CS230

Screening COVID-19 by Cough Audio Data Using Attention-Based Convolutional Recurrent Neural Networks

Chris Gu
Department of Computer Science
Stanford University
cgu26@stanford.edu

Nantanick Tantivasadakarn
Department of Computer Science
Stanford University
nantanic@stanford.edu

Serena Zhang
Department of Computer Science
Stanford University
serena2z@stanford.edu

Abstract

The importance of accurately classifying coughs lies in the potential for automatic detection of COVID-19, which can aid in disease control and mitigation efforts. Previous literature has primarily focused on converting cough audio data into spectrograms used to train convolutional neural networks (CNNs). In this study, we examine two CNNs, a convolutional recurrent neural network (CRNN), and a CRNN with novel multi-head self-attention. The results show that the CRNN and CRNN with attention models achieved the highest area under the curve (AUC) values, with AUC scores of 0.88 for both. This suggests that models that can analyze the temporal aspect of inputs may be best suited for this classification task. However, the addition of attention did not seem to significantly improve performance.

1 Introduction

Since the start of the COVID-19 pandemic, many diagnostic tools have been developed to provide quick screening for COVID-19. The two standard COVID-19 tests, the PCR test and the rapid antigen test, while efficient and accurate, can be economically taxing or inaccessible. Our project aims to solve this problem by screening for COVID-19 through audio-recorded coughs. This method allows for a free, efficient, and effective alternative to traditional diagnostic methods.

The input to our algorithm is an audio clip of a cough, which we turn into a Mel spectrogram (frequency by time image) using Fourier Transforms. We then run the data through 3 models of our own design: a Convolutional Neural Network (CNN), a Convolutional Recurrent Neural Network (CRNN), and a CRNN with a novel attention mechanism, to output a prediction of COVID-19 or healthy.

CS230: Deep Learning, Fall 2022, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

2 Related work

There has been a recent interest in using audio data to diagnose COVID-19. Coughs, breathing patterns, and speech have been studied in detecting the disease (1). In the scope of this study, we will mainly focus on cough sounds.

There are 4 main datasets found in the literature we have reviewed: COUGHVID (2), Virufy (3), Cambridge (4), and Coswara (5). Previous works first did feature selection of the audio data, usually in a Mel-scale spectrogram (3; 6). Other methods such as Tonal Centroid, Chromagram, and Spectral Contrast were also used in (6). They then applied machine learning models to classify the cough sounds. Chaudhari et. al. (3) created an ensemble model of Convolutional Neural Networks (CNN) and fully connected models with varying features. Chowdhury et. al. (6) took a similar approach but also included non-deep learning methods such as XGBoost and random forest in their model. This work used the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) method to rank and combine the results of each model, which resulted in a better result (AUC = 0.92, over AUC = 0.72) on multiple datasets including Cambridge and Coswara. Esposito et. al. (7) did an ensemble of Convolutional Recurrent Neural Networks (CRNNs) adapted from the basic CNN architecture of Visual Geometry Group 13 (VGG13) (8). They modified the VGG13 architecture by eliminating global average pooling in their convolutional layers, which preserves temporal information. Their ensemble of a CNN, CRNN, Gated-CNN, and Gated-CRNN achieved an accuracy of 65.28% on COUGHVID. No other metrics or datasets were reported. Chaudhari et. al.(3) briefly mentions the consideration of Convolutional Recurrent Neural Network (CRNN) to analyze cough audio data, but they did not end up using it in their research, nor did they describe any details of a CRNN. Hamdi et al. (9) were the only ones who implemented a hybrid CNN-LSTM and an attention-based hybrid CNN-LSTM, the latter of which achieved an accuracy of 89.3%.

As there is no clear benchmark in the literature, it is difficult to compare results across studies). Multiple works use a different combination of multiple datasets or different metrics. For example, Chaudhari et al. (3), combined results of COUGHVID and Coswara (AUC = 0.771). Esposito et al. (7) combined COUGHVID, Coswara, and Cambridge (Accuracy 65.28%). Chetupalli et al. (10) combined results from COUGHVID as well as speech and breathing samples (AUC = 92.40). Thus, we were unable to establish which of these studies is the "state-of-the-art" nor determine which architecture is best suited for this task. Nevertheless, CNNs are the most prevalent deep-learning architectures, so we will be using them as our baseline.

For our model, we will be taking inspiration from (9), which uses a combination of CNN-LSTM and attention mechanism to predict coughs, as well as the success of multi-head self-attention in Transformer models for Natural Language Processing (11).

3 Dataset and Features

We used the COUGHVID dataset (2), a crowd-sourced collection of 25,000 cough recordings. The dataset contains metadata on the age, gender, and status (healthy, COVID-19, or symptomatic) of the subject. Each cough recording also has a "cough-detected" probability derived from the original Nature paper where the authors trained a XGB classifier to perform cough classification.

Of the 25,000+ cough recordings, we filtered for samples that were labeled (self-labeled or by experienced physicians) and had a cough-detected probability > 0.8. This gave us a total of 13,535 samples: 720 COVID, 10,132 healthy, and 2,683 symptomatic. Of those, we removed 788 corrupted audio files (475 healthy, 20 COVID-19, and 293 symptomatic).

In our early experiments, we used all data points and all three classes to train our models but got poor results. We suspected the symptomatic data may have been ambiguous and interfering with the results, so we tried excluding them in our subsequent experiments. We also suspected that the healthy to COVID-19 dataset imbalance was interfering with training our model, as the model was classifying everything as healthy. In addition to methods such as weighting the loss of each class proportional to its prevalence, we tried creating multiple datasets to test our hypothesis. In the end, we settled on using a balanced dataset with 3500 COVID-19 samples and 3500 healthy samples, where we over-sampled the COVID-19 samples. (We arbitrarily decided that we don't want covid samples to be augmented more than 5 times). The train, validation, and test distribution for our dataset are shown in the diagram above as Figure 1.

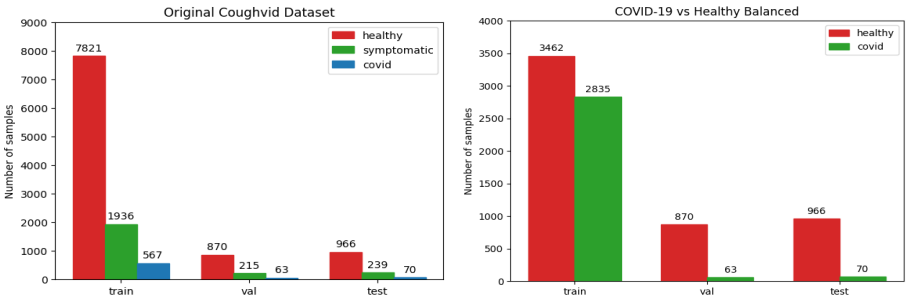

Figure 1: Graphs of Dataset Distributions

The audio files were originally given as .webm/.ogg files, for which we converted all to .wav files. For pre-processing, we re-sampled the audio to a sample rate of 16,000 Hz, collapsed the channel to mono, padded shorter audio clips, and cut longer audio clips to be of the same length.

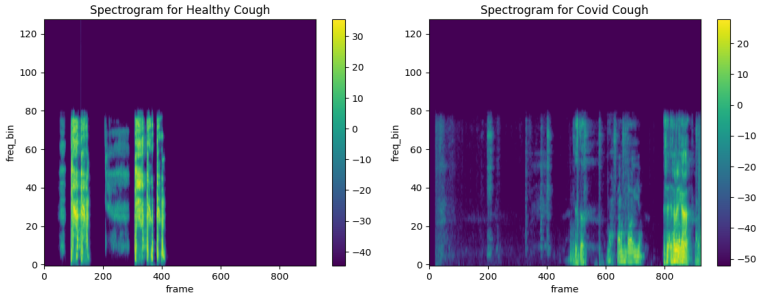

Figure 2: Mel Spectrogram Input

We augmented the audio data using pitch shift, colored background noise, time inversion, and polarity inversion. Then, we converted the audio to a Mel spectrogram using fast Fourier transforms extracting frequency and amplitude with respect to time. The hyperparameters we used for the Fourier transform were n_fft = 1024 (length of the FFT), hop_length = 512 (distance between each successive FFT window), and n_mels = 128 (shape of output). Figure 2 displays a couple of examples.

4 Baseline

For our baseline, we will be comparing our models to the reported scores in (9; 3). To account for differences in data processing, we also created two CNN models for comparison. The first is a "Small CNN" with 4,681 trainable parameters, which we built according to (12) (Accuracy = 95.18). The second CNN model used a similar CNN architecture in (9), which we will call "Large CNN", (with 161,122 parameters), in order to save time on adding LSTM and attention sections in later sections. We $lr = 1e-4, 1e-5$, weight decay, and 0.2 dropout for hyperparameters. There are four layers in the CNN, with each layer encompassing a convolution kernel, a Rectified Linear Unit (ReLU) activation, and a max-pooling kernel, followed by batch normalization. The overview architectures of both models are shown in figure 3, and their results are in Table 1. A detailed visual describing each model's convolutional layers is referenced in appendix section C as Figure 8 and Figure 9, and tables representing the models' full architecture with dimensional information are also referenced in the same section as Figure 10 and Figure 11.

5 Methods

We propose a novel combination (CRNN + Attention) of convolutional layers, Long Short-Term Memory (LSTM), and multi-head self-attention. The CNN portion of the model is meant to capture

0.9

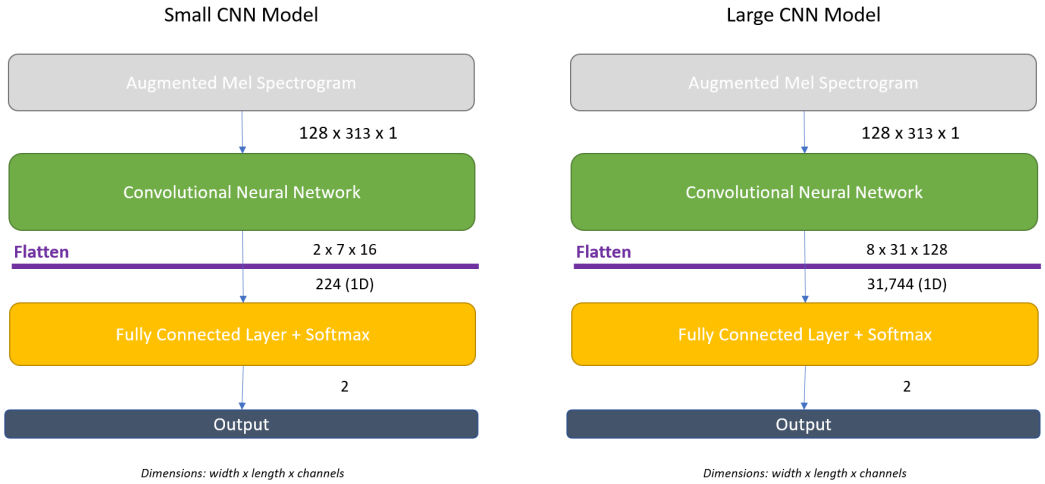

Figure 3: Baseline Architectures (Small and Large CNN Models)

localized representations of the different parts of the spectrogram. The convolutional design of this part was the same as the Large CNN model. Since audio is sequential in nature, the output of the CNN is flattened except for the temporal axis and is then passed to LSTM and attention layers. The LSTM model (equations in Appendix A as displayed in Figure 6), combines the input with a representation of past steps to analyze and make predictions on the next time step. Our LSTM model was bidirectional and its hidden state was 256 dimensions. The multi-head self-attention layer 'attends' to the most relevant parts of the input by using the combination of multiple "heads" to pay attention to different aspects multiplied by a weight (W_O), which is displayed in equation 2. In our case, we used four heads. Each head uses a query-key-value system as shown in equation 3 to generate attention. Equation 4 describes that the query (Q), key (K), and value (V) values are generated by trainable weights multiplied with the outputs of the LSTM (denoted as X), and d_k is the dimension of the key. With our method, the CRNN + attention model captures the spatial elements of our visual data along with the sequential nature of audio.

To see if the addition of the multi-head attention layer contributed to the performance of the model, we also trained a CNN followed by an LSTM without attention (CRNN) using the same hyperparameters described above. An overview of the architectures can be found in Figure 4. We used the cross entropy loss (equation 1) to train our models[1]. Detailed tables representing the CRNN and CRNN + Attention architectures with dimensional information are referenced in the appendix section C as Figure 12 and Figure 13.

$$\mathbf{L} = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{1}$$

$$MultiHead(Q, K, V) = concat(head_1, head_2, ..., head_n)W_O \tag{2}$$

$$head_i = Attention(Q_i, K_i, V_i) = softmax(\frac{Q_i K_i^T}{\sqrt{d_k}})V_i \tag{3}$$

$$Q_i = W_i^Q X, \qquad K_i = W_i^K X, \qquad V_i = W_i^V X \tag{4}$$

[1]We intentionally used a softmax layer and a multi-class cross-entropy loss instead of their binary equivalents due to our early experiments that had 3 classes

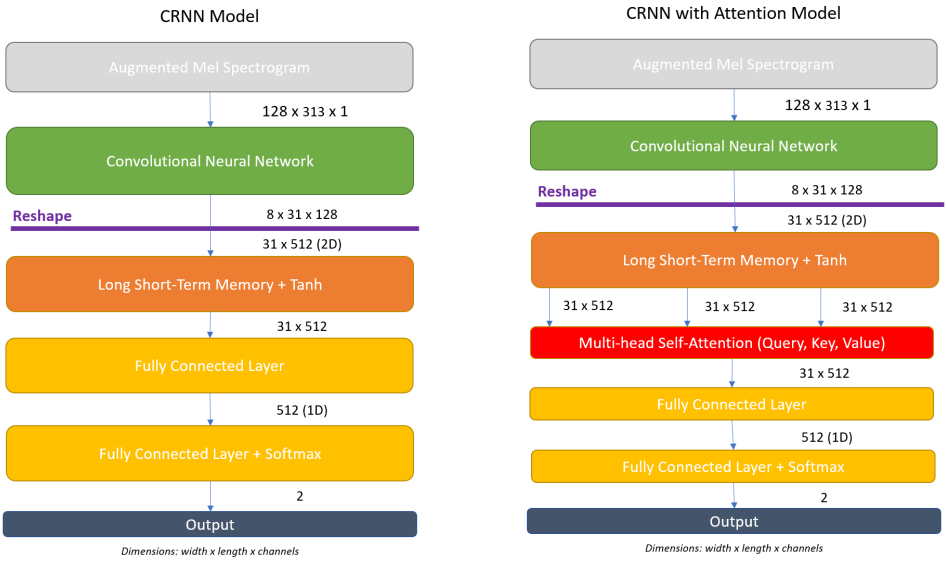

Figure 4: Our CRNN and CRNN + Attention Model Architectures

We implemented our models in Pytorch & Torch Audio (13). To do audio augmentations, we used Torch-audiomentations which was implemented on top of Torch Audio [2]. For hyperparameter tuning, we experimented with learning rate, weight decay, and dropout, which are further detailed in the Experimentation section. We also tuned the epochs (20, 50, 100), batch size (16, 256), loss function (Cross Entropy, NLL loss), and added L2 regularization.

6 Experiments/Results/Discussion

Due to the problem being on classification, we will be using AUC, which is used in most of the literature, as our main evaluation metric. This was chosen to allow using the same code on training 2 or 3 class models, despite accuracy not being a good metric in imbalanced dataset. We at first trained our models using a $lr = 1e-3, \beta_1 = 0.9, \beta_2 = 0.99$, which are the default values of the Adam optimizer in PyTorch. Using the original dataset, which had highly unbalanced labels, all models had the problem of classifying almost all samples as healthy. We initially assumed that this was due to the dataset and attempted to counteract the imbalance with the following:

1. Weighting the loss by the proportion of samples in the dataset by $\frac{(1-p_i)}{\Sigma_j(1-p_j)}$, where p_i is the proportion of samples for each class.
2. Merging the "symptomatic" and "covid" classes in order to simplify the problem
3. Segmenting the cough samples to not contain silent sections.
4. Adding random audio augmentations to artificially increase the size of the training set
5. Oversampling / Undersampling the dataset
6. Changing the hyperparameters for creating the Mel spectrogram (n_fft, hop length, etc.)

In almost all of our experiments and across all splits (train, validation, and test), the results were nearly identical to what is shown in Figure 5, with the only exception being the balanced dataset. For that set of data, we noticed that our models were classifying all classes randomly and that the loss was not changing across epochs. After significant investigation and testing different sets of hyperparameters and methodologies, in the end, we found out that the problem was due to an inappropriately set learning rate.

For the remainder of our experiments, we decided to use the balanced training dataset. For each model, we experimented with learning rates from $1e-2$ and $1e-5$, applying 0.2, 0.1, or no dropout,

[2] https://github.com/asteroid-team/torch-audiomentations

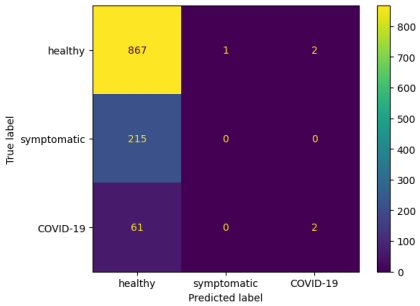

Figure 5: Confusion Matrix that appeared on early experiments

and applying $1e-5$ weight decay. After this, we reported the results from the experiments with the best set of values, as shown in Table 1. We excluded dropout for the CRNN and CRNN + Attention because the loss was not decreasing when included and was terminated early.

Model	Learning rate	weight decay	dropout	epochs	dev AUC	test AUC
CNN (9)	-	-	-	-	-	0.85
CRNN (9)	-	-	-	-	-	0.88
CRNN + Attention (9)	-	-	-	-	-	0.91
Ensemble with CNN (3)	-	-	-	-	-	0.77
Ours						
Small CNN (baseline)	$1e-4$	$1e-5$	0.2	38	0.51	0.53
Large CNN (baseline)	$1e-4$	$1e-5$	0.2	49	0.54	0.53
CRNN	$1e-4$	$1e-5$	-	32	0.88	0.80
CRNN + Self-Attention	$1e-4$	$1e-5$	-	39	0.88	0.80

Table 1: Experiment Results

From our experiments, we concluded that the CRNN and CRNN + Attention were the best-performing models, with their final test AUC scores being very similar. We believe this is due to the fact that the CRNNs are capturing the temporal aspect of the spectrogram: the "frequency" axis of the Mel-spectrogram is collapsed across all channels as the "time" axis is kept constant. The results are passed to the LSTM, which considers the temporal order of the stream of data. We concluded that the CNN's ability to extract spatial and spectral features from Mel-spectrogram images may not be enough for accurate cough classification. This is further corroborated by the fact that we found regularization to not reduce our large CNN's overfitting—Figure 7 shows the loss & accuracy curves and additional analysis.

Both our CNN and CRNN performed worse than the models in (9). Hamdi et al. did spend more time (3 and 8 hours, respectively) training their models, whereas we employed tactics such as early stopping when our loss curves plateaued. We suspect that the differences in the data processing step, the way we augment the data, and other hyperparameters, such as dropout, may explain the difference between the model results.

7 Conclusion/Future Work

In conclusion, the CRNN and CRNN + Attention were our best-performing models; however, they performed equally well, so it seems like the Attention didn't improve the model classification. During this process, we ran into many errors with data processing and model training; thus, we implemented many troubleshooting methods, including weighting loss, more hyperparameter tuning and grid search, dataset preparation, and more. For the future, we'd like to implement include contrastive learning approaches, such as SimCLR for audio (14).

8 Contributions

Nick:

- **Code:** Training loop, baseline model experiments, hyperparameter tuning, CNN implementation, final code clean up
- **Report:** Literature review, baseline, methods, experiments & results, final presentation slides

Serena:

- **Code:** Training loop, data augmentation, data processing, final code clean up, baseline model experiments, CNN implementation, hyperparameter tuning
- **Report:** Datasets & features, methods, experiments & results, final presentation slides

Chris:

- **Code:** CNN implementation, development and implementation of CRNN & CRNN + Attention models, baseline model experiments, hyperparameter tuning
- **Report:** Literature review, methods, results, model figures, section revisions
- **Note:** Chris was sick for several weeks of the quarter.

References

[1] G. Deshpande and B. W. Schuller, "Audio, speech, language, & signal processing for covid-19: A comprehensive overview," *arXiv preprint arXiv:2011.14445*, 2020.

[2] L. Orlandic, T. Teijeiro, and D. Atienza, "The coughvid crowdsourcing dataset, a corpus for the study of large-scale cough analysis algorithms," *Scientific Data*, vol. 8, no. 1, pp. 1–10, 2021.

[3] G. Chaudhari, X. Jiang, A. Fakhry, A. Han, J. Xiao, S. Shen, and A. Khanzada, "Virufy: Global applicability of crowdsourced and clinical datasets for ai detection of covid-19 from cough," *arXiv preprint arXiv:2011.13320*, 2020.

[4] 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," *arXiv preprint arXiv:2006.05919*, 2020.

[5] N. Sharma, P. Krishnan, R. Kumar, S. Ramoji, S. R. Chetupalli, P. K. Ghosh, S. Ganapathy, *et al.*, "Coswara–a database of breathing, cough, and voice sounds for covid-19 diagnosis," *arXiv preprint arXiv:2005.10548*, 2020.

[6] N. K. Chowdhury, M. A. Kabir, M. M. Rahman, and S. M. S. Islam, "Machine learning for detecting covid-19 from cough sounds: An ensemble-based mcdm method," *Computers in Biology and Medicine*, vol. 145, p. 105405, 2022.

[7] M. Esposito, S. Rao, V. Narayanaswamy, and A. Spanias, "Covid-19 detection using audio spectral features and machine learning," in *2021 55th Asilomar Conference on Signals, Systems, and Computers*, pp. 1146–1150, IEEE, 2021.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[9] S. Hamdi, M. Oussalah, A. Moussaoui, and M. Saidi, "Attention-based hybrid cnn-lstm and spectral data augmentation for covid-19 diagnosis from cough sound," *Journal of Intelligent Information Systems*, pp. 1–23, 2022.

[10] S. R. Chetupalli, P. Krishnan, N. Sharma, A. Muguli, R. Kumar, V. Nanda, L. M. Pinto, P. K. Ghosh, and S. Ganapathy, "Multi-modal point-of-care diagnostics for covid-19 based on acoustics and symptoms," *arXiv preprint arXiv:2106.00639*, 2021.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[12] Q. Zhou, J. Shan, W. Ding, C. Wang, S. Yuan, F. Sun, H. Li, and B. Fang, "Cough recognition based on mel-spectrogram and convolutional neural network," *Frontiers in Robotics and AI*, p. 112, 2021.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[14] D. Jiang, W. Li, M. Cao, W. Zou, and X. Li, "Speech simclr: Combining contrastive and reconstruction objective for self-supervised speech representation learning," *arXiv preprint arXiv:2010.13991*, 2020.

Appendices

A Model equations

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Figure 6: LSTM equations

B Curves

C Detailed model architectures

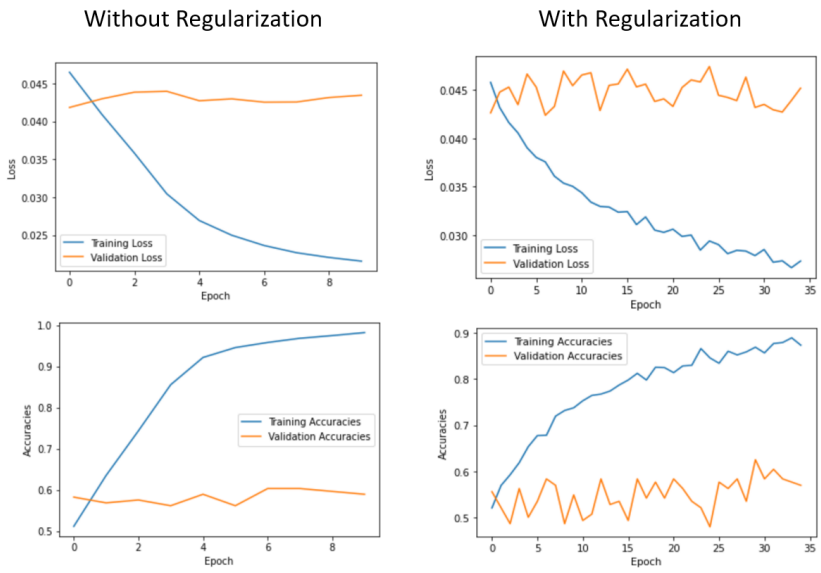

Figure 7: Loss and Accuracy Curves for Our Large CNN Model during Preliminary Testing. The graphs on the right are indicative that our regularization attempts (adding dropout and weight decay) appear to work. However, even with proper regularization, the model fails to stop overfitting (and testing our model on our test set confirms this). This result helps confirm that a model performs significantly worse if it cannot analyze the temporal aspect of cough audio data for COVID-19 classification.

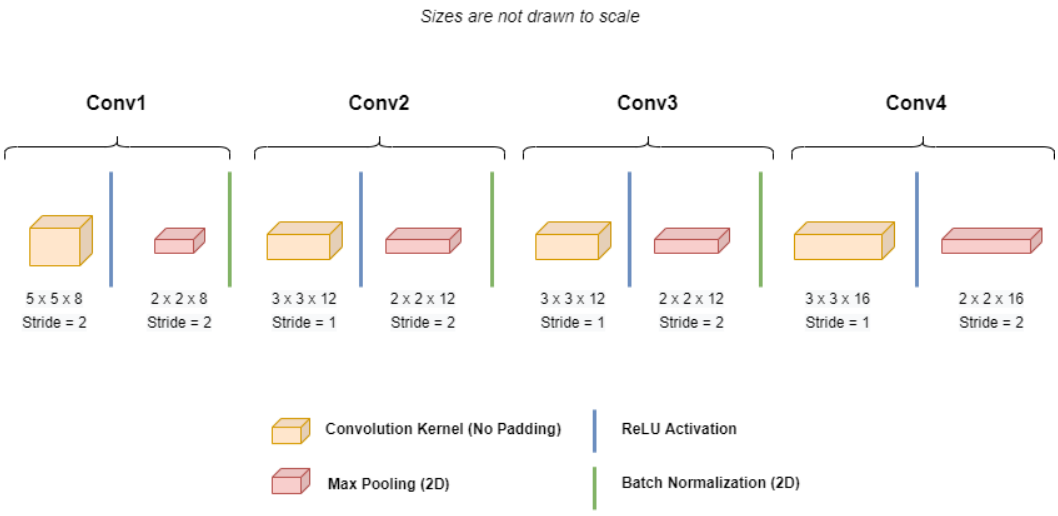

Figure 8: Detailed Visual Displaying Convolutional Layers of Small CNN

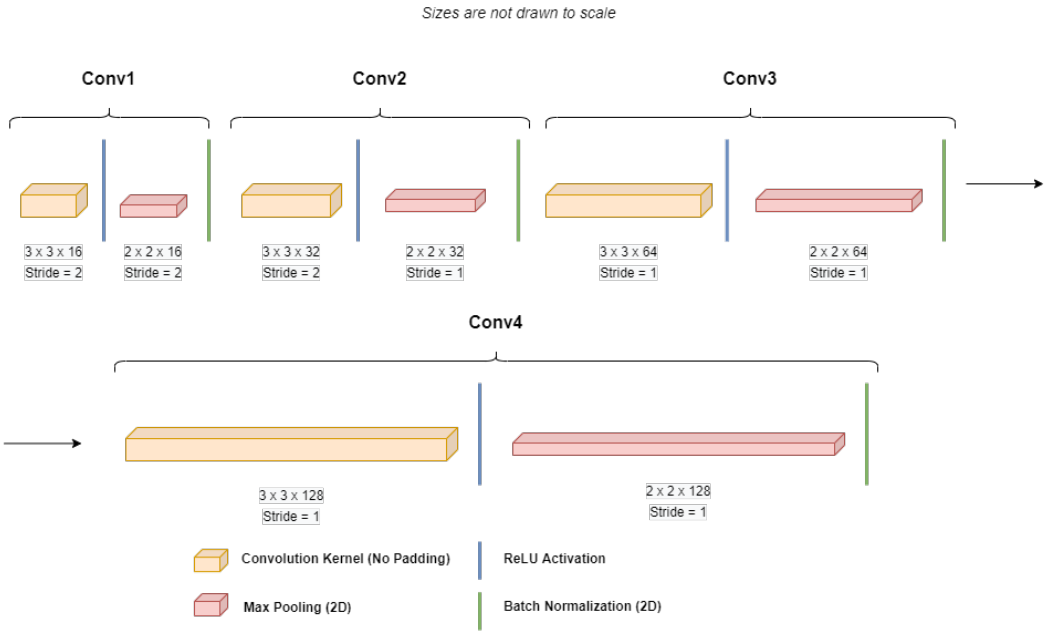

Figure 9: Detailed Visual Displaying Convolutional Layers of Large CNN

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN                                      [256, 2]                  --
├─Sequential: 1-1                        [256, 8, 31, 77]          --
│    └─Conv2d: 2-1                       [256, 8, 62, 155]         208
│    └─ReLU: 2-2                         [256, 8, 62, 155]         --
│    └─MaxPool2d: 2-3                    [256, 8, 31, 77]          --
│    └─BatchNorm2d: 2-4                  [256, 8, 31, 77]          16
│    └─Dropout: 2-5                      [256, 8, 31, 77]          --
├─Sequential: 1-2                        [256, 12, 14, 37]         --
│    └─Conv2d: 2-6                       [256, 12, 29, 75]         876
│    └─ReLU: 2-7                         [256, 12, 29, 75]         --
│    └─MaxPool2d: 2-8                    [256, 12, 14, 37]         --
│    └─BatchNorm2d: 2-9                  [256, 12, 14, 37]         24
│    └─Dropout: 2-10                     [256, 12, 14, 37]         --
├─Sequential: 1-3                        [256, 12, 6, 17]          --
│    └─Conv2d: 2-11                      [256, 12, 12, 35]         1,308
│    └─ReLU: 2-12                        [256, 12, 12, 35]         --
│    └─MaxPool2d: 2-13                   [256, 12, 6, 17]          --
│    └─BatchNorm2d: 2-14                 [256, 12, 6, 17]          24
│    └─Dropout: 2-15                     [256, 12, 6, 17]          --
├─Sequential: 1-4                        [256, 16, 2, 7]           --
│    └─Conv2d: 2-16                      [256, 16, 4, 15]          1,744
│    └─ReLU: 2-17                        [256, 16, 4, 15]          --
│    └─MaxPool2d: 2-18                   [256, 16, 2, 7]           --
│    └─BatchNorm2d: 2-19                 [256, 16, 2, 7]           32
│    └─Dropout: 2-20                     [256, 16, 2, 7]           --
├─Flatten: 1-5                           [256, 224]                --
├─Linear: 1-6                            [256, 2]                  450
├─Softmax: 1-7                           [256, 2]                  --
==========================================================================================
Total params: 4,682
Trainable params: 4,682
Non-trainable params: 0
Total mult-adds (G): 1.17
```

Figure 10: CNN small model

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNNNetwork                               [256, 2]                  --
├─Sequential: 1-1                        [256, 16, 31, 78]         --
│    └─Conv2d: 2-1                       [256, 16, 63, 156]        160
│    └─ReLU: 2-2                         [256, 16, 63, 156]        --
│    └─MaxPool2d: 2-3                    [256, 16, 31, 78]         --
│    └─BatchNorm2d: 2-4                  [256, 16, 31, 78]         32
│    └─Dropout: 2-5                      [256, 16, 31, 78]         --
├─Sequential: 1-2                        [256, 32, 14, 37]         --
│    └─Conv2d: 2-6                       [256, 32, 15, 38]         4,640
│    └─ReLU: 2-7                         [256, 32, 15, 38]         --
│    └─MaxPool2d: 2-8                    [256, 32, 14, 37]         --
│    └─BatchNorm2d: 2-9                  [256, 32, 14, 37]         64
│    └─Dropout: 2-10                     [256, 32, 14, 37]         --
├─Sequential: 1-3                        [256, 64, 11, 34]         --
│    └─Conv2d: 2-11                      [256, 64, 12, 35]         18,496
│    └─ReLU: 2-12                        [256, 64, 12, 35]         --
│    └─MaxPool2d: 2-13                   [256, 64, 11, 34]         --
│    └─BatchNorm2d: 2-14                 [256, 64, 11, 34]         128
│    └─Dropout: 2-15                     [256, 64, 11, 34]         --
├─Sequential: 1-4                        [256, 128, 8, 31]         --
│    └─Conv2d: 2-16                      [256, 128, 9, 32]         73,856
│    └─ReLU: 2-17                        [256, 128, 9, 32]         --
│    └─MaxPool2d: 2-18                   [256, 128, 8, 31]         --
│    └─BatchNorm2d: 2-19                 [256, 128, 8, 31]         256
│    └─Dropout: 2-20                     [256, 128, 8, 31]         --
├─Flatten: 1-5                           [256, 31744]              --
├─Linear: 1-6                            [256, 2]                  63,490
├─Softmax: 1-7                           [256, 2]                  --
==========================================================================================
Total params: 161,122
Trainable params: 161,122
Non-trainable params: 0
Total mult-adds (G): 8.53
```

Figure 11: CNN large model

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CRNN                                     [256, 2]                  --
├─Sequential: 1-1                        [256, 16, 31, 78]         --
│    └─Conv2d: 2-1                       [256, 16, 63, 156]        160
│    └─ReLU: 2-2                         [256, 16, 63, 156]        --
│    └─MaxPool2d: 2-3                    [256, 16, 31, 78]         --
│    └─BatchNorm2d: 2-4                  [256, 16, 31, 78]         32
│    └─Dropout: 2-5                      [256, 16, 31, 78]         --
├─Sequential: 1-2                        [256, 32, 14, 37]         --
│    └─Conv2d: 2-6                       [256, 32, 15, 38]         4,640
│    └─ReLU: 2-7                         [256, 32, 15, 38]         --
│    └─MaxPool2d: 2-8                    [256, 32, 14, 37]         --
│    └─BatchNorm2d: 2-9                  [256, 32, 14, 37]         64
│    └─Dropout: 2-10                     [256, 32, 14, 37]         --
├─Sequential: 1-3                        [256, 64, 11, 34]         --
│    └─Conv2d: 2-11                      [256, 64, 12, 35]         18,496
│    └─ReLU: 2-12                        [256, 64, 12, 35]         --
│    └─MaxPool2d: 2-13                   [256, 64, 11, 34]         --
│    └─BatchNorm2d: 2-14                 [256, 64, 11, 34]         128
│    └─Dropout: 2-15                     [256, 64, 11, 34]         --
├─Sequential: 1-4                        [256, 128, 8, 31]         --
│    └─Conv2d: 2-16                      [256, 128, 9, 32]         73,856
│    └─ReLU: 2-17                        [256, 128, 9, 32]         --
│    └─MaxPool2d: 2-18                   [256, 128, 8, 31]         --
│    └─BatchNorm2d: 2-19                 [256, 128, 8, 31]         256
│    └─Dropout: 2-20                     [256, 128, 8, 31]         --
├─LSTM: 1-5                              [256, 31, 512]            2,625,536
├─Tanh: 1-6                              [256, 31, 512]            --
├─Sequential: 1-7                        [256, 512, 1]             --
│    └─Linear: 2-21                      [256, 512, 1]             32
│    └─ReLU: 2-22                        [256, 512, 1]             --
├─Flatten: 1-8                           [256, 512]                --
├─Linear: 1-9                            [256, 2]                  1,026
├─Softmax: 1-10                          [256, 2]                  --
==========================================================================================
Total params: 2,724,226
Trainable params: 2,724,226
Non-trainable params: 0
Total mult-adds (G): 29.35
```

Figure 12: CRNN model

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CRNN_with_Attention                      [256, 2]                  --
├─Sequential: 1-1                        [256, 16, 31, 78]         --
│    └─Conv2d: 2-1                       [256, 16, 63, 156]        160
│    └─ReLU: 2-2                         [256, 16, 63, 156]        --
│    └─MaxPool2d: 2-3                    [256, 16, 31, 78]         --
│    └─BatchNorm2d: 2-4                  [256, 16, 31, 78]         32
│    └─Dropout: 2-5                      [256, 16, 31, 78]         --
├─Sequential: 1-2                        [256, 32, 14, 37]         --
│    └─Conv2d: 2-6                       [256, 32, 15, 38]         4,640
│    └─ReLU: 2-7                         [256, 32, 15, 38]         --
│    └─MaxPool2d: 2-8                    [256, 32, 14, 37]         --
│    └─BatchNorm2d: 2-9                  [256, 32, 14, 37]         64
│    └─Dropout: 2-10                     [256, 32, 14, 37]         --
├─Sequential: 1-3                        [256, 64, 11, 34]         --
│    └─Conv2d: 2-11                      [256, 64, 12, 35]         18,496
│    └─ReLU: 2-12                        [256, 64, 12, 35]         --
│    └─MaxPool2d: 2-13                   [256, 64, 11, 34]         --
│    └─BatchNorm2d: 2-14                 [256, 64, 11, 34]         128
│    └─Dropout: 2-15                     [256, 64, 11, 34]         --
├─Sequential: 1-4                        [256, 128, 8, 31]         --
│    └─Conv2d: 2-16                      [256, 128, 9, 32]         73,856
│    └─ReLU: 2-17                        [256, 128, 9, 32]         --
│    └─MaxPool2d: 2-18                   [256, 128, 8, 31]         --
│    └─BatchNorm2d: 2-19                 [256, 128, 8, 31]         256
│    └─Dropout: 2-20                     [256, 128, 8, 31]         --
├─LSTM: 1-5                              [256, 31, 512]            2,625,536
├─Tanh: 1-6                              [256, 31, 512]            --
├─Linear: 1-7                            [256, 31, 512]            262,656
├─Linear: 1-8                            [256, 31, 512]            262,656
├─Linear: 1-9                            [256, 31, 512]            262,656
├─MultiheadAttention: 1-10               [256, 31, 512]            1,050,624
├─Sequential: 1-11                       [256, 512, 1]             --
│    └─Linear: 2-21                      [256, 512, 1]             32
│    └─ReLU: 2-22                        [256, 512, 1]             --
├─Flatten: 1-12                          [256, 512]                --
├─Linear: 1-13                           [256, 2]                  1,026
├─Softmax: 1-14                          [256, 2]                  --
==========================================================================================
Total params: 4,562,818
Trainable params: 4,562,818
Non-trainable params: 0
Total mult-adds (G): 29.55
```

Figure 13: CRNN + Attention model