Question 2: Cov-AI-d to the rescue!

1 Team

Team Members	SooBin Kim (11234151), Rishabh Singal (18802141) Ana Cecilia Leon Morales (16559205), Giorgio Andre Viancha Sgarbi (11787132), Kushal Mohee (34236166)
Kaggle Team Name	*Cov-AI-d*

2 Introduction

The COVID-19 classification problem is a binary image classification problem. It involves the correct classification of lung images of COVID-19 patients as opposed to patients infected with other viruses which are termed as ‘background’ in this case. The main issue with this classification problem is the size of the dataset. Training would be limited to 70 examples and testing to 20. On a bigger picture, the problem with classifying COVID-19 patients is that the determining features of the disease are still unclear and extensive exploratory analysis would be required to identify trends and make predictions. Furthermore, many images in the dataset contain text on them which can potentially cause CNN to learn “bad” features such as text on the image instead of the more useful features such as lung contours, size, etc.

3 Method

To approach this classification problem, we explored numerous machine learning techniques such as Data Augmentation, Transfer Learning, Ensemble Methods, Keras models amongst others. We will begin our discussion with the more promising ones.

To tackle the issue of a small dataset, we made the use of Data Augmentation. We spent some time identifying which kinds of transforms (Augmentations) would be the most appropriate for lung images. Some might say that feeding the model images that were flipped horizontally is not a good idea since the location of a heart is mostly on the left of the chest and might be wrongly categorized as an infection (false positive). However, other transforms such as resizing and colour jitter would be appropriate. Given that we pursued a Transfer Learning approach (discussed later), we mostly used transforms that were standard for the pre-trained model used.

Machine Learning techniques such as Convolutional Neural Networks and other Neural Networks in Image classification often demand very large datasets to maximize learning. To leverage their potential with smaller datasets, we might need to train these models with other, somewhat, related datasets first [2]. This technique is known as Transfer Learning. To apply it to our classification problem, we first needed to determine which parts of the model to fine tune. Pre-trained models such as Resnet, Densenet, VGG... use data that is quite different from our data and our dataset (70 examples) is much smaller in comparison. The chosen approach was to freeze the parameters used in the convolutional layers but fine tuning the output dense layer of the pre-trained model [3].

Ensemble Methods were also considered as a viable approach. These methods can be very powerful given that it requires numerous models to corroborate on the output label. They, however, require a lot of computational power to train the numerous models and get predictions. Therefore, it was not an accessible approach for our team.

4 Experiments

4.1 Transfer Learning

In this attempt, we used the torchvision pytorch package to perform the Machine Learning techniques. The Data Augmentations performed on the training data were resizing to half the original size and mean normalization while only resizing and normalization was done on the test data. The training data was split into a training and validation set. Numerous pre-training models were considered along with hyper-parameters to be tuned according to validation accuracy.
Pre-trained models: Resnet18, Resnet 142, ResNet152 DensetNet201, VGG18
Final Dense Layer output: single linear layer, two linear layers with dropout and ReLU activation
Starting Learning rate: 0.001, 0.01, 1.
Criterion: Cross Entropy Loss, Negative Log Loss

The Convolutional layers for the pre-trained model were frozen and the output layer was a linear layer which took as input parameters from the previous layer and returned values for the 2 classes (COVID(index 1) or background(index 0)). The model was trained for a certain number of epochs. The parameters for the number of epochs that yielded the highest validation accuracy were stored and predictions were made based on those.

4.2 Keras

In this attempt, a CNN with the same ImageNet architecture as that described on slide 5 of lecture 35b was designed using Keras.

5 Results

Model	Kaggle Score
Transfer Learning Resnet152	*0.90322*
Keras	*0.85714*

6 Conclusion

6.1 Transfer Learning

For the transfer learning attempts described above, the most promising one was as follows:
The chosen pre-trained model was Resnet152. The criterion used was the Cross entropy loss and a stochastic gradient descent was performed starting at a learning rate of 1, momentum of 0.9 and decaying the learning rate by a factor of 10 after every 7 epochs. Subsequently, the model was trained for 40 epochs. With this implementation, we achieved a training accuracy of 96% and Validation accuracy of 85%. The score associated with that was of 90.3% reported on Kaggle.

We made the following observations when implementing the different models and hyper-parameters:

- A learning rate that started out too small did not lead to an optimal solution. There were very small changes on the training and validation error as the algorithm progressed through the epochs.
- A bigger initial learning rate with stochastic gradient descent and gradually decaying the learning rate started out with relatively high error and gradually became lower and converged to a better solution.
- Adding more layers to the dense output portion required more epochs to reach better accuracy but also lead to more overfitting. As a consideration for improvement, this issue could have been mitigated theoretically by using a form of regularization such as dropout.

- The data needed to be pre-processed in the same manner as Resnet152 images for it to yield better results. These models are sensitive to initial pre-processing.

6.2 Keras

At first, when training set was input into the model without normalization, the model was predicting all 1s for the lung images and didn't seem to learn to predict 0. To improve the model, input data was zero-meaned and normalized to [-0.5, 0.5]. Furthermore, batch normalization layers were added after each convolutional and fully connected layer. These steps are taken to ensure that input data at each layer is much less sensitive to small changes in weights and is easier to optimize. As a result, the model also started predicting 0s and the training error went down to 15.17% with 5 epochs and batch size of 10 (randomly sampled from the entire training set of 70 examples). However, interestingly, this model could not achieve a training error of less than 15% even after increasing the number of epochs and the batch size. No improvement was observed even after dividing the batch size into half covid samples and half non-covid samples. This suggests that perhaps the model was too complicated and a potentially lesser number of convolutional filters should be used per layer.

7 References

[1] Pointer, I. (2019). Programming PyTorch for deep learning: creating and deploying deep learning applications. Page128, Sebastopol, CA: OReilly Media, Inc.

[2] Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), 1345-1359.

[3] Gupta, D. (2017). Transfer learning and the art of using Pre-trained Models in Deep Learning. Retrieved from https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/