequency below which $p\%$ of energy of the spectrum for a frame lies [6]. p is a threshold, typical values for p are 85 and 95 [12]. We used $p = 85$ for our work.

6) 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 [6]. Typically, the first 13 coefficients are used.

C. 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 Instantaneous Features).

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 that makes this condition true. The Hann windowing is a very common windowing function. Keeping a hop length less than frame length (i.e., 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 [13]:

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

D. Input Vectors to RNNs

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

VII. Experiments and Results

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

A. SVM Model

We used `scikit-learn` [14] to train the SVM.

Procedure:

- Applied principal component analysis (PCA) on the data to reduce the dimensionality of the data, using 20 components (which explained 84% of the variance).
- Passed the output from the PCA into a support vector machine classifier with a radial basis function (RBF) kernel.
- 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.
- 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 III
CONFUSION MATRIX USING SVM MODEL FOR BREATH DATA

		Predicted	
		`normal` (0)	`covid` (1)
True	`normal` (0)	157	2
	`covid` (1)	6	29

1) Audio Type – Breath:

- Precision = 0.94
- Recall = 0.83
- F1 = 0.88
- Accuracy = 0.96

TABLE IV
CONFUSION MATRIX USING SVM MODEL FOR COUGH DATA

		Predicted	
		`normal` (0)	`covid` (1)
True	`normal` (0)	153	4
	`covid` (1)	1	34

2) Audio Type – Cough:

- Precision = 0.89
- Recall = 0.97
- F1 = 0.93
- Accuracy = 0.9740

Note: We had to drop 1 `covid` sample and 1 `normal` sample because they contained NaN values.

B. CNN Model

Though we were able to obtain excellent results using an SVM by directly using the imbalanced dataset, the CNN model was sensitive to the class imbalance.

We tried class weighting in `Keras` [15] using a range of class weights, but that gave poor results. Hence we downsampled the majority class and performed data augmentation.

Current status: Model is overfitting. It is predicting all samples as `normal` and hence giving an accuracy of 82%, which is the fraction of `normal` samples in the dataset. We encountered a similar problem with the LSTM model, which we rectified by simplifying the architecture. We will try the same with the CNN as well.

Procedure:

- 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 is alright, because we can shift the audio forward and backward without affecting the class of the sample or meaning of the spectrogram. This is because shifting along the time axis still allows spectrograms to be comparable. 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 the spectrogram is not a meaningful transformation.
- Training the model using the architecture detailed below with early stopping, with the Adam optimizer.

TABLE V
CNN ARCHITECTURE

Layer	**Activation**	**Layer Information**
Convolution 2D	relu	$f_c = 32, f_s = (3, 3)$
Convolution 2D	relu	$f_c = 32, f_s = (3, 3)$
Max pooling 2D	None	$f_s = (2, 2)$
Dropout	None	$p = 0.25$
Convolution 2D	relu	$f_c = 64, f_s = (3, 3)$
Max pooling 2D	None	$f_s = (2, 2)$
Dropout	None	$p = 0.25$
Convolution 2D	relu	$f_c = 128, f_s = (3, 3)$
Max pooling 2D	None	$f_s = (2, 2)$
Dropout	None	$p = 0.25$
Flatten	None	None
Fully connected	relu	$n = 64$
Dropout	None	$p = 0.5$
Fully connected	sigmoid	None

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

C. 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.

TABLE VI
LSTM ARCHITECTURE FOR BREATH DATA

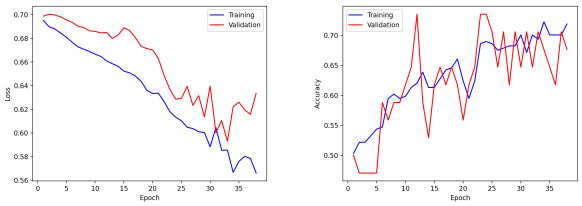
Layer	**Activation**	**Layer Information**
LSTM	tanh	$u = 32$
LSTM	tanh	$u = 32$
Fully connected	relu	$n = 32$
Fully connected	sigmoid	$n = 1$

u = Number of LSTM units, n = Number of neurons

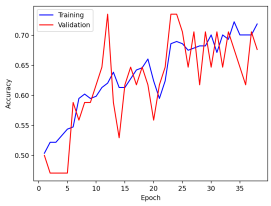
1) Audio Type – Breath: Metrics on the breath data have very noisy curves when plotted against epoch. Currently the LSTM model is performing, extremely poorly, slightly better than random chance (AUC = 0.5694). It is probably due to the small size of the dataset, rendered even smaller because of downsampling.

Test set metrics for the LSTM trained on breath data are below.

- Precision = 0.6000
- Recall = 0.6670
- AUC = 0.5694
- Accuracy = 0.6111



(a) Loss (b) Accuracy



(c) AUC

Fig. 2. Training and validation metrics for breath data

TABLE VII
LSTM ARCHITECTURE FOR COUGH DATA

Layer	Activation	Layer Information
LSTM	tanh	$u = 64$
LSTM	tanh	$u = 64$
Fully connected	relu	$n = 64$
Fully connected	sigmoid	$n = 1$

u = Number of LSTM units, n = Number of neurons

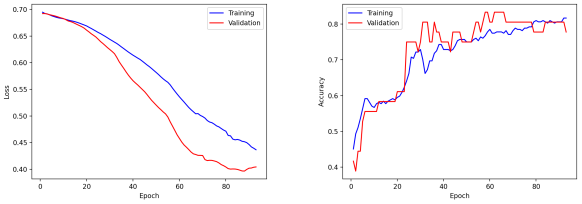
2) Audio Type – Cough: The LSTM model performs significantly better on the cough data than on the breath data.

Test set metrics for the LSTM trained on breath data are below.

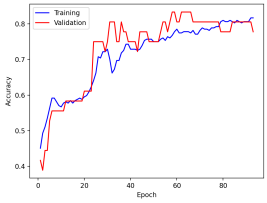
- Precision = 0.7500
- Recall = 0.8333
- AUC = 0.8025
- Accuracy = 0.7778

VIII. Conclusion and Future Work

We are yet to complete implementing the CNN and RNN models, tune the current ML models and explore other traditional ML models. Only then can a fair comparison be done among the three approaches. Ensembling techniques have shown promise in the past, and the three-fold approach is a perfect opportunity to implement ensembling. There is a lot of scope of applying deep learning techniques to similar tasks in the future. However, the primary roadblock at the moment is the lack of sufficient amount of data. Neural networks, especially with several layers, tend to start overfitting very quickly on a small dataset.



(a) Loss (b) Accuracy



(c) AUC

Fig. 3. Training and validation metrics for breath data

IX. Work Update

A. Work Completed

- Data collection
- Data cleaning and preprocessing
- Feature exploration and extraction
- Input generation for the three types of models
  - Structured data for ML models
  - Mel spectrograms for CNNs
  - Sequential input for RNNs
- Training all three models
  - SVM
  - CNN
  - LSTM

B. Work Remaining

- Dealing with imbalanced data for CNN and LSTM
- Simplifying CNN architecture and retraining
- Retraining and improving metrics on LSTM, especially for breath data

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[5] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555
[6] C. Brown, J. Chauhan, A. Grammenos, J. Han, A. Hasthanasombat, D. Spathis, T. Xia, P. Cicuta, and C. Mascolo, "Exploring automatic diagnosis of covid-19 from crowdsourced respiratory sound data," 2020.
[7] N. Sharma, P. Krishnan, R. Kumar, S. Ramoji, S. R. Chetupalli, N. R., P. K. Ghosh, and S. Ganapathy, "Coswara – a database of breathing, cough, and voice sounds for covid-19 diagnosis," 2020.
[8] R. Dunne, T. Morris, and S. Harper, "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," 2020.
[9] G. Pinkas, Y. Karny, A. Malachi, G. Barkai, G. Bachar, and V. Aharonson, "Sars-cov-2 detection from voice," *IEEE Open Journal of Engineering in Medicine and Biology*, 2020.
[10] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, 2015, pp. 18–25.
[11] C. E. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.
[12] G. Peeters, "A large set of audio features for sound description (similarity and classification) in the cuidado project," *CUIDADO IST Project Report*, vol. 54, no. 0, pp. 1–25, 2004.
[13] D. O'Shaughnessy, *Speech Communication: Human and Machine*, ser. Addison-Wesley series in electrical engineering. Addison-Wesley Publishing Company, 1987. [Online]. Available: https://books.google.co.in/books?id=mHFQAAAAMAAJ
[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html
[15] F. Chollet *et al.*, "Keras," https://keras.io, 2015.