CS535 Big Data

DISCRIMINATING COVID-19 CHEST X-RAYS USING CNNS AND TRANSFER LEARNING ACROSS A DISTRIBUTED CLUSTER

Allison Chilton
Laksheen Mendis
Menuka Warushavithana
Rachit Dalal

Contents

List of Figures

List of Tables

Abstract

Neural Networks are a fast growing method to assist in many tasks previously unachievable by machines. However, the training time and the dataset sizes required can be significant obstacles to developing these systems. In this paper, we explore using transfer learning to cut down on time and data required to successfully train a classifier using a COVID-19 dataset.

1 Introduction

In order to create a classifier network, one must have a significant source of data to train the algorithm. However, such data is not always possible. In our study, we identified that there was shortage of data provided for COVID-19 Chest X-Rays. In the datasets available publicly, we found two datasets that were both less than 100 MB each, a relatively small amount of data. Training a neural network on this amount of data leads to poor results. However, there is another dataset with a large amount of chest X-Ray data. We hypothesized that training a network on this dataset would yield a network which is capable of recognizing features specific to lungs, and would therefore be able to perform better at the task of learning to discriminate COVID-19 images from non-COVID-19 images.

To facilitate this process, we used a variety of well known Big Data libraries and techniques. This paper will walk through the various methods, tools and frameworks used to accomplish the task. The rest of the paper is organized as follows: section 2 formulates the problem we are trying to solve, section 3 explains in detail the datasets, tools, algorithms, approaches we used in this study. In section 5, we present the findings our experiments and evaluate the results. Finally, in Section 6 we conclude the paper.

2 Problem Formulation

With the global outbreak of COVID-19 (or the Coronavirus disease), one of the most vital challenges medical professionals are faced with is effectively diagnosing the disease. The problem we are focusing on is, how can one identify COVID-19 cases using chest X-Rays? At the time of this writing, testing for COVID-19 is heavily underutilized in the United States because of lack of access to test kits. However, X-Ray machines are widely available at medical labs and X-Ray images are one of the most commonly used imaging techniques used to diagnose diseases associated with lungs. We are trying to determine if it is possible to identify a high degree of correlation between a chest X-Ray and specifically COVID-19 vs other types of illnesses or healthy lungs.

This is an interesting big data problem because of the amount of hyper parameters available for Deep Learning. Especially with Convolutional Neural Networks (CNNs), there are many parameters, such as stride, kernel depth, padding, and more as well as general architecture that performing a large search of these possibilities cannot be done in a timely manner on one machine. Additionally, the datasets for these models are very large, and take many resources to process them. As we are in the midst of the COVID-19 pandemic, the success of this experiment will be accommodating to medical professionals across the globe to quickly and efficiently identify patients who have contracted COVID-19 and proceed to treatment.

3 METHODOLOGY

3.1 Datasets

Three major datasets that were used in this project.

3.1.1 NIH Chest X-Ray Dataset

The first dataset is be a 42 GB Chest X-Ray dataset from the National Institute of Health, which was posted 2 years ago on Kaggle [1]. This comprises 112,120 X-Ray images with disease labels from 30,805 unique patients. There are 15 classes (14 diseases, and one for "No findings"). Images can be classified as "No. findings" or one or more disease classes:

- Atelectasis
- Consolidation
- Infiltration
- Pneumothorax
- Edema
- Emphysema
- Fibrosis
- Effusion
- Pneumonia
- Pleural Thickening
- Cardiomegaly
- Nodule
- Mass
- Hernia

3.1.2 COVID-19 Chest X-Rays

The second dataset consists of COVID-19 cases with chest X-ray or CT images. It contains COVID-19 cases as well as MERS, SARS, and ARDS. This particular dataset was posted on March 10, 2020 [2]. The size of the dataset is 80 MB at the time of our initial pull. Although both datasets contain data beside X-Rays, we used only X-Ray images for our analysis.

3.1.3 Secondary COVID-19 Chest X-Rays

Because the main validation and test sets we used for implementation were just a hold out set from the main COVID images, we decided to find another dataset consisting of different COVID-19 chest X-Rays [3]. We wanted to test our system's performance on a dataset that was completely different from the original dataset in case it was sensitive to preprocessing, or there were repeated patient images (with different angles) that spread across training set and test set, or otherwise anything else unique to the original dataset.

3.2 Algorithms, Techniques, and Models

3.2.1 Preprocessing and Exploratory Analysis

Reorganizing the images

The images in the original NIH Chest X-Ray dataset was not organized in a useful way. They were partitioned into a several directories. In order to feed this shuffled dataset we created a custom loader class while doing pre-processing. But, providing images from the respective folders of the 15 diseases category would help us to make robust, scalable system in distributed manner. Therefore, with the help of a CSV file which had detailed listing of the diseases suspected to be associated with each image, we reorganized the images by copying the images to a set of directories named after the disease. This made it more convenient to be accessed for the training tasks. Once images distributed into respective folders, we could perform certain transformation to improve images quality. After converting images to PyTorch Tensors, we applied vertical and horizontal flip for each batch, resized images into three different categories ($224px \times 224px$, $256px \times 256px$, $512px \times 512px$) by preserving aspect ratio. This followed by gray scaling image to get black and white images with improved quality. Finally, to deal with images intensity distribution we have performed normalized transformation to make distribution across the batch standard. Also, all the images are zero padded as well. NIH and COVID-19 dataset have chest x-rays which is spread across the entire frame because of which we cannot crop the image to preserve tiny details of the images. Data communication is one of the problems while dealing with Big Data Applications. Therefore, using if we could identify a way to improve disk utilization by by augmenting the data to improve performance we did all these transformation.

3.2.2 Deep Learning Model and Application

3.2.2.1 Convolutional Neural Network (CNN)

For training the Chest X-ray dataset, we used a Convolutional Neural Network (CNN), with 2 Convolutional Layers, 2 fully connected layers and 1 output layer. By using convolutional layers, we can condense an image. Some convolutions will change the image, in such a way that certain features in the image get emphasized. Hence, the Neural Network can extract features which distinguish image categories.

On the other hand, Max Pooling was also used during the forward pass. It is a way of compressing an image. When Max Pooling is combined with convolution, it is very powerful. Further, it will preserve the features that were highlighted by the convolution, while simultaneously reducing the size of the image.

The below list contains more details on the topology of the Deep Neural Network (DNN).

1. Conv1 → input channels=1, output channels=6, kernel size=(5, 5), stride=(1, 1)
2. Conv2 → input channels=6, output channels=12, kernel size=(5, 5), stride=(1, 1)
3. Fully Connected1 → input features=187500, output features=120, bias=True
4. Fully Connected2 → input features=120, output features=60, bias=True
5. Output → input features=60, output features=15, bias=True

First convolutional layer (Conv1) takes only 1 input channel because the images are transformed to Greyscale when loading. In this CNN, we're using 6, 5*5 filters/kernels, thus 6 outputs will be there from first convolutional layer. A Relu function is applied on the output and Max pooling is performed with a 2×2 kernel with a stride of 2. Hence, if we use a rescaled image of size 512×512 as the input to this layer, the output will be of size 254×254.

These 6 outputs will be the inputs to the next convolutional layer. Thus, in the second convolutional layer, we're using 2, 5×5 filters/kernels and there will 12 outputs from this layer. Again a Relu function is applied on the output and Max pooling is performed with a 2×2 kernel with a stride of 2. Therefore, the output of this layer will be 125×125.

Now, these 12 outputs of 125×125 needs to be flatten and passed into the linear fully connected layer. Hence, the size of the input would be $12 \times 125 \times 125 = 187500$. This layer will output 120 features. Then a Relu function is applied on the output.

Next, this output is again passed through another linear fully connected layer. Thus, it takes 120 input features and outputs 60 features. Then a Relu function is applied on this output. Finally, it will be sent through an output layer with only 15 features. The output layer is restricted to 15 layers because we have only 15 class labels for the images in the dataset.

Note: Although above functionality was explained using a rescaled image size of 512, we also experimented with a size of 224. In that scenario, input to the fully connected linear layer was $12 \times 53 \times 53$.

3.2.3 Standalone Method for Establishing a Results Baseline

To establish a baseline of comparison, we developed a model that was capable of discriminating the COVID cases without any prior transfer learning. This uses a ResNext model with randomized initial weights. This allowed us to determine how well the model performs without any of the new methods we tried. The dataloaders oversample the minority class so that the number of COVID cases and non COVID cases are near even. This prevents the model from over fitting on the distribution of data. We also transform the input images so that they are the same input tensor shape, as well as normalized to the appropriate range for the model. In the training set we randomly flip horizontally. We also randomly jittered the brightness of the image, as well as slight adjustments to the rotation of the image, to both increase our amount of pseudo-unique data and prevent overfitting to particular orientations.

We compare the result of this using ResNext topology as a starting point for training a new classifier, versus a pretrained ResNext model that we used to do transfer learning. As another point of comparison, we measured the performance on the COVID datasets both before and after fine-tuning the model parameters with hyper-parameter search.

3.2.4 Training CNN with NIH Chest X-Ray dataset

Method 1

As the first method, we trained the CNN with NIH Chest X-Ray dataset. The idea was to use the trained CNN for transfer learning of COVID-19 Chest X-Ray dataset. The results obtained can be found in Section 5.1. When training the model, both Stochastic Gradient Descent (SGD) and Adaptive Moment (ADAM) optimizers were used. However, it was observed that ADAMs performance was not better than SGD (Section 5.1.1). Hence, we continued our training by gradually increasing the number of epochs, only with SGD.

Furthermore, it was observed that there is a trade-off between loss and time for training. i.e. loss became lesser when resizing the images to 512, instead of 224. However, due to the increased size and the higher number of computations, training with 512 took more time.

A batch size of 1024 was used during training. This gets distributed among the number of machines (world size in torch.distributed).

Transfer Learning is a ML technique and it really helps when you have two different datasets where one dataset is very huge and another one is smaller one. We did the same thing by training the our model on NIH dataset and we applied transfer learning to smaller COVID-19 dataset. To do this, we froze the initial layer of the original model changed the output layer and applied SGD optimizer while training and also applied stepped learning rate decay by gamma = 0.1 at every 7 steps. Finally, we fine-tunned the top layers and train model with 50, 100 epochs. After this we loaded the test dataset and checked that accuracy of the model and we got 86.66% - 94.11% range based on the epochs range(50-150) we trained our model. The batch size for this algorithm was 32 and we trained this on CPU. We used single machine for this experiment as the dataset size is very small.

Method 2

For the second method, we used the previously discussed ResNext code implementation, except rather than using the pre-trained ResNext model, we used a slight modification of the code to transfer learn from the NIH trained model. The basic process is the same: first, the NIH model is loaded from a saved state dictionary, and the model is initialized with the final outputs of the NIH X-Ray model. Then the dataloaders are initialized by oversampling the input images to cause the COVID and non-COVID cases to be in a near 50/50 balance; they are resized to the input dimension of 224 x 224. The configuration functions replace the final Linear output layer of the NIH outputs to an output of only 2 channels, COVID or non-COVID. The model is then fine tuned for 80 epochs to transfer the learning of the previous NIH state to a new output of COVID and non-COVID.

3.2.5 Frameworks

In this section, we describe the frameworks that were used during this project.

3.2.5.1 PyTorch

PyTorch is an open-source machine learning library for Python, based on Torch, used for applications such as natural language processing and computer vision. It is primarily developed by Facebook's artificial-intelligence research group.

We planned to use PyTorch as the computing framework. We first considered both PyTorch and TensorFlow, as both of them are open-source. However, the reason why we decided to proceed with PyTorch is because of the way the two frameworks define computations graphs. Tensorflow creates a static graph, while PyTorch is operated on dynamic graphs, which means that in Tensorflow, the entire computation graph of the model has to be defined before running the machine learning model. But in PyTorch, it is possible to dynamically define the computations graph and it is much easier to setup the distributed environment. We have also tried Horovod initially but due to software and library dependencies we were getting a lot of errors. Thus, we dropped this idea.

3.2.5.2 PyTorch Distributed

The torch.distributed package provides PyTorch support and communication primitives for multiprocess parallelism across several computation nodes running on one or more machines. The class torch.nn.parallel.DistributedDataParallel() builds on this functionality to provide synchronous distributed training as a wrapper around any PyTorch model [4].

torch.distributed supports three backends, each with different capabilities. By default, the Gloo and NCCL backends are built and included in PyTorch distributed. However, we used Gloo (refer 3.2.5.3) in our implementation. Since we did not use GPU to train our model, there was not a special need to use NCCL.

During the training and testing of the CNN, we used some collective communication functions available in torch.distributed. They are torch.distributed.all_reduce() and torch.distributed.gather(). However, we later figured out, when DistributedDataParallel is used there is no need to explicitly synchronize the gradients. It was also a bit challenging when implementing the torch.distributed.gather().

3.2.5.3 Gloo

Gloo is a collective communications library introduced by Facebook [5]. It comes with a number of collective algorithms useful for machine learning applications which include a barrier, broadcast, and allreduce. We used Gloo as the backend in the process of distributing the learning tasks.

3.2.5.4 Tune

Overview

Tune [6], a library in the Ray distribution, is a hyperparameter tuning framework that is capable of running on one or many distributed machines. It has various implementations of search algorithms, such as HyperBand search and Population Based Training (Genetic Algorithms).

Usecase

For this project we utilized Tune to perform a population based hyper parameter search for our standalone COVID models and method 2 of transfer learning. The basic process for setting up the distributed mode of communication was as follows: a single node is designated the head node. Various other nodes are designated as worker nodes. On the head node, a message bus via a redis server is started. We wrote a script to log into all the worker machines and automatically run a file to connect to the redis message queue. In the code, you tell the Ray framework (parent to Tune) to initialize by connecting to the head node. We tried Tune in both the single machine mode and the distributed mode. When used in single machine mode, Tune just iterates through every individual trial individually until they are all complete. When used in distributed mode, it is much faster, because it runs many different trials in parallel. It was able to run 5 epochs of training for 50 different models in about 5 minutes on 10 worker nodes. In total, we trained 300 different models 5 epochs each to evaluate the best learning criteria, 150 for the ResNext, and 150 for the NIH X-Ray transfer learning.

Because we only used Tune for the standalone implementation, the model topology and configuration was pretty much already set from the previous trainings. The hyperparameters we tuned for these were primarily for configuring the optimizer. The specific hyperparameters we tuned were the following:

- Optimizer selection (Adam vs SGD)
- Learning Rate
- Learning Step Size
- Learning Rate Gamma
- Momentum

4 CONTRIBUTIONS

Member	**Conributions**
Alison Chilton	ResNext Model, Transfer Learning Method 2, RayTune Hyper Parameter Search, Report
Laksheen Mendis	Laksheen Mendis & CNN, Distributed training of CNN, Hyperparameter tuning, Evaluating CNN, Report
Menuka Warushavithana	Preprocessing, Scripting the distributed execution of learning tasks, Report
Rachit Dalal	Data-Preprocessing, Data-Cleaning, Custom DataLoader, CNN for Covid-19 Dataset , Training of CNN on Distributed Environment, Hyperparameter tunning for the model, Transfer Learning for validation and Identification of Covid-19 Images (Method 1), Report

Table 1: Contributions

5 Results and Evaluation

5.1 Evaluation of Convolutional Neural Network (CNN) on NIH Chest X-Rays dataset

This section contains the results obtained from the trained CNN, which was described in Section 3.2.2.

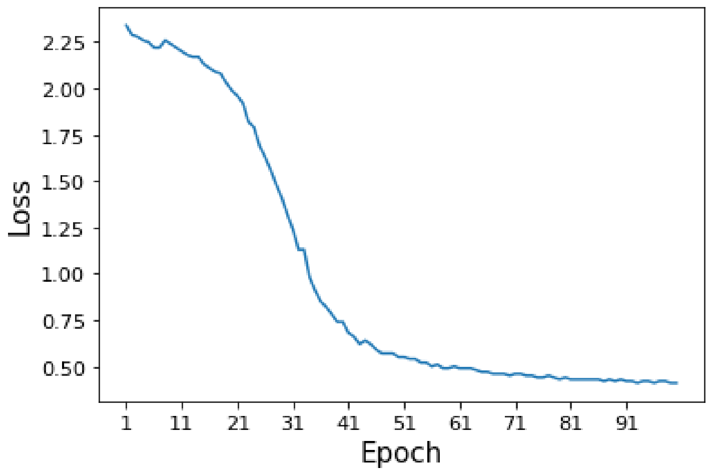


Figure 1: SGD-100 epochs, Learning Rate: 0.1

Figure 1 shows how the loss decreased from 2.34 to 0.41, over 100 epochs while training. This is the best model we could train due to huge training time involved. In this scenario, we used a size of 512 × 512 as the input to the convolutional layer by resizing the initial image size of 1024 × 1024. A learning rate of 0.1 was used.

On the other hand, For the another CNN we applied a resized transformation and converted to 224×224 image, trained over 150 epochs and the loss reached 0.52. It used a learning rate of 0.1 as well.

We also applied a resized transformation and converted to 256 × 256 image, trained over 50 epochs. But the loss was worst. It used a learning rate of 0.1 as well.

5.1.1 CNN with SGD and ADAM optimizers

As mentioned in Section 3.2.2, initially we used both SGD and ADAM as optimizers. Figure 3 and Figure 2 shows the loss against the epochs. when the learning rate was 0.1. At the end of 40 epochs, SGD resulted in a loss of 2.04 compared to ADAM, which was only 2.27. Hence, we decided to stop training with ADAM optimizer. However, to achieve much higher performance, there might be other parameters which should be used with ADAM. Due to the time constraints we were not able to experiment with such parameters.

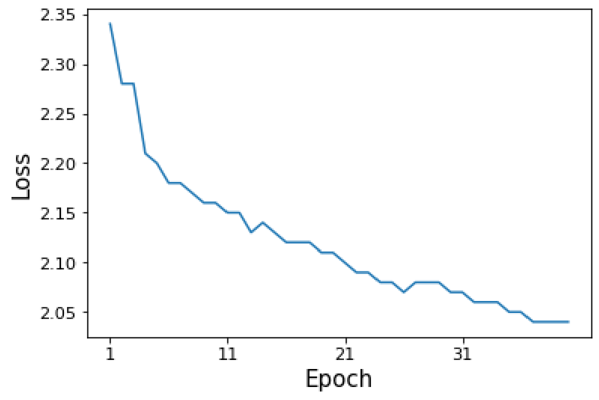


Figure 2: SGD-40 epochs, Learning Rate: 0.1

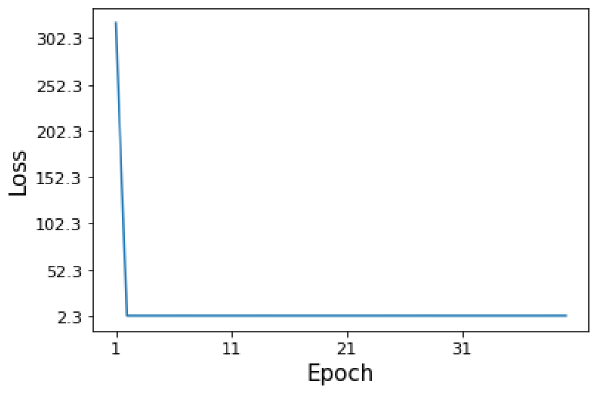


Figure 3: ADAM-40 epochs, Learning Rate: 0.1

5.1.2 CNN with SGD and 0.1 versus 0.05 learning rates

Below Figure 4 and Figure 2 represents the loss over 40 epochs. According to the plots, the best loss achieved was 2.07 and 2.04 for 0.05 and 0.1 learning rates respectively. Therefore, we selected 0.1 learning rate to train the model with SGD optimizer.

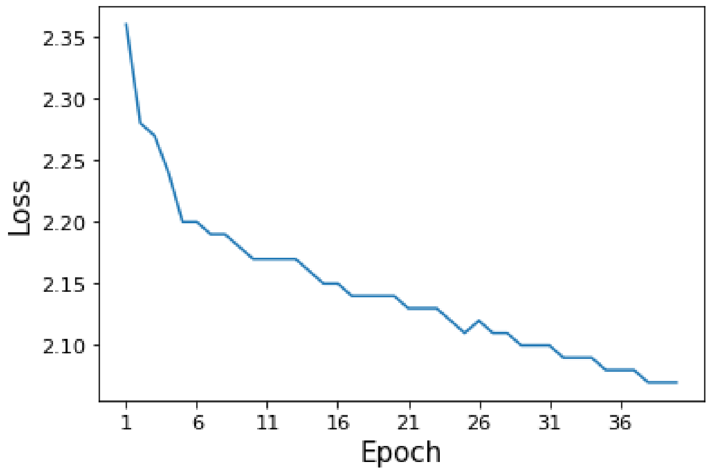


Figure 4: SGD-40, Learning Rate: 0.05

5.1.3 Performance of CNN on the test set of NIH Chest X-Rays dataset

The accuracy for the test set of NIH Chest X-Ray dataset was 14.48%. With more training we could have achieved a better accuracy. Since our primary goal was to use the trained model on COVID-19 Chest X-Ray dataset for transfer learning, and it gave successful results, along with the time constraints we decided to accept this as our best model.

Classification Report

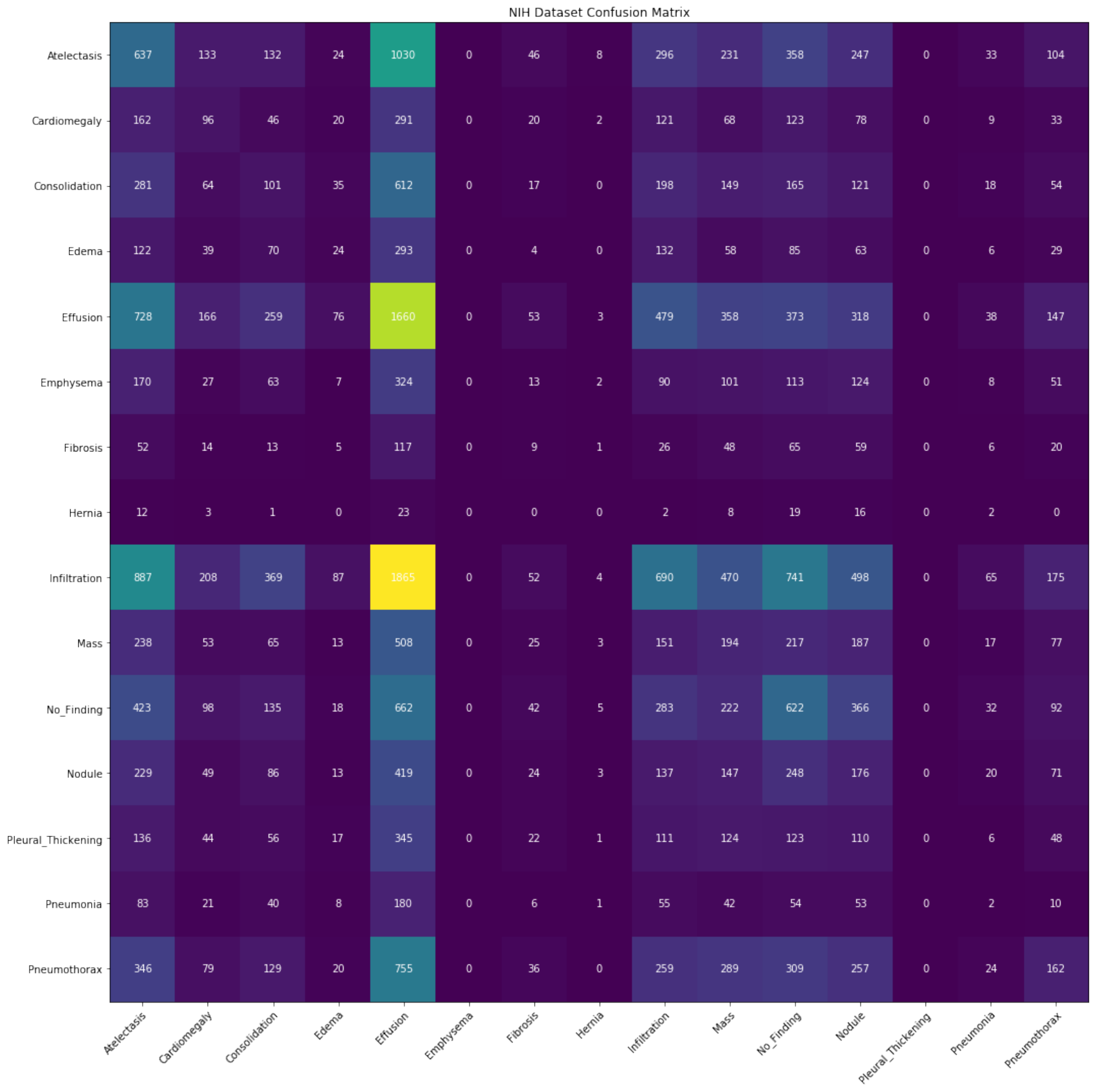
Below Table 2 represents Precision, Recall, F1-score and Support for all the classes in NIH Chest X-Ray dataset. We used sklearn.metrics.classification_report function to generate it.

	Precision	**Recall**	**F1-Score**	**Support**
Atelectasis	0.14	0.19	0.16	3279
Cardiomegaly	0.09	0.09	0.09	1069
Consolidation	0.06	0.06	0.06	1815
Edema	0.07	0.03	0.04	925
Effusion	0.18	0.36	0.24	4658
Emphysema	0.00	0.00	0.00	1093
Fibrosis	0.02	0.02	0.02	435
Hernia	0.00	0.00	0.00	86
Infiltration	0.23	0.11	0.15	6111
Mass	0.08	0.11	0.09	1748
No_Finding	0.17	0.21	0.19	3000
Nodule	0.07	0.11	0.08	1622
Pleural_Thickening	0.00	0.00	0.00	1143
Pneumonia	0.01	0.00	0.00	555
Pneumothorax	0.15	0.00	0.09	2665
Average/Total	0.14	0.14	0.13	30204

Table 2: Classification Report

Heat map of the confusion matrix

The confusion matrix is a performance measurement for ML classification problem where output can be two or more classes. In our case it is 15 × 15 confusion matrix which has 15 different classification including No Findings. Diagonal Entry represents the truly predicted values out of all the values in particular row whereas other values in the row showcase the false predictions made by models.



	Atelectasis	Cardiomegaly	Consolidation	Edema	Effusion	Emphysema	Fibrosis	Hernia	Infiltration	Mass	No_Finding	Nodule	Pleural_Thickening	Pneumonia	Pneumothorax
Atelectasis	637	133	132	24	1030	0	46	8	296	231	358	247	0	33	104
Cardiomegaly	162	96	46	20	291	0	20	2	121	68	123	78	0	9	33
Consolidation	281	64	101	35	612	0	17	0	198	149	165	121	0	18	54
Edema	122	39	70	24	293	0	4	0	132	58	85	63	0	6	29
Effusion	728	166	259	76	1660	0	53	3	479	358	373	318	0	38	147
Emphysema	170	27	63	7	324	0	13	2	90	101	113	124	0	8	51
Fibrosis	52	14	13	5	117	0	9	1	26	48	65	59	0	6	20
Hernia	12	3	1	0	23	0	0	0	2	8	19	16	0	2	0
Infiltration	887	208	369	87	1865	0	52	4	690	470	741	498	0	65	175
Mass	238	53	65	13	508	0	25	3	151	194	217	187	0	17	77
No_Finding	423	98	135	18	662	0	42	5	283	222	622	366	0	32	92
Nodule	229	49	86	13	419	0	24	3	137	147	248	176	0	20	71
Pleural_Thickening	136	44	56	17	345	0	22	1	111	124	123	110	0	6	48
Pneumonia	83	21	40	8	180	0	6	1	55	42	54	53	0	2	10
Pneumothorax	346	79	129	20	755	0	36	0	259	289	309	257	0	24	162

Figure 5: Heatmap of the Confusion Matrix

5.2 Results of the Transfer Learning on COVID-19 Dataset For the Binary Classification

Label	Test 1	Test 2
ResNext No Pretrain	84.44%	50.0%
ResNext Pretrain	95.56%	65.82%
ResNext Pretrain HyperParam	84.44%	50.63%
XRay Transfer Learn Method 1 - 50 Epochs	88.63%	89.96%
XRay Transfer Learn Method 1 - 100 Epochs	91.23%	94.11%
XRay Transfer Learn Method 2	82.22%	58.23%
XRay Transfer Learn HyperParam Method 2	86.67%	56.96%

Table 3: Model Accuracy

5.2.1 Losses for Different Model Trainings

For the sake of consistency every model was trained for 80 epochs. However, most models converged (or had a minimum) on the validation set in epoch 20-30. The ResNext models were much less susceptible to overfitting. Its possible that because the ResNext models were much larger topologies they did not overfit as much because there were more parameters to tune which would not cause wild swings in the validation set. It is a good loss to achieve so quickly. 10 epochs takes about 10 minutes on only a single CPU training for the ResNext models, and about 100 seconds for the NIH XRay Transfer Learning.

While we didn't use early stopping, it does use the best model according to validation loss. However, since the test set is small and uneven in distribution, if the model's predictions are biased in one way or the other, it may cause it to be more accurate than is truly reflective of a good model, causing an early epoch's model to be selected as the best result purely by chance, rather than a more mature and more true measure of success in a later model. Perhaps if there were more time, it would be worth sampling the secondary test sets loss over time, to see if the normal validation set is not representative of good model selection due to the uneven distribution of classes.

The main thing that could improve results is further fine tuning the learning rate and perhaps experimenting with even more optimizers so that the models don't converge so early. Additionally, while this goes beyond the scope of traditional transfer learning, its possible that defining the hyper parameter search to alter the topology of the networks could further improve results.

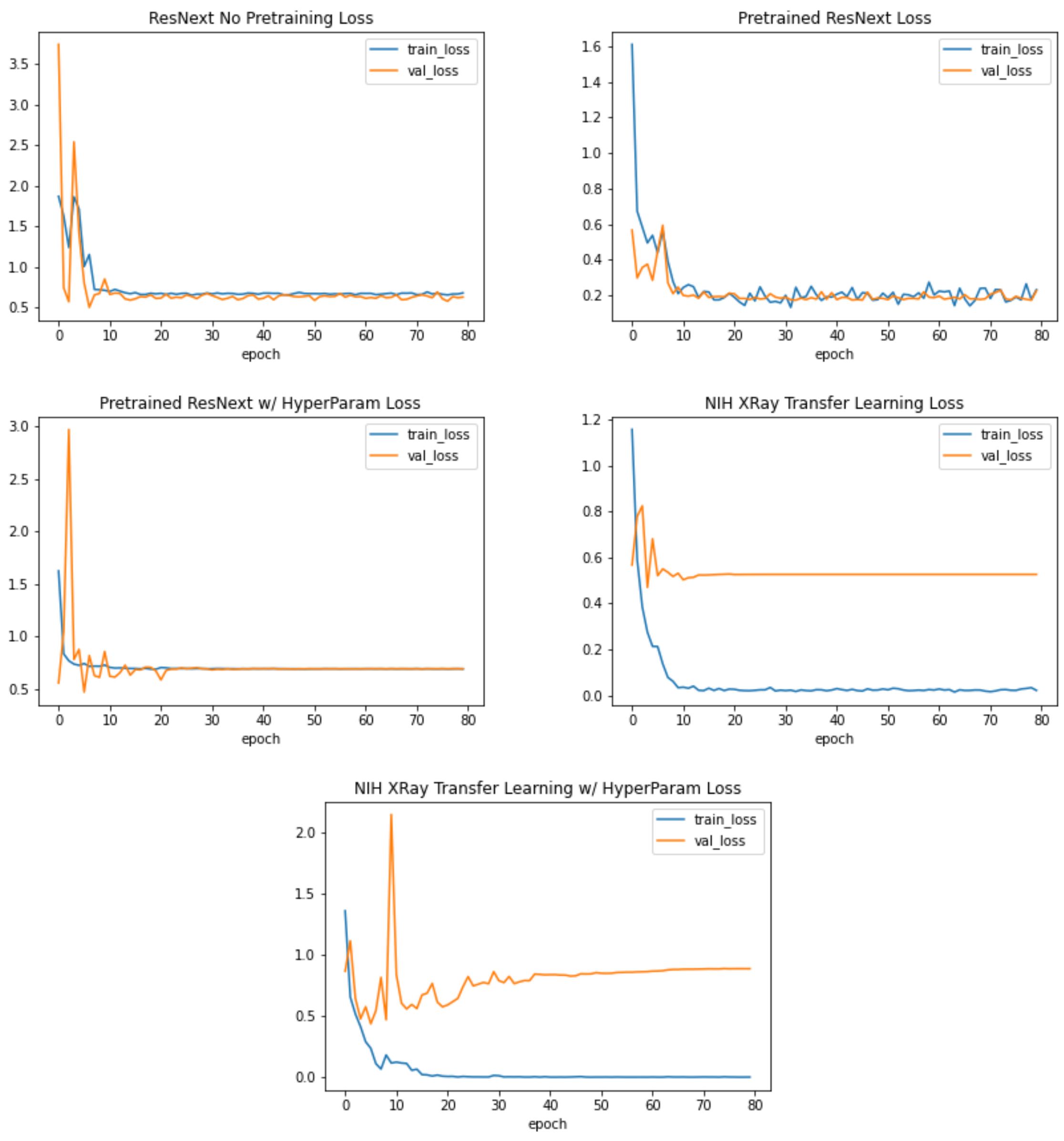


Figure 6: Training and Validation Losses for Standalone

5.2.2 Example COVID prediction labels

Below we have added a sample of prediction labels for the model built by method 1. To demonstrate one weakness of the system, we have also included a nonsense/noisy image to the set to demonstrate that the model will predict only within its binary choice of options. You can see that it classifies the cat and dog picture[**Figure:13**] as COVID. If there were more time, it may be more suitable to train the model for a third category, of Not Applicable or Unknown. Rest of the images[**Figure:7-12**] are classified correctly with almost 94.11% accuracy.

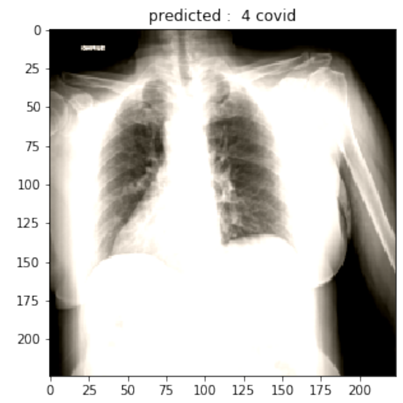


Figure 7: Actual - Covid Image

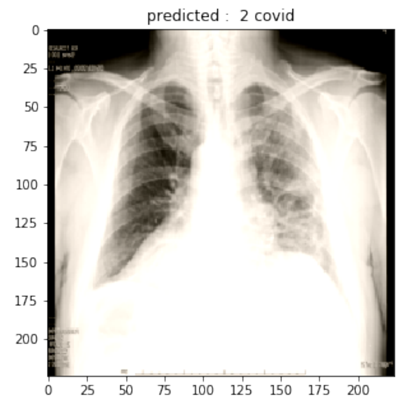


Figure 8: Actual - Covid Image

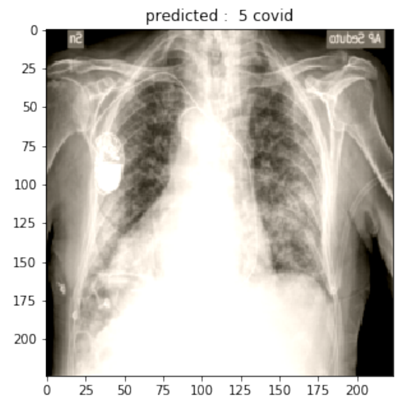


Figure 9: Actual - Covid Image

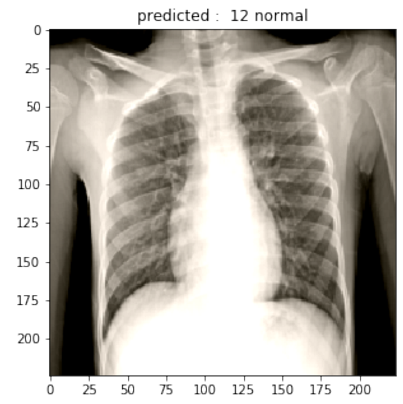


Figure 10: Actual Normal Image

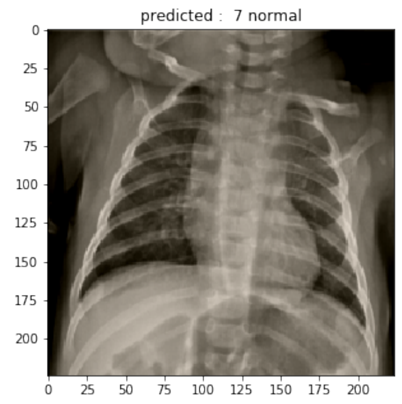


Figure 11: Actual Normal Image

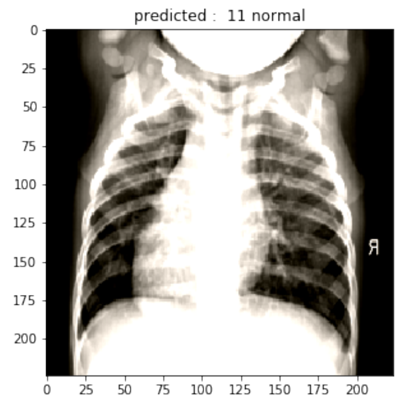


Figure 12: Actual Normal Image

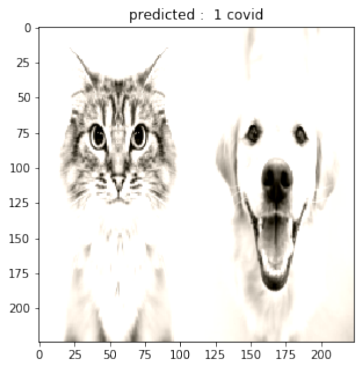


Figure 13: Non-classified

6 Conclusions

We sought to explore the usage of transfer learning and distributed techniques to demonstrate their effectiveness for Deep Learning in a Big Data environment. Our experiments show that there is indeed much value in distributed training in reductions on training time, hyper parameter search time, as well as useful for batch preprocessing of our input data.

However, our results showed that while transfer learning from a larger and more related dataset is indeed better than a naive approach (training the dataset with the minimal data available), it was not better than using a general CNN architecture that was pretrained with a very large dataset (such as ImageNet) and more hyperparameter optimization; in fact, our further hyper parameter search was still small enough at 750 total epochs that it actually made the model perform worse. It is likely that with the current state of the art networks such as the ResNext, the pretrained CNNs function as a general purpose feature extractors that can be applied to novel situations without needing explicit knowledge of the domain (such as radiological images).

The key distinction between Method 1 and Method 2 in Transfer Learning has been observed because of the best model selection process as well as the input image size. To select best model we used different combinations of the image size and epochs and chose best one. After training model on NIH dataset, to apply to smaller covid dataset for classification we froze initial/lower layers of the model and hyper tuned the top layers. This has resulted into best model out of all we tried[**section 5.2 : Table 3**] and we are able to achieve 94.11% accuracy with this setup.

This phenomenon may be unique to massively trained architectures such as CNNs and computer vision; there is much research and business case for developing highly trained models for this task. Our original idea was not totally without merit because it did outperform the naive approach, and domains without high amounts of research or capital may benefit from these techniques.

References

[1] N. I. of Health, “Nih chest x-rays,” Feb 2018. [Online]. Available: https://www.kaggle.com/nih-chest-xrays/data

[2] J. P. Cohen, P. Morrison, and L. Dao, “Covid-19 image data collection,” *arXiv 2003.11597*, 2020. [Online]. Available: https://github.com/ieee8023/covid-chestxray-dataset

[3] A. M. V. Dadario, “Covid-19 x rays.” [Online]. Available: https://www.kaggle.com/dsv/1019469

[4] “Distributed communication package - torch.distributed¶.” [Online]. Available: https://pytorch.org/docs/master/distributed.html#distributed-communication-package-torch-distributed

[5] Facebookincubator, “facebookincubator/gloo,” Apr 2020. [Online]. Available: https://github.com/facebookincubator/gloo

[6] “Tune: Scalable hyperparameter tuning.” [Online]. Available: https://docs.ray.io/en/latest/tune.html