Diagnosing COVID Infection Using Chest CT Scans and Deep Learning Models

W251 Final Project

Simon Li, Matthew Hui, David Yen, Jason Papale

Background

The ability to quickly and effectively diagnose COVID-19 in individuals is an important part of the fight against the current pandemic. Though RT-PCR testing is an accurate and effective testing method, there are a number of situations, such as instances where an individual requires expedited care due to rapidly worsening respiratory conditions, where making a COVID diagnosis using imaging is more ideal. Population-wide testing may not be feasible in developing countries due to high costs and more limited testing infrastructure. Chest X-rays and CT scans currently serve as a routine tool to diagnose pneumonia, while Chest CT has shown to have a high sensitivity for COVID-19 diagnosis (Ai et. al, 2020). Limited data during the onset of novel diseases also pose additional challenges that need to be addressed. To that end, our project entailed building and evaluating a series of deep learning models which predict a COVID-19 diagnosis based on a CT chest image, while also using GAN to augment our dataset.

COVID-19 Chest CT Dataset

The dataset that was used in this project consisted of 194,922 CT slices from 3,745 patients compiled across 9 different open source datasets. Each image was labelled, and there were three possible classifications: no disease, pneumonia, and COVID. The dataset was also reasonably well balanced, with ~48% of the images being labelled as having the COVID diagnosis, ~21% having the pneumonia diagnosis, and ~31% having no diagnosis. Of note, these classifications were based on a confirmed medical diagnosis (i.e., RT-PCR, radiologist-confirmed, etc.).

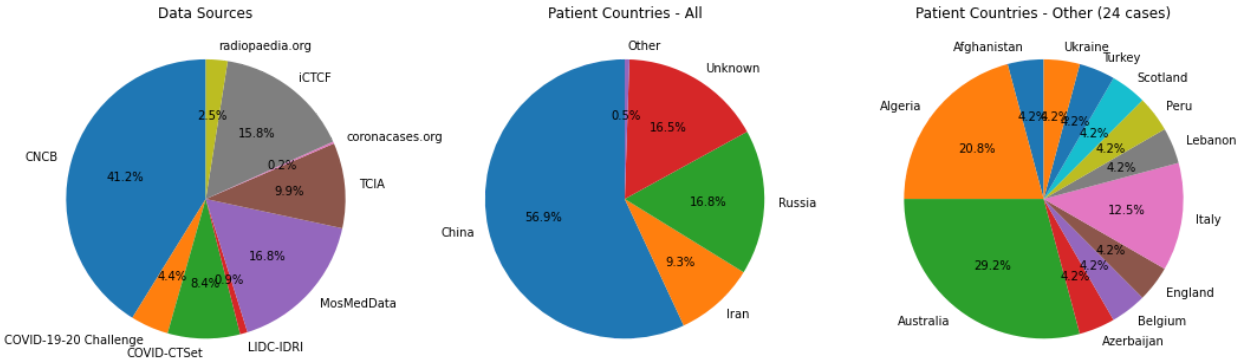

Figure 1: Distribution of images by data source and patient countries

Exploratory data analysis revealed that COVID cases have a statistically significant higher mean pixel value than Normal or Pneumonia. At the same time, class 1 Pneumonia had a higher fraction of brighter pixels (>250) than Normal and Covid. This finding was encouraging in the hopes the convolutional networks would be able to pick up this feature.

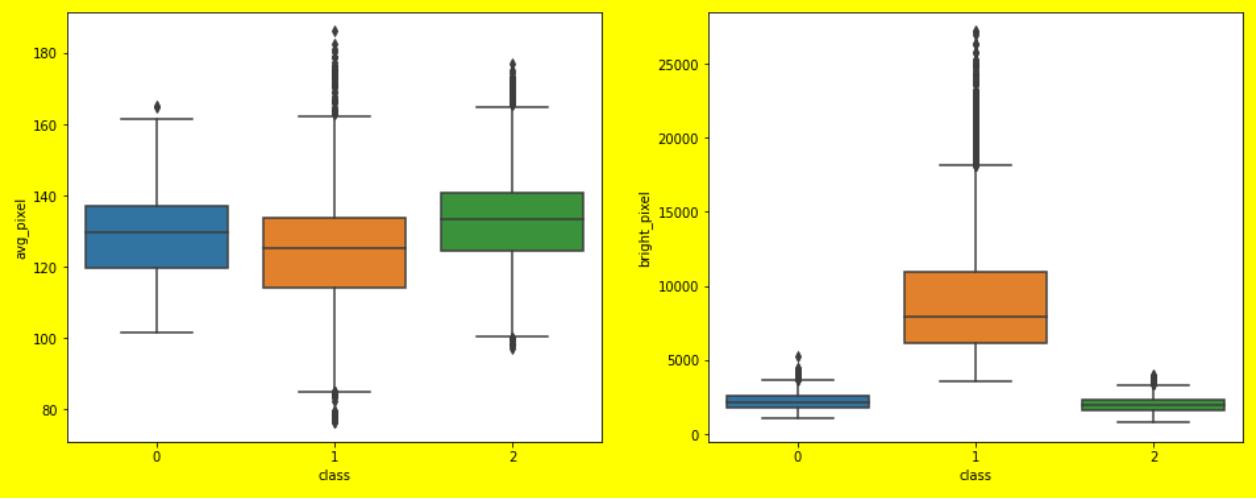

Figure 2: Distribution of average pixel value by class (left) and very bright pixels.

Approach

CNNs were chosen as the method of diagnosis using Chest CT scans, due to their popularity and documented performance on a variety of image classification tasks. A baseline CNN model was developed to gauge overall CNN performance and as a

reference point for other models. In addition, we leveraged two pre trained deep learning models, Squeezenet and Resnet-18, for transfer learning to understand the performance of popular models successful at other classification tasks. To address potential issues of data scarcity in a novel disease setting, we also attempted to use GANs to augment our dataset, as well as performed additional data augmentation during transfer learning such as image resizing, flipping, and cropping.

Baseline CNN

The first model which was developed to classify a disease given a CT image was a convolutional neural network model. The primary purpose of this model was to function as a baseline model against which it would be possible to compare the performance obtained from the more advanced, pre-trained models that were developed. It also allowed for a relatively quick assessment regarding the degree of predictive power that could be expected with a deep learning model given the CT images being assessed.

To determine the specific architecture to use in the model, model performance was evaluated using a subset of the training dataset and 72 combinations of hyperparameters, which included: kernel size, the number of feature maps generated by each convolutional layer, the number of fully connected layers, and the number of neurons in each fully connected layer. This evaluation was performed iteratively, where the results of each iteration consisted of two epochs of training and model performance was assessed against the validation dataset after each epoch of training. The results of those evaluations were then, in turn, captured and stored in a Pandas DataFrame. After completing all iterations, the DataFrame was sorted based on the lowest model loss obtained when evaluating the model on the validation dataset. The top ten rows from this DataFrame are shown below as Figure 3.

	HYPERPARAMETER SELECTION	EPOCH	KERNEL SIZES	NUMBER OF FILTERS	FIRST HIDDEN LAYER DIMENSIONS	SECOND HIDDEN LAYER DIMENSION	OPTIMIZER	AVERAGE TRAINING LOSS	VALIDATION LOSS	VALIDATION ACCURACY
2	2	0	[2, 3, 4]	[25, 25, 25]	200	50.0	Adam	0.832745	0.781840	65.779848
4	4	0	[2, 3, 4]	[25, 25, 25]	200	100.0	Adam	0.865472	0.849433	66.567064
0	0	0	[2, 3, 4]	[25, 25, 25]	200	NaN	Adam	1.645104	0.879331	62.911448
6	6	0	[2, 3, 4]	[25, 25, 25]	500	NaN	Adam	0.923998	0.881820	64.067895
8	8	0	[2, 3, 4]	[25, 25, 25]	500	50.0	Adam	0.900638	0.899858	57.777537
12	12	0	[2, 3, 4]	[50, 50, 50]	200	NaN	Adam	0.933332	0.922048	63.020585
10	10	0	[2, 3, 4]	[25, 25, 25]	500	100.0	Adam	0.926332	0.924838	56.479234
7	7	0	[2, 3, 4]	[25, 25, 25]	500	NaN	RMSprop	1.307246	0.937304	43.780282
3	3	0	[2, 3, 4]	[25, 25, 25]	200	50.0	RMSprop	3.687932	0.945084	64.993283
9	9	0	[2, 3, 4]	[25, 25, 25]	500	50.0	RMSprop	1.893101	0.955423	55.514554

Figure 3: Baseline CNN Hyperparameter Tuning Table

After performing the hyperparameter evaluation routine, the architecture of the baseline CNN was finalized and can be seen below in Figure 4. The first component of the model architecture consisted of 3 convolutional layers, each outputting 25 feature maps. The first convolutional layer utilized a 2x2 filter, and the second and third convolutional layers utilized filter sizes of 3x3 and 4x4, respectively. A ReLU activation function was utilized on the output of each convolutional layer. Following the activation function, a max pooling layer was then utilized to take the highest value from the resulting tensor. This resulted in a tensor with 25 values for each convolutional layer. Since there were three convolutional layers, three of these 25-element tensors were created at this step in the process. These tensors were then concatenated, resulting in a tensor of 75 values. This concatenated tensor was then fed to a fully connected layer with 200 neurons and then a fully connected layer with 50 neurons. Both of these fully connected layers utilized the ReLU activation function, as well. Finally the output of the second fully connected layer was fed a final fully connected layer with three neurons which used the Softmax activation function to facilitate the model's classification.

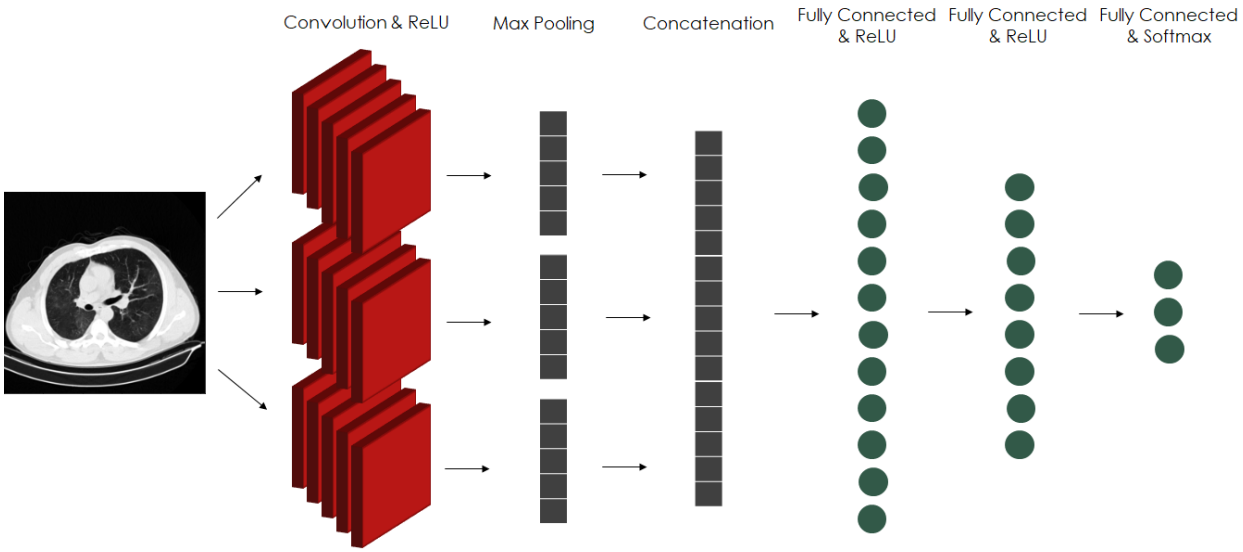

Figure 4: Baseline CNN Model Architecture

Once the model architecture for the Baseline CNN model was finalized, it was possible to train the model on the full training dataset. Initially, the model was only trained on 15 epochs; however, due to the fact that model performance was still steadily improving at the conclusion of training, this was expanded to 50 epochs of training. The training and validation loss, as well as the validation accuracy, observed over the course of training the model over the 50 epochs can be seen below in Figure 5.

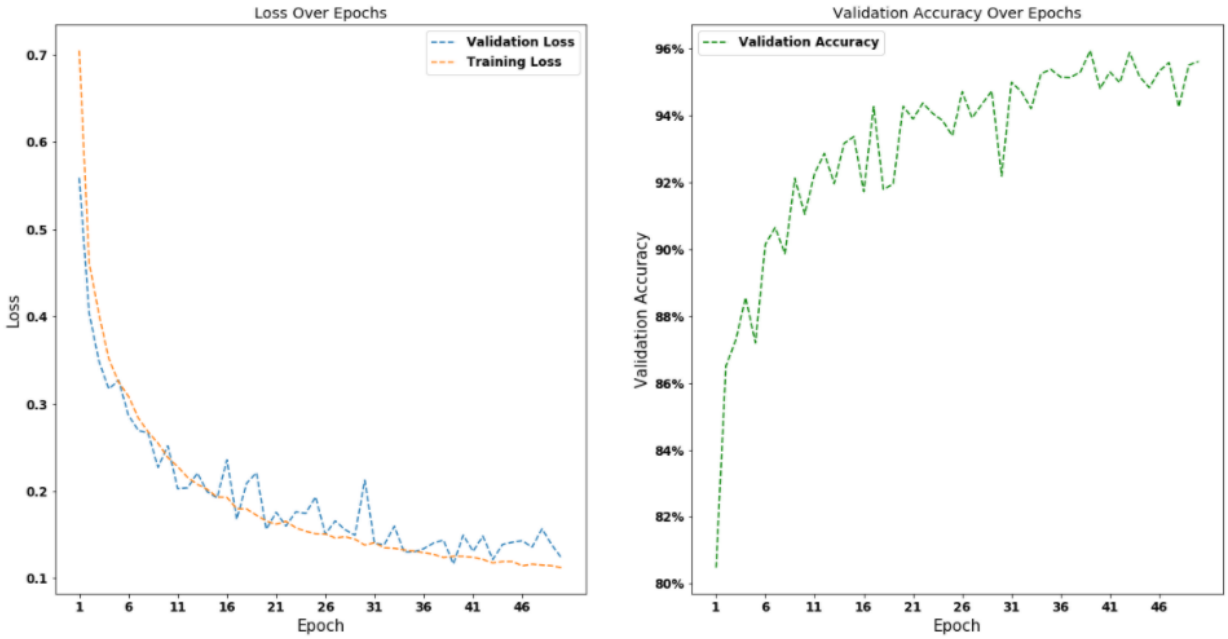

Figure 5: Loss and Accuracy Observed While Training Baseline CNN

As can be seen in Figure 5, the Baseline CNN achieved an accuracy of >95% following the 50 epochs of training. Additionally, as expected, both the training and validation loss values decreased approximately hyperbolically over the training epochs. It is also worth noting that both the validation accuracy and the training and validation loss values appeared to steady out after this number of training epochs. After fully training the model on the full training dataset for 50 epochs, the model was evaluated on the test dataset. The results of that testing can be seen in the classification report shown below as Figure 6. Ultimately, the model performed quite well, achieving an average F1 score of 96%.

	precision	recall	f1-score	support
0	0.91	0.98	0.94	6032
1	0.95	0.94	0.94	4027
2	1.00	0.95	0.97	9433
accuracy			0.96	19492
macro avg	0.95	0.96	0.95	19492
weighted avg	0.96	0.96	0.96	19492

Figure 6: Baseline CNN Model Performance on Test Dataset

The baseline CNN model was the first model to be developed as part of this effort, and it performed remarkably well despite only a modest degree of hyperparameter tuning and consisting of a relatively simple architecture. Moreover, additional components

such as batch normalization and dropout were not used but could have potentially improved model performance further. Ultimately, these additional measures were not explored, as the baseline CNN model achieved sufficiently satisfactory performance to fulfil its intended function of serving as a reasonably good model to predict disease classification given a CT scan and as a benchmark against which the more advanced pretrained models could be compared.

Transfer Learning

SqueezeNet

The first pretrained model selected was SqueezeNet. Developed as a compact substitute for AlexNet, it has 50x fewer parameters. Due to its smaller size, SqueezeNet was considered a good candidate due to easier implementation on an edge device with fewer resources. SqueezeNet also served as the basis for COVIDiagnosis-Net (Ucar, 2020), a deep learning model for COVID-19 diagnosis using X-rays.

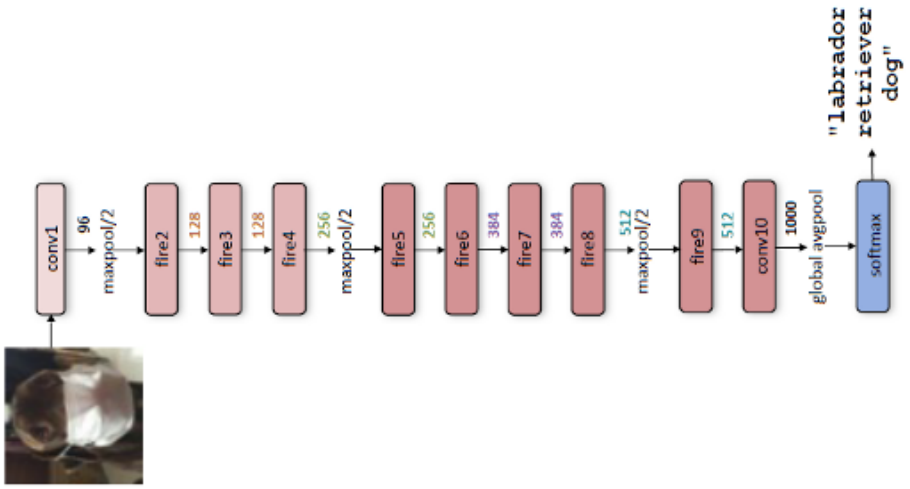

Figure 7: SqueezeNet Architecture

In Figure 7, we can see that SqueezeNet consists of fifteen layers with five different layers as two convolution layers, three max pooling layers, eight fire layers, one global average pooling layer, and one output layer softmax. The fire layer consists of a squeeze and an expand layer. The squeeze layer utilizes 1x1 filters, while the expand layer uses filter sizes of 1x1 and 3x3 (Figure 8).

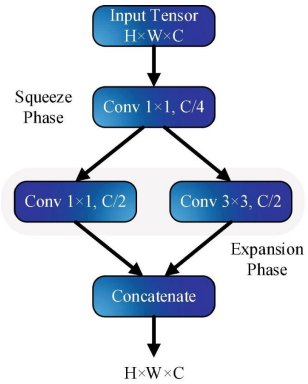

Figure 8: Structure of Fire Layer

Additional data transformations were performed on the data to prevent overfitting, including flipping, resizing, and cropping. Using the torchvision implementation of SqueezeNet on PyTorch, the model was trained on both the full training data set and a smaller subset of 10,000 images using stochastic gradient descent as the optimizer. Given the relatively large number of images available, the whole network was fine tuned to the new task of diagnosing COVID-19 with the pretrained weights used as the initial starting values. However, in cases where limited data is available such as the onset of a novel disease, a feature extractor and a classifier on top could be trained in lieu of the entire network.

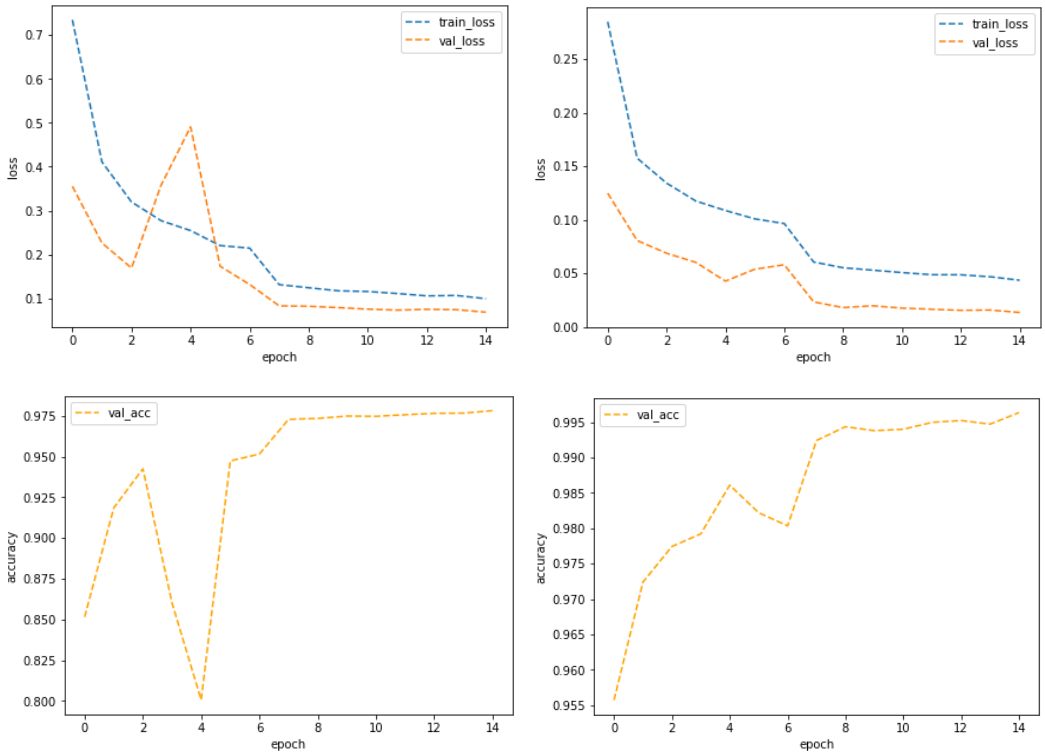

Figure 9: Loss and accuracy during training between smaller sample size 10,000 images (left) and full sample size (right).

In the case of both the reduced and full-size data sets, the model converged in 15 epochs. In the case of COVID-19 diagnosis, recall increased from 97.9% when training w/ 10,000 images to 99.6% when training with the full 130,000 image dataset. Thus SqueezeNet managed to be relatively robust in the case with the reduced training dataset.

	precision	recall	f1-score	support
Covid	0.985	0.979	0.982	9433
Normal	0.972	0.987	0.979	6031
Pneumonia	0.971	0.961	0.966	4027
accuracy			0.978	19491
macro avg	0.976	0.976	0.976	19491
weighted avg	0.978	0.978	0.978	19491

	precision	recall	f1-score	support
Covid	0.996	0.996	0.996	9432
Normal	0.995	0.997	0.996	6032
Pneumonia	0.996	0.993	0.995	4027
accuracy			0.996	19491
macro avg	0.996	0.995	0.996	19491
weighted avg	0.996	0.996	0.996	19491

Figure 10: Classification report on test set for SqueezeNet models trained on smaller dataset (left) and full dataset (right)

ResNet-18 CNN

A second model, ResNet-18, was applied in transfer learning to Covid-19 CT scans. ResNet-18 stands for residual neural network with 18 layers and applies skip connections during training to prevent exploding/vanishing gradients in deep nets. A pretrained ResNet-18 was a good candidate model for transfer learning given it has been trained on over one million images and over a thousand categories. An image of the model's architecture can be seen in Figure 11.

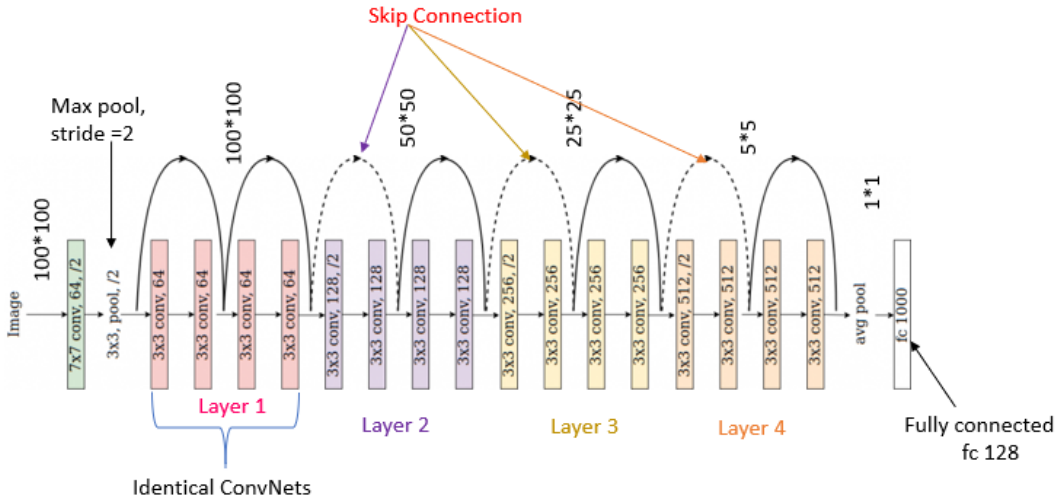

Figure 11. ResNet18 Architecture [5].

From Figure 11, ResNet18 contains three main parts: a Conv1 block (convolution + batch normalization + max pooling), four layers (each containing four blocks of convolution + batch normalization + identity shortcut or projection shortcut + ReLU), and the output block (average pooling + fully connected + softmax). The identity shortcuts (solids lines) or projection shortcuts (dotted lines) allow for propagation to skip layers.

Transformations were also applied to images fed to ResNet18. In training, images were randomly resized, cropped, horizontally flipped, and normalized. This type of data augmentation produced feature maps from different parts of the images and was found to work well in Inception Net [6]. In a brief study, we also found that without the transformations, the model would easily overfit. Thus the transformations added diversity, harder problems, and forced the model to focus on the important features in the images.

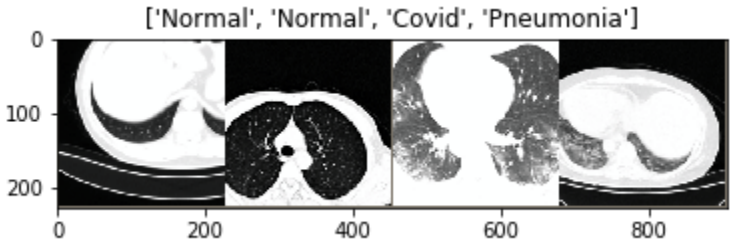

Figure 12. Example of data augmentation on images

For training, model weights were initialized with the pretrained model weights from ImageNet and then allowed to update on the Covid-19 CT scan dataset. Given the pretrained weights, two scenarios were explored: training on a partial training dataset of 10,000 images versus on the full 130,000 images. As seen in Figure XX, both scenarios showed convergence within 15 epochs. Though a higher validation accuracy of >99% was achieved when training the full dataset, an accuracy of >98.5% was achieved with the smaller dataset. The comparison demonstrates that transfer learning could be successfully applied to medical imaging classification even when there are a limited number of images.

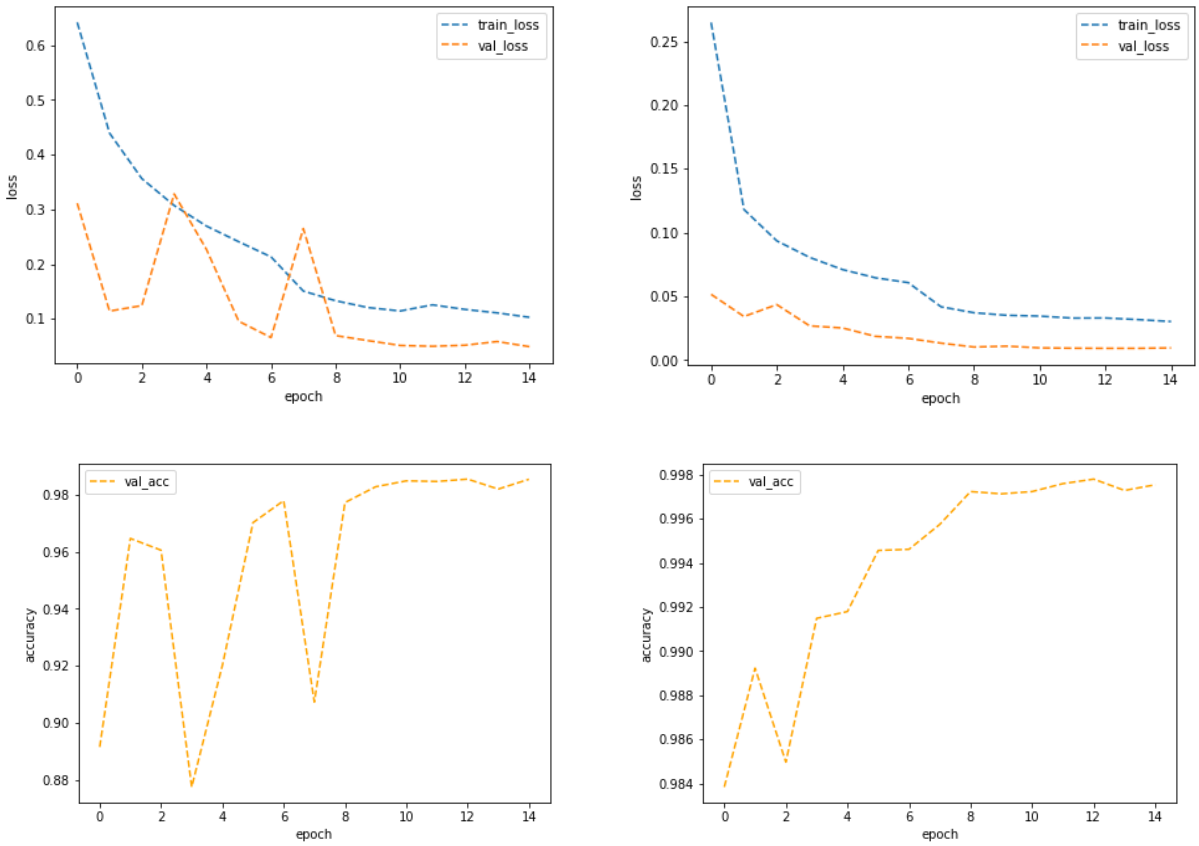

Figure 13. Loss and accuracy during training between smaller sample size 10,000 images (left) and full sample size (right).

Typically the validation loss curve is lower than the train loss curve but we saw an opposite effect as seen in Figure 13. The results seen in the loss curves could be a result of two events: when batch normalization and error calculation occurs. During training, batch normalization is applied and the mean and standard deviation may vary for each batch. Whereas in validation, the mean and standard deviation is of the entire batch. Secondly, validation error is calculated after the model has learned a complete epoch of training whereas training loss is calculated during training. These two effects could lead to a lower validation loss during training.

In evaluation, the both models were tested on the full test set and the classification report is shown in Figure 14. The recall was of particular importance given that patients who had Covid-19 should not be categorized as Normal and sent home. With the smaller dataset, the model achieved >98% recall. With the full dataset, >99% recall was achieved.

	precision	recall	f1-score	support
Covid	0.988	0.982	0.985	9335
Normal	0.985	0.989	0.987	6058
Pneumonia	0.971	0.977	0.974	4098
accuracy			0.983	19491
macro avg	0.981	0.983	0.982	19491
weighted avg	0.983	0.983	0.983	19491

	precision	recall	f1-score	support
Covid	0.997	0.998	0.997	9336
Normal	0.998	0.998	0.998	6057
Pneumonia	0.997	0.995	0.996	4098
accuracy			0.997	19491
macro avg	0.997	0.997	0.997	19491
weighted avg	0.997	0.997	0.997	19491

Figure 14. Classification report on test set for ResNet18 models trained on smaller dataset (left) and full dataset (right)

GAN - Generative Adversarial Network

Conditional GAN (cGAN)

To supplement the shortfall of certain classes (Normal, Pneumonia, or COVID-19) of CT images, we have chosen to use conditional GAN or cGAN to generate CT scan images of the specific class. Conditional GAN is just the regular GAN with an additional input of class label to both the generator and the discriminator as shown in Figure 15. This way the generator and discriminator can learn the special features of different classes through training and thus will be able to produce images specific to these classes.

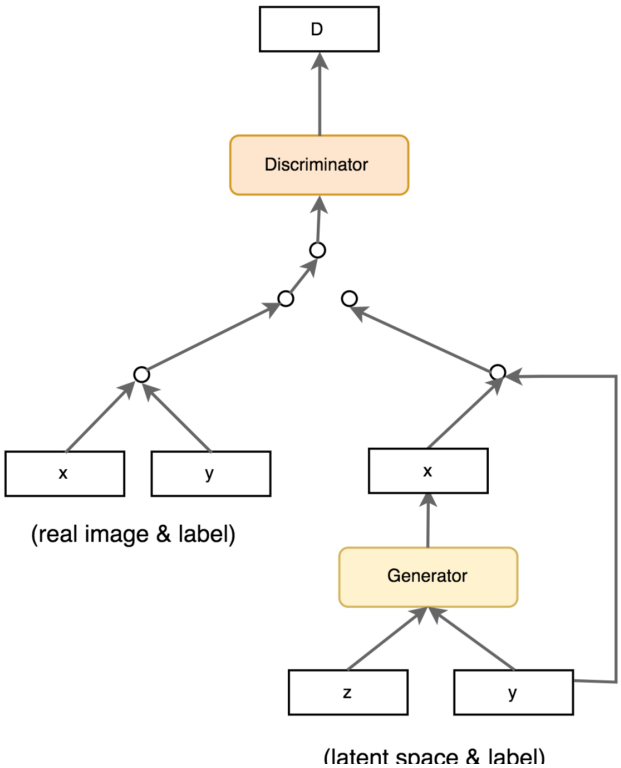

Figure 15. Overall architecture of a conditional generative adversarial network

Shallow cGAN Model

A baseline cGAN model was first developed based on a four times (4x) magnification factor to generate realistic images from a 7x7 foundational feature space to 28x28 resolution output images. A detailed diagram of the generator architecture can be seen in Figure A1 in the appendix. The generator model takes an input vector of 100-element latent dimensions and a class input. The key to make conditional GAN is to first convert the two types of inputs mathematically to the same 7x7 multi-dimensional array. As shown in Figure A1, the class label input is first embedded and reshaped into this same 7x7 representation as the latent dimension input so that the two inputs can be concatenated together before connecting to the downstream layer. The discriminator model has a similar architecture except arranged in reverse order; a real or a generated 28x28 image, a class label input similarly embedded (as the generator) and reshaped into a 28x28 array can be combined as a numpy array, then fed into the downstream network layers. The discriminator network mirrors the generator network and produces

7x7x128 representations before the output classification layer to determine whether the image is real or fake.

Deep-Convolutional cGAN model

A second GAN model is a Deep-Convolutional cGAN. This network is also called DCcGAN which is fashioned after the famous deep convolutional DCGAN architecture as shown below.

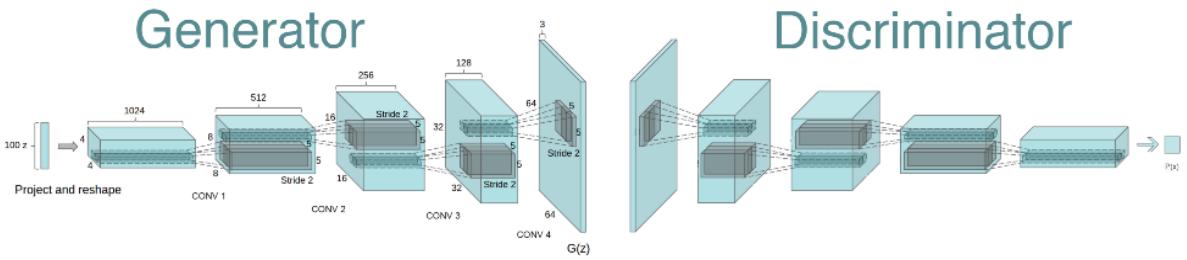

Figure 16. DCGAN architecture

The DCGAN network features multiple stages of upsizing/downsizing between layers by a factor of 2 using 5x5 kernel and stride 2 in the deconvolution or convolution operations. The original DCGAN has 4 deconvolution layers to boost the resolution from 4x4 to 64x64. In our case, we implemented a 5-stage enlargement from a feature map of 8x8x512 to 256x256x16 with a factor of 32 magnification before synthesizing all information together into a generated 256x256 grayscale CT scan image. We also used a symmetrical design on the discriminator side to reverse the process so the discriminator could take in a high resolution (256x256) CT scan (real or fake), process through the network and make a determination whether the image was real or fake at the end. Like the shallow cGAN network, we had the same input layer and out layer design for the DCcGAN. On the image side of the input layer, the generator took in a specification from the 100-element latent space and mapped that into a fully connected layer of neurons with sufficient numbers (in this case 8x8x512) to be reshaped into 512 low resolution (8x8) images to feed into the first deconvolution layer. On the label side, the class labels was embedded in 50 bins and reshaped into 8x8x1, then concatenated with the image side to form the final input of (8x8x513) to the rest of DCcGAN network. The complete model design for the generator is illustrated in Figure A2 in the appendix

along with the model designs of discriminator and GAN (combined generator and discriminator network) in subsequent figures.

GAN Results and Discussions

Setting up training of the GAN models are described here. First, we trained the discriminator with two half batches: one batch of real images and another batch of randomly generated images. Then the weights of the discriminator model were fixed. Training of the generator then followed by sending an entire batch of 128 images of generated images to the discriminator. A binary cross entropy loss function and ADAM optimizer were chosen for training both the generator and the discriminator models. The learning rate was 0.0002 and beta_1 was 0.5. As shown in the model plot described earlier, Leaky ReLU in all convolutional layers with the alpha parameter set at 0.2.

Difficulties were encountered during training in terms of convergence and image quality. We tried many so-called *GAN hacks* which are best practices and tricks to help balance the contention between the discriminator and the generator during training, which can be viewed as a game play between two players. The players can't learn much if the game is dominated by one side. Ideally, the two players should be comparable in skill and they have equal chance of winning. So we would expect a perfect outcome if the discriminator had an accuracy of 50% for both real and fake images when the generator is fully trained to be as powerful as the discriminator.

Figure 17 illustrates an example of successful training for our low-resolution model. The top graph shows the losses for discriminator and generator plotted over the number of batch runs on the x-axis. After a period of trials, the generator starts to generate images good enough to fool the discriminator sometimes and quickly settles to a low loss level for both the discriminator and the generator. The lower graph shows the accuracy for real and fake images. As the generator learns through the training process, the accuracies for both the real and fake images tend toward an equilibrium of accuracy of the ideal 0.5. In this case, we have about 0.7 which is satisfactory.

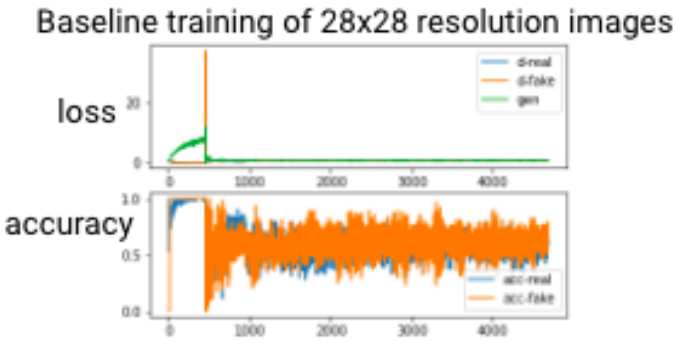

Figure 17. Baseline GANN with successful convergence during training

The training of our high-resolution model, however, was not quite satisfactory. A typical result is shown in the following. In this case, although there is still learning going on but the learning tends to be spiky and most spikes are followed by a low variation period throughout this training run of 100 epochs. In the lower graph, we see that the discriminator is able to consistently maintain an almost perfect (close to 100%) accuracy for both the real and fake images.

Figure 18. High resolution GAN training failed to converge

Below are sample images generated after 30 epochs and 100 epochs of training. The images at 30 epochs are certainly not perfect with many defects like opened chest borders and unreasonable shapes. The images generated after 100 epochs are more refined and more realistic but there are many artificial stripes as some literatures refer to as checkerboard artifacts superimposed on the generated images.

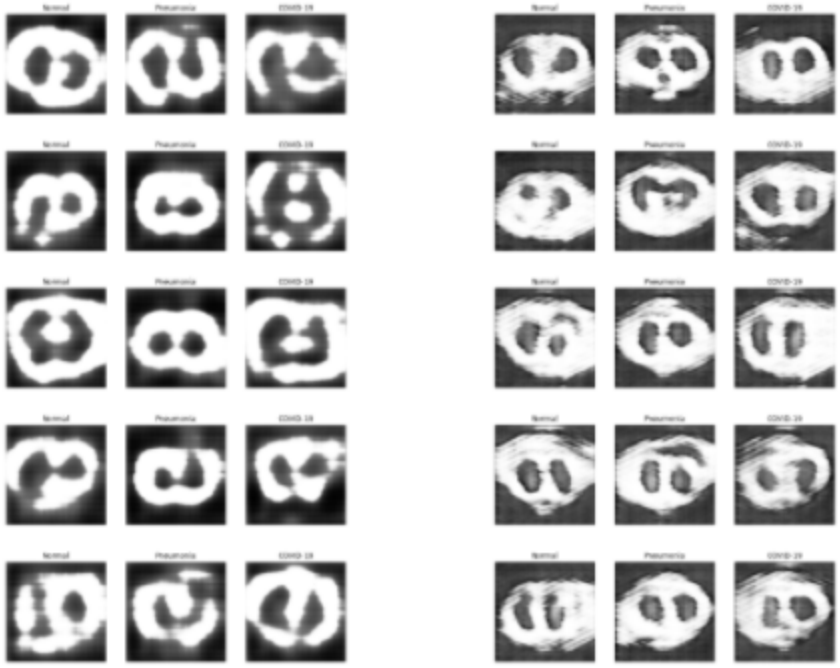

Figure 19. Generated CT scan images after 30 epochs (left) and 100 epochs (right)

Having tried many *GAN hacks* hat researchers have suggested in papers and blogs, a couple of them are shared below:

Best Practices Used in DCGAN & cGAN

- Normalize input to [-1, 1]
- Add a embedding layer to class label inputs to divide them into 50 discrete bins
- Use a latent_dim of 100 for the generator
- Factor of 2 downsizing and upsizing using 5x5 kernel/stride 2 between layers in conv/deconv
- Use Leaky ReLU with alpha=0.2 as activation function for all conv/deconv layers
- Use Tanh as the activation function for Generator out-layer
- Use Sigmoid as the activation function for Discriminator and GAN model out-layer
- Use ADAM for optimizer with .0002 for learning rate and 0.5 for beta_1 regulation
- Use binary cross entropy as the loss function for both discriminator and GAN models
- Use a dropout of 0.4

General Recommendable GAN Hacks

- Use large kernels
- Use more filters
- Use stride rather than pooling for scaling conv/deconv layers
- Batch normalization
- Wasserstein loss function
- Noisy/soft label
- Separate training for real and fake images
- Gaussian weight initialization

Unfortunately, results were limited given the challenges of convergence for a high quality image and time constraints of the project. As shown in the report however, the

dataset was rich enough that excellent results with convolutional networks were achievable without GAN augmented images. Additional work could be done by modifying or fine tuning the proposed DCcGAN model or trying out more advanced models like progressive GAN. The work on GAN presented above provides background for others who may be interested in this route.

Model Comparison

Here the three convolutional neural networks are compared based on evaluation metrics: precision, recall, and f1-score. In Figure 21, all three models perform well in precision and recall as shown by the curves reaching to the upper right corner. Being much deeper models and including data augmentations, SqueezeNet and ResNet18 outperforms our more shallow baseline CNN.

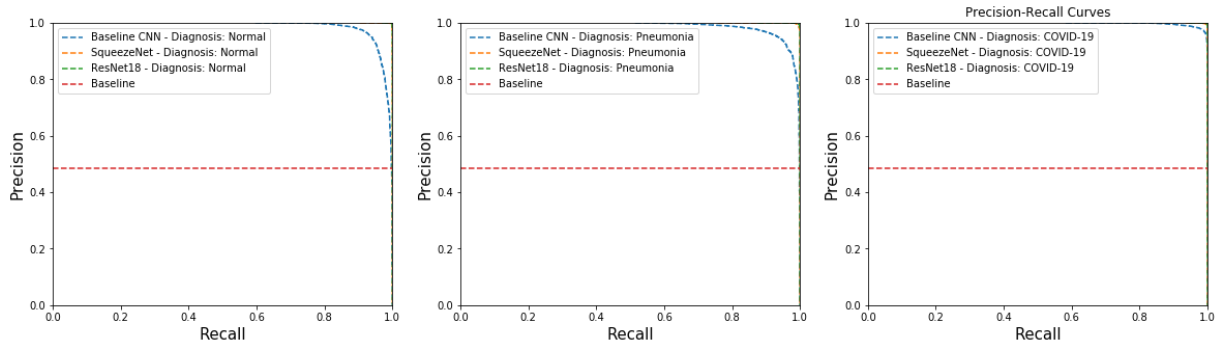

Figure 21. Precision-Recall Curve

In comparison between the two deeper convolutional neural networks, ResNet-18 slightly outperforms SqueezeNet as shown in Figure 22. The two networks differ quite a bit in their design so it is difficult to pinpoint which parts of the model may lead to one outperforming the other. Considering the number of parameters, SqueezNet performed quite well even with an order of magnitude fewer parameters than ResNet-18: 1.2 million versus 11.7 million respectively. In similar research, Lam showed that SqueezeNet and ResNet-18 were very comparable in recall and ResNet-18 outperformed in precision [13].

	precision	recall	f1-score	support
Covid	0.996	0.996	0.996	9432
Normal	0.995	0.997	0.996	6032
Pneumonia	0.996	0.993	0.995	4027
accuracy			0.996	19491
macro avg	0.996	0.995	0.996	19491
weighted avg	0.996	0.996	0.996	19491

	precision	recall	f1-score	support
Covid	0.997	0.998	0.997	9336
Normal	0.998	0.998	0.998	6057
Pneumonia	0.997	0.995	0.996	4098
accuracy			0.997	19491
macro avg	0.997	0.997	0.997	19491
weighted avg	0.997	0.997	0.997	19491

Figure 22. Comparison of metrics between SqueezeNet and ResNet-18

Deployment of Final Model to NX

Three convolutional neural networks described in the previous sections: baseline CNN, SqueezeNet, and ResNet-18 were trained on cloud and deployed to the edge utilizing the Nvidia Jetson Xavier NX and a webcam. The models were trained on an AWS EC2 p3.2xlarge GPU instance that had a Deep Learning AMI and then the model weights associated with the best performing model were saved. Additionally, a Python script containing the model frameworks for inference was also developed. Both the model weights and Python script were stored on GitHub, which could be readily version controlled and pulled into the NX.

To ensure that the environment on the NX contained all of the dependencies necessary to execute all of the above steps. Docker was used on the NX and the container was fitted with Nvidia L4T-ML, PyTorch, torchvision for pretrained models, and aws cli for communication with AWS S3. Within the docker container, the model weights and Python script were pulled from Github. To run inference on the NX, the Python script was executed and then directions were shown to the user to facilitate the image capture with the USB camera, image processing, model inference, and transfer of the image and diagnosis to the S3 bucket.

An example of the process wherein an image is captured, processed, and then fed to the model on the NX is shown below in Figure 20. From the left hand side, one of the actual images which was utilized in training and evaluation of the models is shown. In this case, the actual diagnosis associated with that image is COVID. As can seen below that image, the baseline CNN model correctly issued a diagnosis of COVID and did so with a >90% probability of classification. On the NX, a picture of that same image was captured using a webcam as seen in the middle image. To avoid confusing the model with the white border surrounding the image, a component of the Python script entailed automatically detecting the largest differences in pixel brightness, thus identifying the border of the image. It was then possible to crop the image to just content the CT scan. However, since this resulted in an image of arbitrary dimensions, it was then necessary to resize the image to 256x256, as these were the dimensions the model was prepared to accept. The image following this cropping and resizing can be seen on the right.

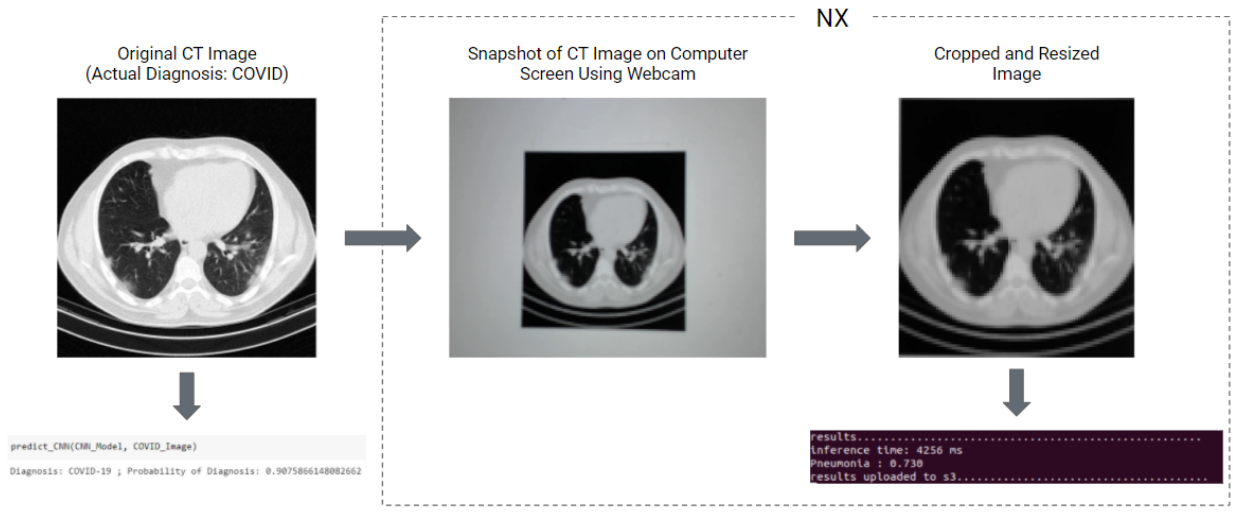

Figure 20: Image Capture and Model Inference on the NX

Unfortunately, when running this image through the model for inference, the model inaccurately issued a diagnosis of Pneumonia. Over the course of repeating this process many times, it became clear that the model generating inaccurate classifications given a picture obtained from the webcam was not atypical. The most likely reason for the model's subpar performance on the NX is due to image quality. Not only is the image captured from the webcam substantially less clear, but the brightness levels of the pixels are also not the same. It is very likely that were a higher fidelity imaging device used, the model would have performed better. In any event, after issuing the diagnosis, the Python script also sent the image and classification to an S3 bucket. It is also worth noting that the amount of time for the model to run inference was <5 seconds, indicating that this solution would be acceptable in most medical applications from a latency perspective.

Conclusion

Due to the immediacy with which various illnesses can be diagnosed with imaging devices such as a CT scanner, having a model which facilitates those diagnoses may make this method of illness classification more optimal than other methods, such as PCR testing, in some cases. In order to evaluate the feasibility of this method in the context of COVID, we evaluated the effectiveness of three different deep learning models. The first model was a baseline convolutional neural network constructed from scratch in Pytorch. The other two models, SqueezeNet and ResNet-18, were pre-trained models which we fine-tuned on the COVID-specific dataset. All three models performed exceptionally well; with the ResNet-18 performing the best with an

F1-score of >99%. We also attempted to use a GAN to generate images for scenarios where images of a new disease could be limited. Ultimately, we were unable to generate images of sufficient quality due to convergence and checkerboard artifact issues. Finally, we deployed the best performing model to the edge using the Jetson Xavier NX to conduct inference. Though the model did not generally predict accurately on the NX, we assess this is largely due to our imaging device, which was a webcam. We're confident that with an adequate imaging device, such as an actual CT scanner, a radiologist would be able to run this model on an edge device and obtain accurate diagnoses.

References

1. DS. Kermany, M., A. Jiménez-Sánchez, S., J. Ding, B., M. Frid-Adar, I., H-C. Shin, H., J. Beutel, H., . . . Tomas. Iesmantas, R. 1970. Deep convolutional neural network based medical image classification for disease diagnosis. Retrieved March 03, 2021, from https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0276-2

2. Gunraj, H., Wang, L., & Wong, A. 2020. Covidnet-ct: A tailored deep convolutional neural network design for detection of covid-19 cases from chest ct images. Retrieved March 03, 2021, from https://doi.org/10.3389/fmed.2020.608525

3. Lan, H., Initiative, T., Toga, A., & Sepehrband, F. 2020. Sc-gan: 3d self-attention conditional gan with spectral normalization for multi-modal neuroimaging synthesis. Retrieved March 03, 2021, from https://www.biorxiv.org/content/10.1101/2020.06.09.143297v1

4. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. 2015. Rethinking the Inception architecture for computer vision. Retrieved March 03, 2021, from https://arxiv.org/abs/1512.00567

5. Singhal, G. 2020. Transfer Learning with ResNet18 in PyTorch. https://www.pluralsight.com/guides/introduction-to-resnet

6. Szegedy, C .et.al. 2014. Going Deeper with Convolutions: https://arxiv.org/pdf/1409.4842.pdf

7. Mirza, M. and Osindero, S., 2014. Conditional generative adversarial nets, arXiv:1411.1784

8. Radford, A., Metz, L. and Chintala, S., 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, arXiv:1511.06434v2

9. Chintala, S.,, Denton, E., Arjovsky, M, and Mathieu, M., 2016. ganhacks: How to Train a GAN? Tips and tricks to make GANs work, https://github.com/soumith/ganhacks

10. Karras, T, Aila, T., Laine, S., and Lehtinen, J., 2018. Progressive Growing of GANs for Improved Quality, Stability and Variation, ICLR 2018

11. Arjovsky, M, Bottou, L., 2017. Towards Principled Methods for Training Generative Adversarial Networks, ICLR 2017

12. Teramoto, A, et.al., 2020. Deep learning approach to classification of lung cytological images: Two-step training using actual and synthesized images by progressive growing of generative adversarial networks, 10.1371/journal.pone.0229951

13. Pham, T. 2020. A comprehensive study on classification of COVID-19 on computed topography with pretrained convolutional neural networks. https://www.nature.com/articles/s41598-020-74164-z.pdf?origin=ppub

14. Li, S., Papale, J., Yen, D., Hui, M., 2021. Github Repository. https://github.com/sli0111/MIDS-251-2021-Final-Project

Appendix

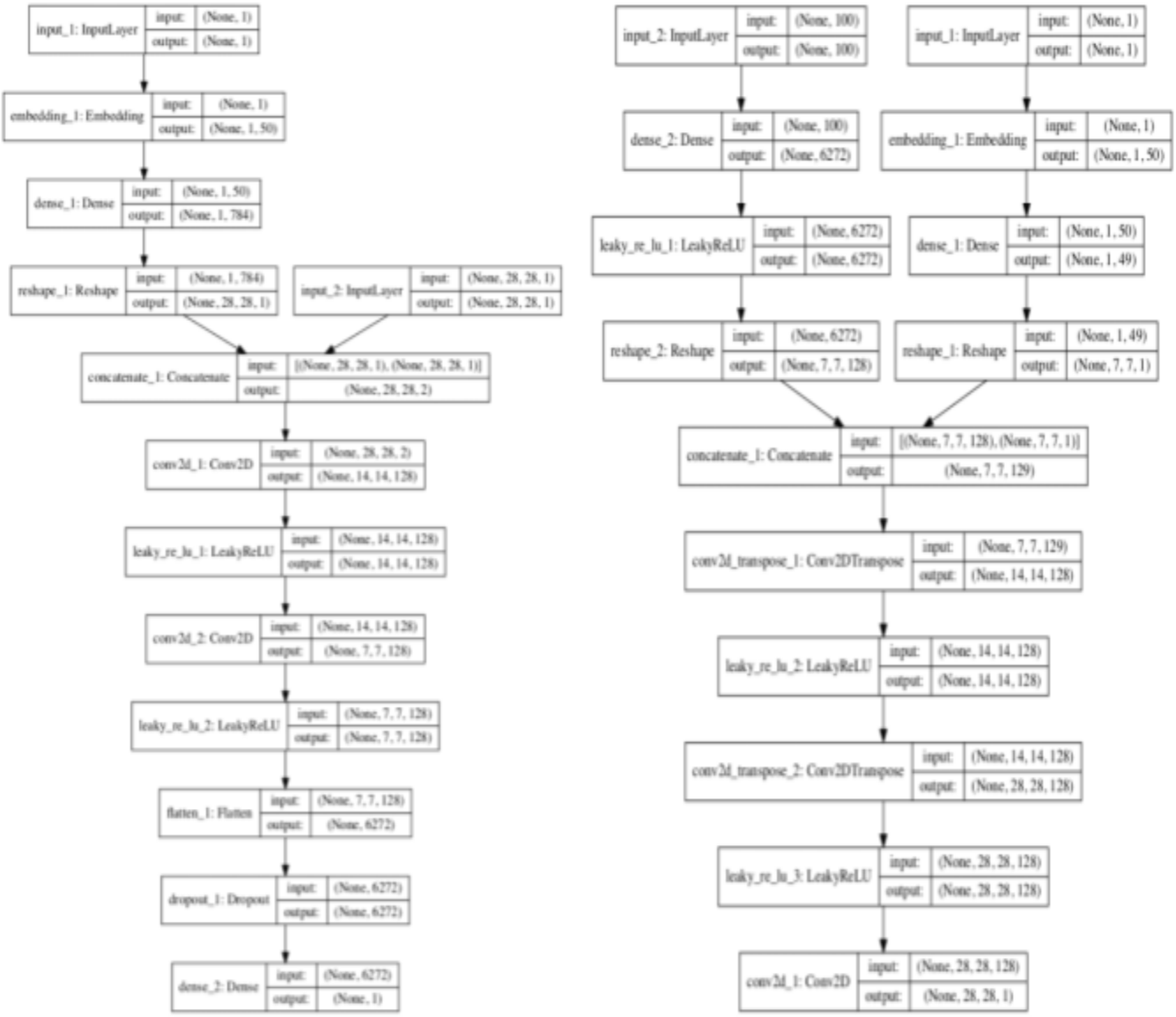

Figure A1: Model plots of discriminator and generator for the shallow GAN

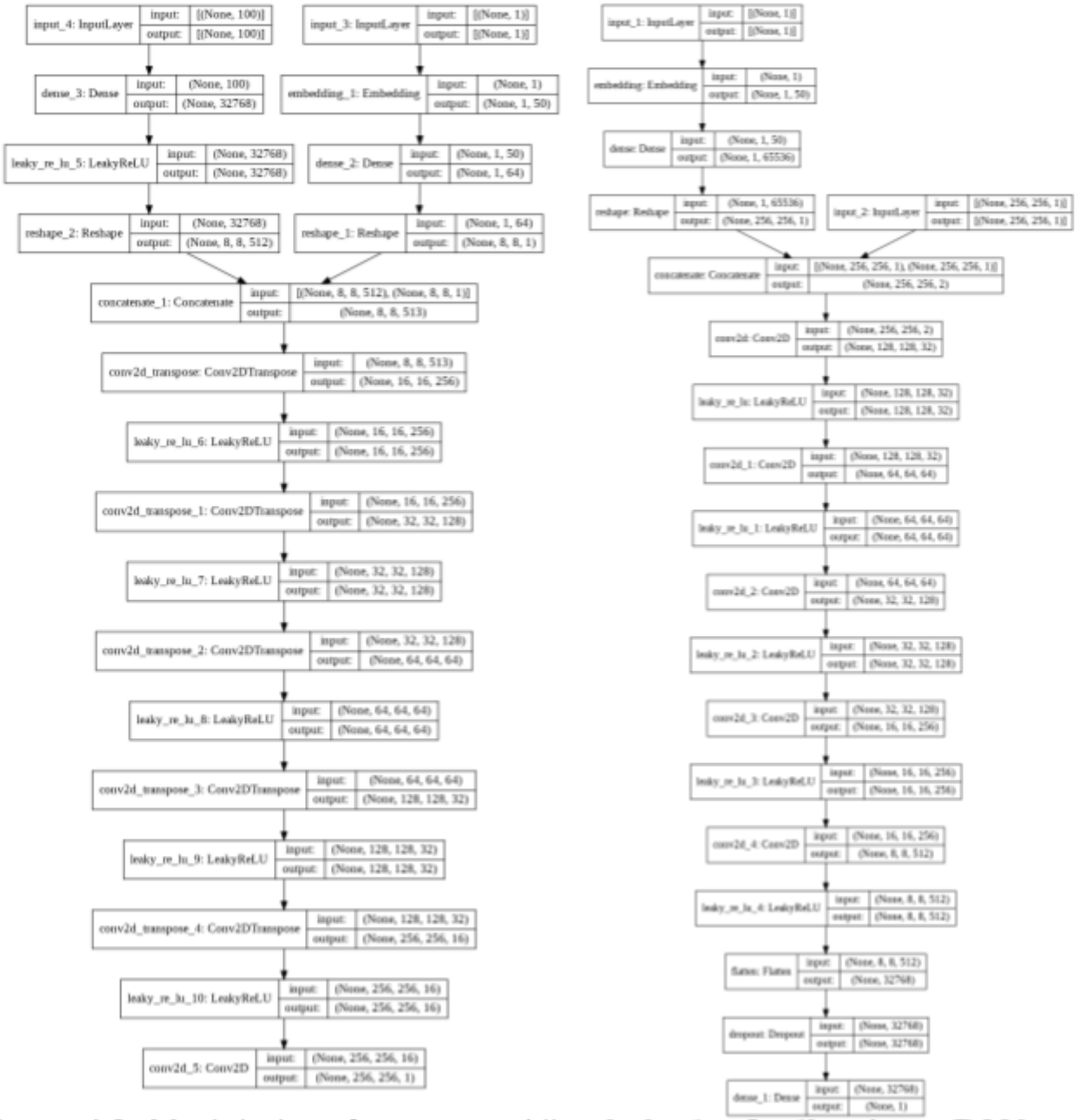

Figure A2: Model plot of generator/discriminator for the deep GAN