Data-Driven Severity Analysis for COVID-19 via Lung CT Images and Convolutional Neural Networks

Shray Alag  
Computer Science  
Stanford University  
alag@stanford.edu

Aksh Garg  
Computer Science  
Stanford University  
akshgarg@stanford.edu

Covid-19 has been lifechanging. Yet, Covid-19 diagnosis still fails to assess a severity of patient’s cases; however, using chest-CT data we can effectively do so. In our paper, we create a novel end-to-end 3D convolutional neural network pipeline for assessing patient severity based on their chest CT images. We train our models on the MosMed Dataset, which contains 1130 images of patients for 5 levels of COVID-19 severity (Zero, Mild, Moderate, Severe, Critical). We first preprocess all images, converting them from hounsfield units to numpy arrays and then apply a series of data augmentations to improve generalization capacity. Given the large dimensionality of our input images, we create several custom shallow CNN architectures and iteratively refine and optimize them, paying attention to learning rates, layer types, normalization types, filter sizes, dropout values, and more. For further exploration, we experiment with residual networks, CNN architectures, transfer learning, and saliency graphs. We combat the large class imbalance problem in our dataset by designing custom loss and accuracy metrics, which weigh our images by the inverse of the number of classes in that class. Our models attain 88.77% validation accuracy when it comes to distinguishing healthy patients from COVID-19 positive patients and 67.06% accuracy in predicting their severity for all 5 classes.

1. Introduction

From 6.29 million worldwide Covid-19 induced deaths to 529 million reported cases to social and economic disruptions, the Covid-19 pandemic continues to influence day-to-day life [1]. Moreover, the ever-changing nature COVID-19’s variants illustrate that the pandemic may not end as soon as hoped [1].

Given the longevity and severity of the pandemic, effective diagnostic and severity testing for Covid-19 is essential. Currently, the reverse transcription-polymerase chain reaction, RT-PCR, is the gold-standard for COVID-19 diagnosis [1]. However, PCR has a high false-negative rate and is unable to specify the severity of the infection. Understanding the severity is becoming increasingly more essential to the wellbeing of patients. Knowing the severity of COVID-19 allows patients to better prepare and obtain relevant medical advice. Depending on their situation, patients may require extra oxygen at hand, need to take stronger medications, be monitored in the ER, etc: such factors are crucial for decreasing mortality. Additionally, from a hospital standpoint, knowing the severity of patient’s cases, allows them to effectively triage and distribute treatments. Especially during times when hospitals are strained, this is crucial.

Creating a severity-analysis test for Covid-19 is a non-trivial challenge even with modern day computing power: ideally the test must be accurate, perform better than current medical professions, quick, and data-driven. Luckily, one can utilize deep learning (DL) algorithms, which can analyze data in seconds and learn intricate features of data without the need for explicit programming.

CT-scans are a promising modality for applying DL approaches for assessing COVID-19 severity in patients. As a 3 dimensional computer processed patching of a series of X-rays—each X-ray taken from an unique angle [3], a CT-Scan provides invaluable insights into internal tissue/muscle visualization. In fact for Covid-19, the data collection properties of CT-scans make it uniquely poised to view the presence of ground-glass opacities, paving patterns, and lesions, which are crucial in determining the severity of Covid-19 [3].

Considering these factors, we attempted to create a fully data-driven CT-scan based severity assessment tool for COVID-19. Our model takes in an input 3-dimensional CT scans and outputs patients’ severity on a 1-5 scale.

2. Related Work

Analyzing the severity of Covid-19 from such scans has been looked into, but pipelines aren’t end-to-end, fully data-driven, or without significant a priori knowledge. With the
pandemic raging and new variants emerging, advancements in severity analysis would be incredibly beneficial to the medical community. In the following subsections, we proceed into more depth about each of the recent work across different subfields.

2.1. Convolutional Neural Networks Covid-19 Advancements

One of the first works to analyze Covid-19 was conducted by Wang et al. [4] where the authors focused on distinguishing between pneumonia and Covid-19 infections, building their deep CNN model around chest X-Ray data. Breve et al. [5] furthered this work, examining the same dataset but with three different Convolutional Neural Network (CNN) architectures. Breve et al. [5] continued this through a CNN framework, bolstering the practical merit behind such an approach. One of the three principle architectures in Breve et al. is the acclaimed Inception network pioneered in Szegedy et al. [2]. In addition, COVID-Net CT created by Gunraj et al. [6] was trained on CT images for determining whether or not the patient has Covid-19. Sahlol et al. [7] begins to drift away from a pure CNN approach, combining a CNN framework with a swarm-based feature selection approach on the same problem.

Perhaps intuitively due to the nature of the X-ray and CT-scan data, the majority of deep learning approaches stem from CNNs. However out of the recent CNN work, there are fascinating intricate approaches to the problems of classification that traverse beyond standard architectures: To achieve a high accuracy, the authors of Horry et al. [8] employ self-supervising transfer learning to lessen overfitting. Zhang et al. [9] focused on dual pipeline of image segmentation and then classification NN to predict whether the patient has Covid-19.

2.2. Severity Assessment Past Work

There has not been a substantial research on identifying the severity of Covid-19 in a patient, and even less when it comes to dealing with CT-scans. Aswathy et al. [10] used a residual U-Net model for severity assessment on 2 dimensional CT data. Another approach used by both Feng et al. [7] and Yazan et al. [11] uses a mix of encoder-decoder methods to extract lesions. Similarly, Lessmann et al. [12] utilized several pre-trained CNNs to assign severity scores on 2 dimensional CT data. However, these methods still struggle to effectively rank severity and take in a significant amount of a priori information. In particular, even when these approaches utilize CT-scans, they use 2 dimensional CT slides despite the greater insights provided by 3D CT-data. Other methods in severity assessment focus on using ensembling techniques like random forests that use pre-existing image features clinical biomarkers for predicting patient severity. Some prominent works within this domain have created a hierarchy of important features (Wu et al. [13]), explored the importance of histogram maps in severity scores (Wu et al. [13]), and attempted to learn probability distribution of at-risk and healthy patients (Rubin et al. [14]).

2.3. Our Objective

While the categories of methods described above are successful, determining the severity of Covid-19 has only been looked into by two or three research teams. Given the vast potential of 3D CT-scans as highlighted by recent publications in the CT-scanning field, suggests that CT’s potential is currently untapped for COVID-19 [15].

Furthermore, most of the work so far has not been end-to-end, relying upon two distinct pipelines and a priori insights for development, many of which require intervention from medical experts. In response, we developed an end-to-end system for COVID-19 severity assessment from 3D Chest CT scans with no additional manual feature introduction into the system while harnessing the extra detail 3 dimensional scans provide. Our approach is CNN-based but we draw insights from recurrent networks as well.

Finally, it is worth noting that due to the inherently novel nature of our approach, large amounts of our exploration was unfruitful. Considering that we had no predecessors to even confirm or deny the feasibility of our approach, to our knowledge at the time we started it had never been done before, many unsuccessful explorations are to be expected.

3. Methods

3.1. Environmental Constraints

Before discussing the twelve sub-approaches and design-arms we followed, it is crucial to understand the environmental constraints we dealt with. Given the large size of the images—64 128x128 images for each of the 1110 CT Scans—we had to be cautious about the structure of the CNNs we used. We accessed a research lab’s GPU cluster; however, even with extremely powerful GPUs with 32 GB of memory, we could only train our models with a batch size of 2/4 and had use smaller self-designed architectures instead of existing models like ResNet, EfficientNet, etc.

Furthermore, much of the previous work in this field utilizes 2D CT-scans and can therefore use well known architectures (Inception, ResNet, etc.); however, given we wanted to examine the entire 3D volumes simultaneously, we did not have that luxury and had to start from scratch. One option we considered to decrease the memory constraints was further decreasing the resolution of the images. We decided against it as this would likely blur/lessen the presences of ground-glass opacity and legions, which is crucial for distinguishing the severity.
3.2. Architecture Design from Memory Perspective

Taking into account our limited memory space, we trained our models with a batch size of 2, used the Adam optimizer to take advantage of the momentum picked up from each update, and deleted any variables not needed for subsequent operations. Furthermore, we structured our convolutions such that increases in the number of channels mirrored equivalent if not greater reductions in the images spatial depth. By doing so, we ensured each stage of our convolution operation fit within our memory constraints.

3.3. Architecture Design

Following the state-of-the-art approaches for CNNs, we design each CNN layer to follow the Conv3D-MaxPool3D-Batch Normalization structure, omitting dropout initially as to ensure that the framework learns. When visually inspecting our data initially, we noticed that indicative features of COVID-19 like ground glass opacities and paving patterns were all relatively small. To ensure our model was able to learn features from both small and large pixel windows, we used small 3x3x3 filters and increased the number of layers in our network to increase the effective spatial resolution. As detailed in the following subsections, this basic framework is thoroughly tinkered with, trying out Average Pooling, Layer Norm, and Dropout with different intensities, etc. We utilized Adam optimization, early stopping (patience varied), accuracy as initial metric, and multi-class cross entropy loss. Each architecture ended with a global average pooling layer followed by dense layers for obtaining class probabilities.

3.4. Small, Medium, Large Models

To begin our exploration, we created three standard CNN architectures of varying depths to operate on all five classes: 4 total layers including fully connected (FC) for "small," 5 layers for "medium," and 6 layers for "large," each trained until early stopping cut it off.

3.5. Class Imbalance Problem

As mentioned in the results section, the three models above did not learn very well. The root of this problem is the severe class imbalance in our dataset: because 61.6% of scans are all of the class CT-1, the model was classifying everything as this class. We devised four main approaches to deal with the class imbalance: (1) custom weighted loss function, (2) custom True Positive and False Positive metrics, (3) grouping classes together to even out the overall distribution of the data (e.g. combining classes CT-2, CT-3, and CT-4 to have 172 samples), and (4) converting the problem to a binary classification one.

3.6. Custom Loss Function

We noted the model consistently ignored the less predominant classes like CT-3 and CT-4. Thus, we created a low-level loss function which used the $y_{pred}$ and $y_{true}$ labels, and scaled each term in the original loss output by the inverse of the number of images in that image’s true class. We didn’t find any multi-class weighted loss function that was suitable so we implemented our own custom loss function using Keras and tested our loss function to make sure it was working before applying it to the architectures above and below.

The equation for our custom loss function is shown below where $L$ is the total loss, $y_j$ is the label for the $j$-th training sample, and $\log(\hat{y}_j)$ is the $j$-th logit.

$$L = \sum_{j=1}^{n_{train}} \frac{1}{n_y} y_j \log(\hat{y}_j)$$

(1)

3.7. Custom Metrics

We were using the standard accuracy metric for our three architectures above. However, the raw accuracy doesn’t provide insights into if the model is predicting all samples to be the same class. Realizing we needed a way to see how many true positives and false negatives the model was predicting for each class, we created custom metrics using the Keras backend for each model architecture. These metrics were shown after each epoch. In addition, we computed precision, recall, and F1-scores for each class and found the macro and weighted averages of these metrics over all classes to get composite scores for our models. Such metrics enabled us to make more well informed decisions about the model performance and architecture.

3.8. Model Redesign, Hyperparameter Tuning

Even after making these architecture changes, the performances of our models were not convincing enough to definitely say that they were learning the key features of CT-scans. To improve our models, we pivoted to tuning our hyperparameters and fine-tuning our network architecture. Given our model-created-from-scratch approach and the absence of previous severity analysis 3D CT-scan research, we expected that our initial model architecture design could use substantial refining. To remedy this, we explored the following paths thoroughly.

3.8.1 Batch Normalization v. Layer Normalization

Due to our memory constraints, our batch size could only be 2 or 4. Because such a small batch size would substantially hamper the benefits, we compared the performance of batchnorm to layernorm, which is not sensitive to the
batch size and computes means/variances across each image’s features directly. We fully trained batchnorm and layernorm on 5 different model architectures and comparing the results between layernorm and batchnorm, finding layernorm to be slightly more effective.

3.8.2 Dropout

As our model began to learn the features of the different classes, we began needing to use regularization techniques to prevent overfitting. The prime regularization technique we employed was Dropout due to its recent success in Deep Learning literature. We ran 6 experiments trying out different dropout amounts (0, 0.2, 0.4) on two different architectures.

3.8.3 Kernel Size

We experimented with two different kernel sizes 3x3 and 5x5. Following standard practices from literature, we preferred stacking several layers with smaller filter sizes instead of having a fewer layers with larger filter sizes. Logically, more layers with smaller filter sizes enables the expression of more complex features of the input with fewer convolution parameters. Therefore, we did not venture past 5x5 filters in size.

3.9. Buildup

Having fully optimized each component in the convolutional blocks, we planned to see the optimal depth of our architecture by a building up approach: To see how many layers it would take the model to not only train but also most optimally classify the classes we created models with convolutional blocks of depths 1, 2, 3, 4, and 5. Each block was the optimal architectures, Conv3D-MaxPool3D-Layer Normalization-Dropout layers. Intuitively, we hoped that with their ability to form a larger receptive field, larger networks would perform better. However, we were unsure about the exact depth that would optimize this, and our buildup offered suggestions to base future models upon.

3.10. Residual Connections

Although a model architecture with 4 conv. blocks appeared to be optimal, we wished to venture beyond just the realms of convolutional layers to improve model performance. Taking inspiration from past literature on residual connections, we designed a custom neural network with skip connections from each block to the output in addition to the regular feed-forward connections present in CNNs. The design was analogous to recurrent neural network designs, with each layer having an output. Intuitively, these additions were meant to increase our model’s capacity to learn features of varying sizes and backpropagate more effectively. A schematic representation of the residual connections is presented in figure 1.

3.11. Class Re-evaluation

After exploring the benefits of residual connections and our build-up approach, we noticed there were minimal differences in several classes we were attempting to distinguish between. Namely, CT-0 (Zero) and CT-1 (Mild) COVID-19 images shared many features, making them difficult to distinguish. To simplify our problem landscape, we therefore, pivoted to distinguishing between more disparate classes like CT-0 and CT-2, CT-3, and CT-4. To do so, we collapsed CT-2, CT-3, and CT-4 into a single class. This had two noticeable advantages. First, it simplified our problem from a multi-class classification problem to a binary classification problem, which was useful in our original stages where the model was failing to learn anything whatsoever. Second, it mitigated challenges stemming from class imbalances. With CT-2, CT-3, and CT-4 combined, the positive class contained 125 + 45 + 2 = 172 images, while the negative class (CT-0) contained 254 images. This created a more balanced training/validation dataset.

3.12. Transfer Learning

Across the 40 experiments we ran, exploring specific hyper parameters, model layers, and overarching architectures, a small portion of our models were successful in learning at all, while the others seemed to converge to poor local minima. To not only improve training of future models but to also see if previous architectures would perform better with a weight transfer-learning “headstart,” we decided to further implement transfer learning: We stored weights found from the well performing models. For some models,
which shared the same architecture, the weights copying was straightforward and the new model was simply initialized with an exact copy of the original model’s weights. For models with differing architectures, we copied over weights from matching portions of the model, while leaving the remaining parts untouched.

3.13. Saliency Graphs

After trying a myriad of different architecture approaches, we wished to visualize what the models were actually training on. To do so, we created saliency maps by calculating which pixels in our input image had the greatest effect on the predicted scores. The maps helped us understand what portions of the images our models were looking at and whether their results seemed to be generalizable. Because of the 64 images per CT-scan nature, we made Saliency graphs for each of the 64 images and one Saliency graph for the overall sample (all 64 combined by max pixel value at each location).

3.14. Final Model Design

Taking all our previous results we created our most optimal final model, which was trained with increasing levels of dropout. Specifically, we noticed that a properly initialized model was able to overfit the training dataset really well—often attaining accuracies as high as 98% while validation accuracies trailed far below. To circumvent such issues, we did 2 things. First, we let the training with overfitting keep occurring with a patience of 50. This allowed us to gauge whether the additional features picked up by the model might improve validation performance. In general, once overfitting started occurring, validation losses would increase and accuracies would drop, leading the model to eventually stop training once the patience thresholds were reached. Once this happened, we initialized our model with the best weights as found from the previous trial and increased the overall amount of dropout applied with the hope that it would reduce the amount of overfitting that occurred.

4. Dataset

We trained our models on the ModMedData dataset, a high-detail dataset containing 1110 3D chest CT images of patients in 5 severity levels taking in a hospital in Moscow, Russia:

<table>
<thead>
<tr>
<th>Class Name</th>
<th># Images</th>
<th>Class %</th>
<th>Val Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT-0 (Zero)</td>
<td>254</td>
<td>22.8%</td>
<td>38</td>
</tr>
<tr>
<td>CT-1 (Mild)</td>
<td>684</td>
<td>61.6%</td>
<td>103</td>
</tr>
<tr>
<td>CT-2 (Mod)</td>
<td>125</td>
<td>11.3%</td>
<td>19</td>
</tr>
<tr>
<td>CT-3 (Severe)</td>
<td>45</td>
<td>4.1%</td>
<td>7</td>
</tr>
<tr>
<td>CT-4 (Critical)</td>
<td>2</td>
<td>0.2%</td>
<td>1</td>
</tr>
</tbody>
</table>

It is crucial to note that the CT-0 class includes 254 3D CT volumes from individuals with no signs of pneumonia or COVID-19, whereas scans in group CT-1 show 25% Covid-19 infection on the lungs, CT-2 is 50% COVID-19 infection, etc. The vagueness of the criterion to bucket a Covid-19 CT-scan into these five categories is a noticeable drawback in MosMed as certain scans across classes may be extremely similar. Other publications dealing with this data hint at this deficit and tangible repercussions of the semi-handwavey bucketing system are seen in our results section [15]. However, the MosMed dataset was the only publically available dataset for severity assessment that we could find, and thus we used it for our severity analysis task.

4.1. Preprocessing

The MosMed CT scans were saved in the NIfTI format, which is a three dimensional modeling format commonly used for medical scans. As most of the previous literature dealt with a database of 2 dimensional CT-scan images, we had to look towards other literature for the common preprocessing steps for three dimensional scans [15] [14]. After an in-depth review of data processing methods for such CT-scans, we followed the current gold-standard preprocessing approach:

All of the CT scans were converted to PNGs, as is common practice when handling these type of data. With each CT scan translating to 64 slices, the resulting dataset contained large 512x512x64 images, with pixel values ranging -1024 to 2000. We then did the following normalization, cropping, shuffling, data-splitting, and data augmentation techniques:

1. Followed standard normalization procedure for CT scans and clipped pixel values between -1000 and 400 (above 400 corresponds to bones).
2. Zoomed in and reduced images to 128x128x64.
3. Shuffled our data and split into the training, validation, and testing sets using a 70-15-15 split (777-167-167 images).
4. Augmented our dataset using 3D rotations to improve generalization.

4.2. Data Visualization

To visualize the initial dataset, Figure 2b), 2c), and 2d) illustrates four out of 64 image slices for a CT-scan patient with no presence of Covid or pneumonia before preprocessing. In addition, to visualize the results of preprocessing, Figure 2a is an image of a 2D slice of the CT-scan after being normalized, cropped, and augmented.

5. Results

Hyperparameters: learning rate, 0.001; mini-batch size, 2; dropout parameter, 0.4; of convolutional blocks, 4; nor-
Figure 2. Four of the 64 slices from a CT-0 class 3D CT-scan before preprocessing.

normalization type, layer normalization, were determined using various tests described in the methods section. Metrics like accuracy, true positives, false negatives, precision, recall, F1-scores, macro and weighted averages. Methods for more detail.

5.1. Original small, medium, and large models

Original tests from these models were unsuccessful in training. The models seemed to consistently produce validation accuracies of around 61%, which likely stemmed from the fact that CT-1 made up 61.6% of the total images in the dataset. Thus, the models had a difficult time capturing the complex features of the image. To some extent, these results made sense. One, our input images volumes were extremely large, which made learning patterns with roughly 1000 pictures difficult. Moreover, these network designs were based on intuition and theoretical expectations, but we hadn’t performed hyperparameter tuning.

5.2. Weighted Loss Function

The inclusion of a weighted loss function managed to successfully balance out performance on each class. However, it came at the expense of overall model performance, dropping the overall accuracy to 42.1%. The performance gaps raised an important question of which metric enhanced our model’s capacity to learn relevant features from the CT images.

5.3. Hyperparameter Modifications

Through our hyperparameter tuning, we found that a learning rate of 0.0001 to be ideal. Additionally, we found dropout when added initially inhibited training. Thus, we trained our first batch of models without dropout and then subsequently added dropout on those trained weights to retrain and reduce overfitting in our final models. Finally, with batch sizes of only 2, batch normalization gave poor estimates of our features’ means and variances. Therefore, we used layer normalization for subsequent testing. More information for hyperparameter tuning and why we chose those values are included in the method section above.

5.4. Buildup

Analyzing the performance of models of varying layer sizes, we noticed that models of layer size 1 and 2 were unsuccessful in learning. Models with layer sizes 3, 4, and 5 learn some features. Table 2 summarizes results from each trained model. It’s easy to discern that the model of size 4 performs the best overall, attaining a peak training accuracy of 0.85190 and validation accuracy of 0.794.

5.5. Residual Layers

We noticed two things when examining our results from our models with residual connections.

One, the residual connections seemed to hamper the learning process. This was because now the early layers had to learn all the complex features needed for the final dense classification and they couldn’t solely focus on learning more general patterns that would be used by later layers. Figure 4, 5a, and 5b present the accuracy plots from models without and with residual connections. We can see that without residual connections the model is able to overfit and thereby learn patterns in the images more effectively.
dictors of the final classes compared to outputs from earlier layers. This is apparent in figure 5a), where the later output layers clearly capture more coherent patterns from the images.

5.6. Transfer Learning

Jumpstarting model learning by initializing them accelerated model training, pretty significantly. With randomized weights, increases in model accuracy would only start occurring by epoch 30. We can see these patterns in figures 6a) and 6b), where initializing weights using pre-learned features allows the models to start learning immediately.

5.7. Class Re-evaluation

Table 3 compares our model’s performances for 3 different classification tasks: (1) CT-0 vs CT-1, (2) CT-0 vs CT-2, and (3) CT-0 vs CT-2, CT-3, CT-4 Combined. All models were initialized using pre-trained weights and trained with a patience of 50 for 100 epochs. Note that raw accuracies are a slightly misleading metric here, as the classification problems each have different levels of class imbalances. Our model seems to perform comparably on problems (1) and (2), suggesting that the relatively easier problem of differentiating between classes 0 and 2 is counterbalanced by the smaller number of images available. We do note, however, that task (3) is performed extremely well. Specifically, our model attains 99.24% training accuracy and 87.72% validation accuracy, well above the imbalance thresholds.

Table 4. Model performance vs. Type of Problem

<table>
<thead>
<tr>
<th>Type</th>
<th>Train Acc.</th>
<th>Val Acc.</th>
<th>Imbalance</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>0.9756</td>
<td>0.8085</td>
<td>72.9%</td>
</tr>
<tr>
<td>(2)</td>
<td>0.7727</td>
<td>0.7368</td>
<td>67.01%</td>
</tr>
<tr>
<td>(3)</td>
<td>0.9924</td>
<td>0.8772</td>
<td>59.6%</td>
</tr>
</tbody>
</table>

5.8. Saliency Graphs

To visualize the model’s actions under-the-hood, we chose to implement saliency graphs, showing the important features of the images the model learns. The Saliency graphs were made for each of the CT slices and are seen in Figure 7. The brighter dots indicate a higher significance for that pixel, so there seems to be a correlation between the lung region of the image and the brightness of the saliency graph, validating our approach as our final model is learning some elements of the scan. However, our graphs are far from perfect predominantly due to human variation in lung size (where each individual’s lungs take up a different amount of the CT-scan). Additionally, lighting could greatly affect the saliency graphs, but the im-

Figure 4. Accuracy plots for Model with Residual Connections

Figure 5. Accuracy and Loss plots for Class 0 vs 2,3,4 combined
Importance of lighting in CT-scans is unclear. It is significant to note the brightness of the saliency graphs on the corners of the images, indicating that for certain reasons—perhaps light, perhaps time/year when the CT-scan was taken, perhaps the hospital they were taken at—the model thinks the corners are significant. This also highlights a drawback to our model and some further intuition on how it can be improved.

Figure 6. Saliency Maps for two of the 64 2D CT slices. On the left is the saliency map and are the right are the CT image inputs for that corresponding picture.

5.9. Final Model

We saw in the earlier section that our model could distinguish between two classes with a validation accuracy as high as 87.72%. However, these high accuracies were high to replicate when predicting across all 5 classes. Amongst all experiments, the best validation accuracy was 0.6706% and the best training accuracy was 0.8023%. Figure 7 presents the training/loss curves for this experiment.

6. Conclusion, Future Work

Our work attempted to design a novel end-to-end 3D convolution pipeline for automatically assessing the severity of COVID-19 from lung CT scans. Given our small dataset size and large input image dimensions, we iterated through many procedures for effective severity assessment. Our final, best performing models were able to attain 88.7% accuracy in binary classification and 67.06% accuracy for 5-class classification. Our final model is still overfitting and we believe that it is being overwhelmed by too many input weight parameters. For future work, we recommend the following approaches:

1. More effective transfer learning: A major obstacle we encountered was the limited amount of images we had to learn and the large size of each image volume. In essence, we only had 1000 datapoints to design a neural network for extracting features from a 64 slice deep image, which was challenging. A self-supervised approach would improve our model’s capacity to learn from small amounts of data. Alternatively, one could try looking into one-shot learning techniques like siamese networks, which are also less reliant upon large sample sizes. Finally, although we tried to create a fully self-contained pipeline for 3D image analysis, a combination of 2D and 3D approaches might be fruitful. For example by copying weights from 2D architectures for COVID-19 across the entire 3D architecture, we might be able to boost performance.

2. GANs for making more images: Another useful way of increasing the number of samples we were dealing with could have been to create new images of our own. This would have not only increased the number of sample available for training, but also allowed us to circumvent imbalance issues.
References


