Using Supervised Contrastive Learning to Improve COVID-19 Cough Classification

Melissa Weyant, Pura Peetathawatchai; TA: Elaine Sui

December 2022

1 Background

The COVID-19 pandemic is one of the biggest recent challenges our world has faced. To combat it's ranging impacts and reduce its global spread, access to rapid testing is crucial. However, access to at-home testing is very limited, particularly in developing countries, and even in developed countries testing kits are costly (1). A test based on audio classification of coughs would be very useful as it could be cheap and made broadly accessible with relative ease.

Audio classification using deep learning has improved markedly in recent years. A wide range of research has been done to develop new and refine existing architectures for the wide-ranging applications of audio classification (2). Commonly used deep learning audio classification techniques include CNNs, RNNs, and transformers. For example, convolutional neural networks have been used in word recognition and musical genre classification (3). Additionally, transformers have been used for sound labelling (4).

Several groups have leveraged these techniques to determine if someone has COVID-19 from audio samples of coughs (5,8). These groups have employed techniques including transformers, CNNs using a modified ResNet architecture, and Support Vector Machines (5, 8). Often, the model architecture can be broken into two main components: 1) an encoder that takes in the audio input as a spectrogram and outputs an embedding vector of the audio and 2) a classifier that takes in the embedding vector and outputs a binary classification (COVID-19 or no COVID-19).

There are a few primary pitfalls present in the task of COVID-19 cough classification. First, since the samples are typically crowd-sourced, they contain a range of background noises (talking, music, TV, etc.) that can significantly impact performance of machine learning models (9,12). Second, the audio samples typically span varying lengths, such as 0.2-10 seconds, depending on the dataset used (12). Third, samples contain varying numbers of coughs, which makes extraction challenging (12). Finally, the datasets are often very imbalanced between positive and negative samples (10).

One approach that has been shown to improve deep learning model performance and may be helpful in the domain of COVID-19 cough classification is contrastive learning. This type of learning allows the algorithm to learn similar embeddings for examples of the same class and different embeddings for examples of different classes (11, 6). This type of learning has been used in a wide range of applications in recent years. For example, it has proven helpful in both the fields of natural language processing and image classification (6). As this approach has not yet been used in past COVID-19 cough classification models, we sought to determine if this approach could help improve the performance of such models. Specifically, we evaluated several approaches to using contrastive loss in addition to a baseline of no contrastive loss as described in the methods section. In our study, we investigated how this type of learning can be applied to potentially improve the performance of deep learning models for COVID-19 cough classification.

2 Dataset

We used COUGHVID's as our dataset. It contains over 25,000 crowdsourced cough recordings; however, it contains only 1,155 COVID-positive samples. The dataset was crowdsourced and includes samples from diverse groups (7). The dataset contains a variety of ogg and webm recordings, which we converted into wav file format. See the waveform graphs, spectrograms, and statistics for one of our positive samples and one of our negative samples in the appendix.

Our dataset has several limitations. First, since it is crowdsourced, there is varying quality of the recordings and some do not contain coughs. For example, we had to remove many empty sound recordings with no audio. Second, background noise in the files can interfere with proper classification. Third, dataset imbalances may cause the model to output more negative predictions due to the overwhelming number of negative samples.

3 Baseline

Our baseline is described in detail in the methods section. In short, it involved developing and training an encoder (CNN) and classifier (MLP) that are trained end-to-end using the traditional cross-entropy loss. We also conducted a hyperparameter search as described in the methods section. We selected the best model based on performance on the validation set according to the F1 score. This model had an F1 score of 0.002 on the training set, 0 on the validation set, and 0 on the test set because it outputted only negative predictions.

4 Methods

Since our dataset contained 25,000 samples, we performed an approximate 80% train, 10% validation, and 10% test split. We first performed preprocessing on our wav audio files. We removed all samples whose raw audio vectors summed to 0, corresponding to empty sound recordings. Initially, we loaded them as 1D raw audio vectors. The shortest and longest audio vectors had lengths of 23,040 and 5,342,400, respectively. Due to the need to pass inputs of fixed dimensions into the downstream architecture and the memory constraints posed by the larger sound files, we trimmed the audio files such that the lengths of all audio vectors were 20,000. This window size allows for one to two coughs. We used a 20,000-length sliding window and selected the window that corresponds to the maximum amplitude. From listening to select samples, this successfully trimmed around the cough in all cases. We then converted the wav files into spectrograms using the torchaudio library, which were then fed as 2D tensors as inputs to the deep learning model.

Our deep learning model consisted of a convolutional neural network architecture for the encoder and a multi-layer perceptron (MLP) for the classifier. We based our initial encoder architecture on Virufy's paper (see appendix 4). As we used a smaller dataset than Virufy and had smaller spectrogram size, we altered the architecture of our initial model before conducting any of the analyses described below. Our final encoder specifications are given by Figure 2. Our classifier is a simple 2-layer perceptron where the hidden layer size is determined by hyperparameter tuning.

A major component of our training strategy is the use of self-supervised contrastive learning. At a high level, we train the encoder such that it produces embedding vectors such that embeddings for coughs with the same label (both positive or both negative) are similar, while those for coughs of different labels are different. The formula for self-supervised contrastive learning is as follows, and is given by Khosla et al's 2020 paper "Supervised Contrastive Learning" (3):

$$\mathcal{L}_{contr}(I) = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P} \log \frac{\exp(z_i \cdot z_p / \tau)}{\sum_{a \in A(i)} \exp(z_i \cdot z_a / \tau)}$$

where I represents the batch, and for any given $i \in I$, $P(i)$ represents the subset of I with the same label as i, excluding i itself, and $A(i)$ is essentially I excluding i. We see here that we are using the dot-product as a measure of similarity. τ is a scaling hyperparameter that needs to be tuned. z represents the embeddings output by the encoder.

We explored three different training strategies:
1. Train the model end-to-end using the standard cross-entropy loss (baseline)
2. Train the encoder using the contrastive loss given above and then freeze the encoder layer and train the full model again using standard cross-entropy loss.

3. Train the model end-to-end using the weighted sum $\mathcal{L}_{contr} + \lambda \cdot \mathcal{L}_{CE}$ where λ is a hyperparameter to be tuned.

Before training our models, we performed hyperparameter tuning using our automated custom tuning function. This function performs a series of random trials: during one trial, each hyperparameter is randomly sampled from the same ranges listed in the appendix. We manually picked reasonable ranges for each hyperparameter and then ran the hyperparameter tuning code to select our optimal values; it outputs 30 random configurations per run. We were able to observe configurations that led to poor performance (for example, too low of a learning rate always led to poor performance), so we were sometimes able to strategically narrow the search window before re-running the code to check for more optimal configurations. We tuned the learning rate, tau (a contrastive loss hyperparameter), lambda (a scaling constant for cross-entropy loss in the weighted sum), the gradient clipping value, the size of the hidden layer in the MLP, the dropout probability, the positive class weight (used in the cross-entropy loss function), batch size, and number of epochs. We selected the hyperparameter trial with the highest validation F1 score as the optimal hyperparameter set.

5 Analysis

Unbalanced Dataset
Upon training our baseline model, we noticed that it consistently only outputted negative predictions. We realized that this was likely because our dataset is imbalanced with only 1,155 COVID-positive recordings in a dataset of 25,000. To combat this issue, we experimented with class weighting in the loss function. We added the class weight parameter to our hyperparameter tuning code to determine the optimal positive class weight for our model while fixing the negative class weight at one.

As separate approach, we also tried oversampling of positives, undersampling of negatives, and a combination of both in order to achieve an approximately balanced dataset.

These approaches significantly improved the F1 training score to 0.9-0.99. However, we found that the model was overfitting because while the F1 scores for the training set were 0.9-0.99, the F1 scores for the validation data were 0.10-0.15.

Overfitting
To combat the overfitting, we tried implementing the following approaches:
1) Weight decay. In our hyperparameter tuning code, we considered values from $10e^{-5}$ to $10e^{-3}$.
2) Dropout on all layers except for the output layer based on past literature (5). In our hyperparameter tuning code, we considered dropout probabilities between 0.15 and 0.55.
3) Elimination of undersampling of negatives, in order to increase the size of the dataset for negative samples.

We tried all possible combinations of the three strategies above and found that implementing options 2 and 3 together were optimal. This improved our validation F1 score to 0.35 while our training F1 score decreased, as expected, to 0.6. Examining the results from hyperparameter tuning, we found that further increasing the strength of regularization using the methods above yielded worse performance on both the validation and training datasets.

Batch Normalization
We added batch normalization after each convolutional layer. This improved performance.

Increased Dataset Size We had access to fewer samples than previous groups, such as Virufy, so we tried techniques to acquire more data. This might also help us combat overfitting. We first tried to implicitly increase the size of our dataset by using larger audio sample windows. There were typically 1-4 coughs in the full version of each recording, so we wanted to see if extending the sample window would improve model performance. Since our original window was sufficient to capture one cough (or at most two), we tried windows sizes that were twice and then three times as large as our original window, respectively. However, performance decreased with both extended window lengths, perhaps due to the introduction of more background noise. As not all cough recordings have multiple coughs, other background noise such as talking, TV sounds, and music may have interfered with our model performance. We discuss steps to reduce the effects of this background noise in the section on preprocessing.

Data Augmentation

We then tried explicitly increasing the size of our dataset by implementing two types of data augmentation. First, we used the AddGaussianNoise function from the audiomentations library to generate one additional training example for each positive sample. This resulted in an increase F1 training score of 0.2 but a small decrease of 0.03 in the F1 validation score. It is possible that the added noise obscured important aspects of the coughs. We also tried audiomentations Compose and Shift functions to shift the location of the cough within the sample window. This type of augmentation was only possible on our larger audio window sizes and since our performance was worse on larger audio window sizes, we decided this type of augmentation was not suitable for our final model.

Preprocessing

To improve the quality of our audio samples, we tried using a relatively simple background noise reduction masking function from the literature. a Data Science Mel Spectogram Article (Citation only). This function helps mask out sounds that are below a certain threshold. However, using this function reduced our performance, so we think it is not well suited to cough recordings. In the future, we would like to try other background noise removal methods, such as speech-brain's specformer.

Finally, we experimented with different methods to normalize and generate spectrograms. In terms of normalization methods, we tried three approaches:
- Normalizing the spectrograms to have mean of 0 and standard deviation of 1.
- Normalizing with PCEN from librosa
- Applying batch normalization before the first convolutional layer, which can have a normalizing effect on the spectrograms.

We compared performance across these methods and found that method 1 was best according to the F1 validation score.

In terms of spectrogram generation, we tried 2 approaches: librosa's melspectrogram and librosa's mfcc spectrogram. However, these spectrograms dimensions were not appropriately sized for our encoder, and while we resized them to fit, it may have hurt performance.

Model Training

We next evaluated our baseline (end-to-end training computed using only cross-entropy loss) versus two learning procedures: (1) the end-to-end training, computed using a weighted sum of cross entropy loss and contrastive loss, and (2) sequential training as described above.

We first trained our end-to-end model using contrastive loss. The F1 scores were 0.55 for the training set, 0.37 for the validation set and 0.37 for the test set. This corresponds to an overall training set accuracy of 69.40%, a positive accuracy of 92.31%, and a negative accuracy of 63.66%. We had a 54.88% overall test set accuracy, a 64.15% positive accuracy, and a 52.46% negative accuracy. The standard deviation in the F1 scores, computed over 5 runs, was 0.017 for the validation set and 0.016 for the test set.

We then trained our sequential model. We first trained the encoder using only contrastive loss. We picked the hyperparameters associated with the lowest loss that had similar F1 training and validation losses (adjusted for the difference in their dataset size) and made sure that the loss consistently decreased during the encoder training. We then outputted embeddings from the encoder (when run on the optimal hyperparmeters) through our MLP classification layers. However, even after hundreds of hyper-parameter tuning cycles for the optimal MLP hyperparameters, they always resulted in an F1 validation and training scores of close to 0 such as 0.005. This sequential method of training does not seem to be well-suited to cough classification.

Overall, the end-to-end training strategy outperformed the baseline and the sequential training method. See appendix 6 for the accuracy of this model and appendix 5 for a results comparison table. There are a few areas we would explore in future work to improve the audio classification accuracies. We believe that further experimenting with preprocessing of the audio samples to remove background noise may improve performance. We also think that increasing our dataset size and exploring other forms of data augmentation may increase performance. We have relatively little data, particularly of positive samples, and groups such as Virufy used larger datasets. Finally, if we obtained a larger dataset we would like to try a larger encoder network, such as our original design from Virufy's paper (5).

6 Implementation

This model was developed using Python 2.7.16 . It is available at GitHub at:
https://github.com/melissaelizabeth9/CS230

7 Combining Models with Patient Data

In order to make our project more real-world applicable, we then combined the probabilities calculated by our best performing training method with additional data about each cough sample in our dataset. This data contains statistics about the patient and basic symptoms. We used 14 features: probability associated with a negative classification, probability associated with a positive classification, one hot vector for female gender, one hot vector for male gender, one hot vector for if the person has a respiratory condition, one hot vector for if the person has a cough or fever, and 8 one hot vectors corresponding to each decade of age (1 through 8). We used an MLP with three layers for classification. It initially had an F1 training score of 0.66 and F1 validation score of 0.30. We then performed random hyperparameter tuning on the learning rate, hidden layer sizes, clip normalization, positive class weight, and number of epochs. The tuned model returned F1 scores of .77 for the training set and .43 for the test set. See Appendix 5 for the result comparison table.

8 Novelty

As described in our analysis section, we incorporated several novel aspects into our project. First, we used self-supervised contrastive learning, which has not yet been used for COVID-19 cough audio classification. Second, we evaluated two training strategies (end-to-end vs sequential) that have not yet been used for COVID-19 audio classification. Third, we evaluated different types of spectrograms and spectrogram normalization methods. While groups have used different approaches, to our knowledge, there hasn't yet been a detailed comparison of these methods.

9 Theory and Problem Solving

As described in our analysis section, we had a few key pivots during our project. For example, when we realized that our validation F1 score was not improving despite increasing regularization strength, we switched to other methods for improving model performance. For example, we tried different types of spectrograms and spectrogram normalization methods. We also evaluated other strategies such as data augmentation (adding Gaussian random noise and sample shifting).

10 Conclusion

We developed and trained a deep learning model for classification of COVID-19 versus normal coughs. We found we were able to improve model performance by using contrastive loss with the end-to-end approach. We hope that subsequent research can further develop these methods and help address a pressing need for COIVD-19 testing.

11 Contributions

Melissa mainly worked on creating the deep learning model in PyTorch. She also worked on researching and implementing methods to improve model performance and data pre-processing.

Pura mainly worked on the high-level design for the project's methodology and architecture. He was also responsible for setting up and maintaining the AWS instance, and made contributions to the code for data preprocessing and hyperparameter tuning.

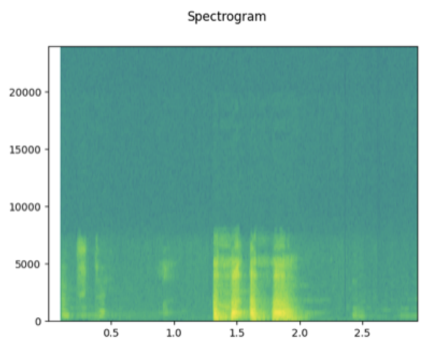
12 Appendix

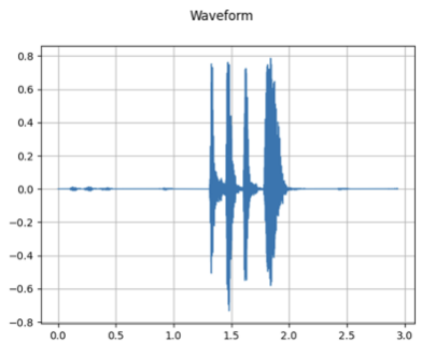
Appendix 1: Dataset Visualizations

Statistics and graphs for one of our negative audio samples:

```
Sample Rate: 48000
Shape: (1, 141120)
Dtype: torch.float32
 - Max:       0.787
 - Min:      -0.733
 - Mean:     -0.000
 - Std Dev:   0.075

tensor([[ 0.0000e+00,  0.0000e+00,  0.0000e+00,  ..., -6.1035e-05,
         -3.0518e-05, -3.0518e-05]])
```

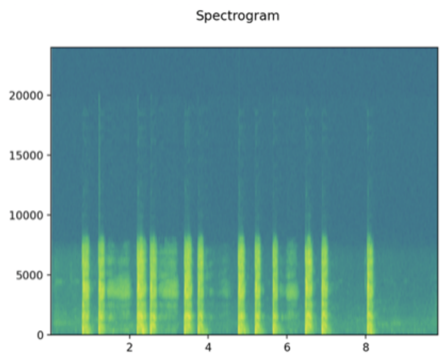


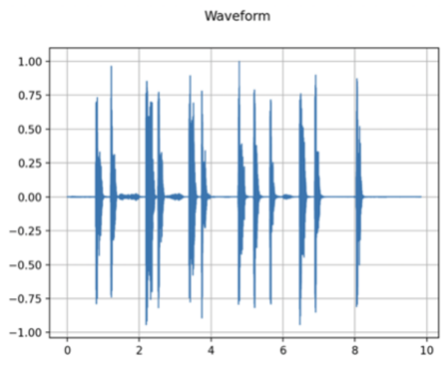


Statistics and graphs for one of our positive audio samples:

```
Sample Rate: 48000
Shape: (1, 472320)
Dtype: torch.float32
 - Max:       1.000
 - Min:      -0.943
 - Mean:     -0.000
 - Std Dev:   0.085

tensor([[0.0000e+00, 0.0000e+00, 0.0000e+00,  ..., 3.0518e-05, 3.0518e-05,
         3.0518e-05]])
```





Appendix 2: Typical hyperparameter tuning sampling ranges

$"lr" : 10 * *np.random.uniform(-5, -3),$
$"tau" : 10 * *np.random.uniform(-1, 2),$
$"hiddensize" : np.random.randint(32, 512),$
$"lambda" : 10 * *np.random.uniform(2, 3.5),$
$"clipnorm" : 0.0001 * np.random.uniform(1, 70),$
$"dropoutprob" : np.random.uniform(.2, .55),$
$"classWt" : np.random.randint(1, 100)$
$"numEpochs" : np.random.randint(15, 60),$
$"batchSz" : np.random.randint(32, 100)$

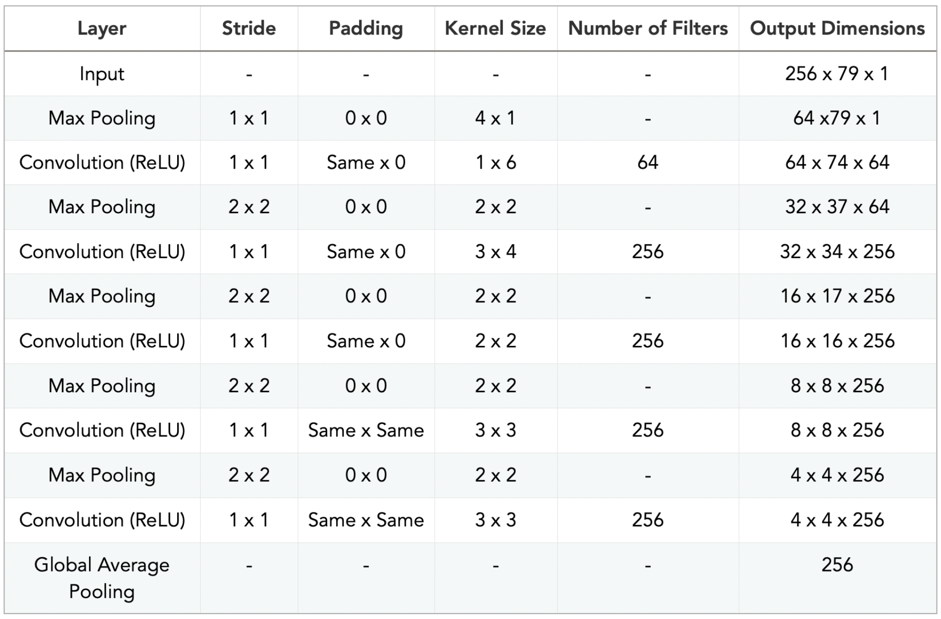
Appendix 3: Optimal Hyperparmeters

Here is an example of our optimal hyperparmeters (from end-to-end training):

'lr':	0.000604175144962678
'tau':	29.429581209580668
'hiddensize':	394
'lambda':	547.7817438774465
'clipnorm':	0.001987183639647019
'weightdecay':	0
'dropoutprob':	0.2
'classWt':	5
'numEpochs':	50

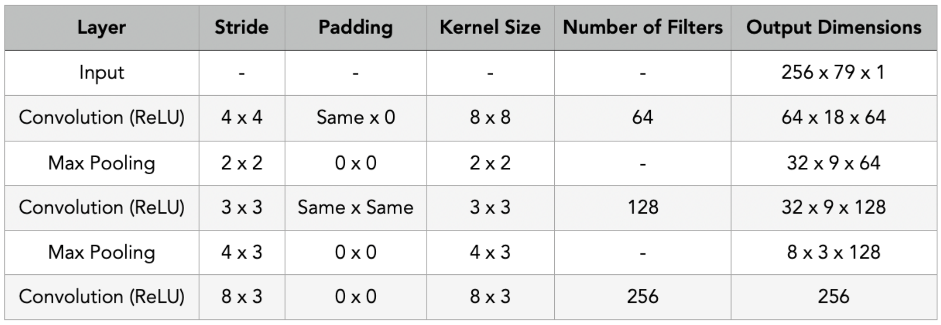
Appendix 4: Encoder Architecture

Figure 1: Initial architecture



Layer	Stride	Padding	Kernel Size	Number of Filters	Output Dimensions
Input	-	-	-	-	256 x 79 x 1
Max Pooling	1 x 1	0 x 0	4 x 1	-	64 x79 x 1
Convolution (ReLU)	1 x 1	Same x 0	1 x 6	64	64 x 74 x 64
Max Pooling	2 x 2	0 x 0	2 x 2	-	32 x 37 x 64
Convolution (ReLU)	1 x 1	Same x 0	3 x 4	256	32 x 34 x 256
Max Pooling	2 x 2	0 x 0	2 x 2	-	16 x 17 x 256
Convolution (ReLU)	1 x 1	Same x 0	2 x 2	256	16 x 16 x 256
Max Pooling	2 x 2	0 x 0	2 x 2	-	8 x 8 x 256
Convolution (ReLU)	1 x 1	Same x Same	3 x 3	256	8 x 8 x 256
Max Pooling	2 x 2	0 x 0	2 x 2	-	4 x 4 x 256
Convolution (ReLU)	1 x 1	Same x Same	3 x 3	256	4 x 4 x 256
Global Average Pooling	-	-	-	-	256

Figure 2: Final architecture



Layer	Stride	Padding	Kernel Size	Number of Filters	Output Dimensions
Input	-	-	-	-	256 x 79 x 1
Convolution (ReLU)	4 x 4	Same x 0	8 x 8	64	64 x 18 x 64
Max Pooling	2 x 2	0 x 0	2 x 2	-	32 x 9 x 64
Convolution (ReLU)	3 x 3	Same x Same	3 x 3	128	32 x 9 x 128
Max Pooling	4 x 3	0 x 0	4 x 3	-	8 x 3 x 128
Convolution (ReLU)	8 x 3	0 x 0	8 x 3	256	256

Appendix 5: F1 Results

	Baseline	Sequential	End-to-End	Combined
Training F1 Score	0.002	.10	.55	.77
Test F1 Score	0.0	.05	.37	.43

Appendix 6: Accuracy Results For End-To-End-Training

	Training Set	Validation Set	Test Set
Overall Accuracy	69.4%	54.88%	54.88%
Positive Accuracy	92.3%	64.15%	64.15%
Negative Accuracy	63.66%	52.46%	52.46%

13 References

(1) Berger, Miriam. 2021. “Coronavirus tests are needed for international travel. But they can cost more than a flight.” The Washington Post. https://www.washingtonpost.com/world/2021/12/09/coronavirus-tests-are-needed-international-travel-they-can-cost-more-than-flight/.
(2) https://paperswithcode.com/task/audio-classification
(3) Gong, Yuan, et al. “AST: Audio Spectrogram Transformer.” Interspeech 2021, Aug. 2021. Crossref, https://doi.org/10.21437/interspeech.2021-698.
(3) A J E Kell, D L K Yamins, E N Shook, S V Norman-Haignere and J H McDermott ”A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy” Neuron, vol.98 pp. 630-644, May 2018
(ICASSP), pp. 21-25. IEEE, 2021. Accessed at https://arxiv.org/pdf/2010.13154.pdf
(4) Gong, Yuan, et al. “AST: Audio Spectrogram Transformer.” Interspeech 2021, Aug. 2021. Crossref, https://doi.org/10.21437/interspeech.2021-698.
(5) Haritaoglu, Esin Darici, Nicholas Rasmussen, Daniel CH Tan, Jaclyn Xiao, Gunvant Chaudhari, Akanksha Rajput, Praveen Govindan et al. ”Using Deep Learning with Large Aggregated Datasets for COVID-19 Classification from Cough.” arXiv preprint arXiv:2201.01669 (2022).
(6) Khosla, Prannay, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. ”Supervised contrastive learning.” Advances in Neural Information Processing Systems 33 (2020): 18661-18673. Accessed at https://arxiv.org/pdf/2004.11362v5.pdf
(7) Orlandic, Lara et al. “The COUGHVID crowdsourcing dataset, a corpus for the study of large-scale cough analysis algorithms.” Scientific data vol. 8,1 156. 23 Jun. 2021, doi:10.1038/s41597-021-00937-4
(8) Pahar, Madhurananda, Marisa Klopper, Robin Warren, and Thomas Niesler. “Covid-19 Cough Classification Using Machine Learning and Global Smartphone Recordings.” Computers in Biology and Medicine 135 (2021): 104572. https://doi.org/10.1016/j.compbiomed.2021.104572.
(9) Gupta, Shivani, and Atul Gupta. “Dealing with Noise Problem in Machine Learning Data-SETS: A Systematic Review.” Procedia Computer
Science 161 (2019): 466–74. https://doi.org/10.1016/j.procs.2019.11.146.
(10) Pahar, Madhurananda, Marisa Klopper, Robin Warren, and Thomas Niesler. “Covid-19 Cough Classification Using Machine Learning and Global Smartphone Recordings.” Computers in Biology and Medicine 135 (2021): 104572. https://doi.org/10.1016/j.compbiomed.2021.104572.
(11) Tiu, Ekin. “Understanding Contrastive Learning.” Medium. Towards Data Science, January 8, 2021. https://towardsdatascience.com/understanding-contrastive-learning-d5b19fd96607.
(12) Ashby, Alice, Julia Meister, Khuong Nguyen, Zhiyuan Luo, and Werner Gentzke. “Cough-Based COVID-19 Detection with Audio Quality Clustering and Confidence Measure Based Learning.” Proceedings of Machine Learning Research, 2022.