DATS - 6203 Machine Learning 2

Pediatric Pneumonia Detection with CNN

Automating Medical Diagnoses

Contents

1. Introduction

Pediatric pneumonia is consistently estimated to be the leading cause of childhood mortality (Rudan et al 2008). It kills more than malaria, HIV/AIDS, and measles combined (Adegbola, 2012). The two most common causes of pneumonia are bacteria and viruses (Mcluckie, 2009), but treatment methods differ significantly between the two. While viral pneumonia has no good treatment (cases often get better on their own), bacterial pneumonia often requires administration of antibiotics.

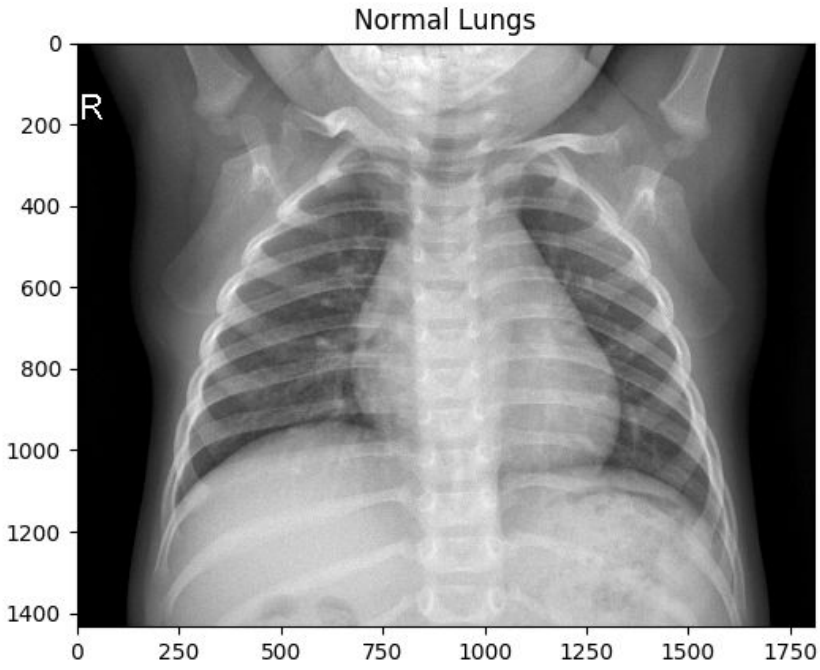
Because of the complications from incorrect diagnoses (false positive diagnoses of pneumonia can lead to wasting hospital resources, missing another potentially severe diagnosis; confusing bacterial pneumonia for viral pneumonia will lead to no treatment, and puts the child at risk for death) and the prevalence of pneumonia cases (X cases in Y year), robust and quick diagnosis is essential for the pediatric medical community. Also, in many areas where pediatric pneumonia is prevalent (i.e. Southeast Asia, Africa), rapid radiologic interpretation of images is not always available (Kermany et al. 2018). This is where AI can make the difference by automating pneumonia classification. Distinguishing viral pneumonia, bacterial pneumonia, and normal lungs is critical to improving the standard of care for hospitalized children and reducing childhood mortality rates globally.

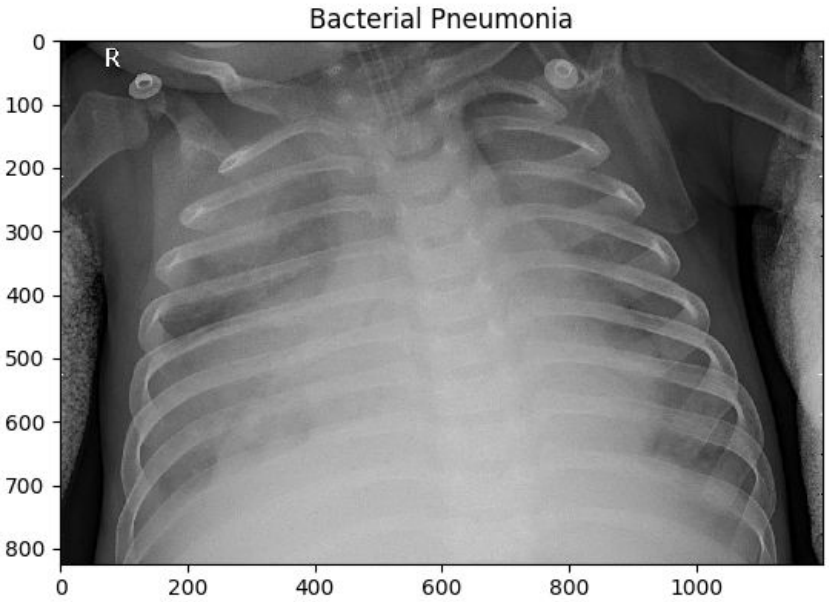
For our project, we attempt to develop an AI to do just that. The problem is fairly complex, as the division between viral and bacterial pneumonia proves difficult to model. However by experimenting with different algorithms and using transfer learning, we present a model that can solve this problem with reasonable (although improvements should be made) robustness. The breakdown of the project is as follows; Section 2 describes the data used to solve this problem, Section 3 discusses the preparation of the data for modeling, Section 4 details models we built to address this problem, the motivations behind them, and the results of those models and in Sections 5 and 6 we discuss our results and identify key areas where we can improve our modeling.

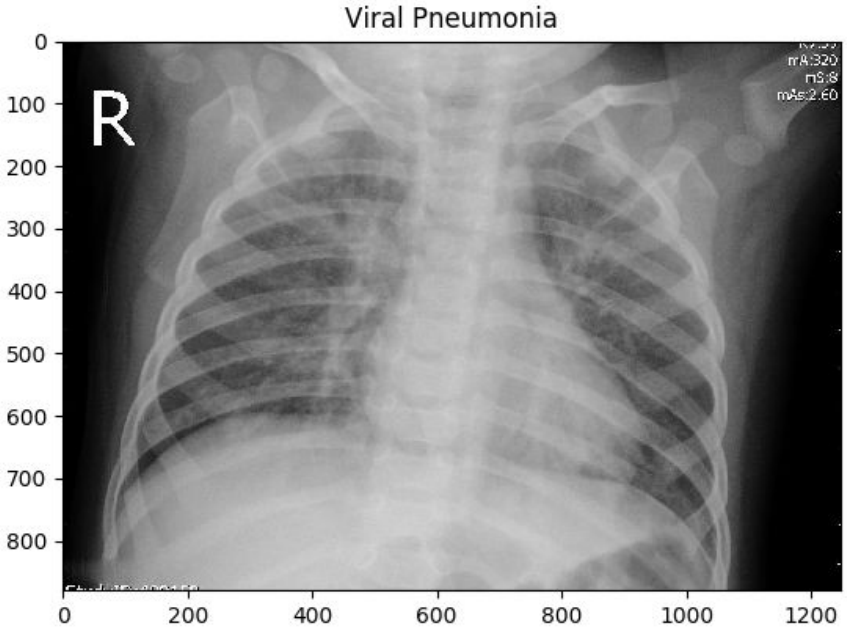
2. Data

To address the problem, a dataset was obtained from Kaggle (https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia) that contained 5,863 chest X-Rays of pediatric patients aged 1-5 years old from Guangzhou Women and Children's Medical Center in Guangzhou. These images are split into a train set (5,216 images), a validation set (16 images) and a test set (624 images). The images consist of 3 channels of the same pixels, approximately (1200 x 1800 pixels) in size.

Below are example X-Rays from each class:




Bacterial Pneumonia
R
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000


Viral Pneumonia
R
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200

In each, we see the lungs on the right side of the spine in the image, with the ribcage surrounding them. The images also have a letter in the corner; “R”, which indicates that the indicated side is the right side.

The breakdown of the data given in model-accessible folders (training and validation) is shown below:

Original Dataset Breakdown

Folder	Class	Number
train	Normal	1341
	Bacterial Pneumonia	2530
	Viral Pneumonia	1345
validation	Normal	8
	Bacterial Pneumonia	8
	Viral Pneumonia	0

Clearly, the dataset is imbalanced, and the validation set is both not large enough to offer real insight to how well our model is doing and does not contain any viral pneumonia samples. We discuss how we deal with this in the next section.

Problem Statement

The original dataset was intended to perform binary classification on the dataset that classifies the images into two categories: Normal & Pneumonia. Majority of the kernels that we went through on Kaggle have chosen to tackle this as a binary classification problem.

This project deals with classifying the data into three categories: Normal, Viral Pneumonia, Bacterial Pneumonia.

3. Model Development

Framework

We use PyTorch as our chosen framework, due to the fact that the group is most familiar with building computer vision algorithms in PyTorch. To make loading the data as simple as possible, we reorganize the data into a format that can be loaded with PyTorch's ImageFolder class. ImageFolder will infer the targets based on the folder names at load time. However, the directory structure of the data is set up to classify Normal vs Pneumonia, and does not distinguish between bacterial and viral pneumonia. To tackle our modified problem statement, we move the data into 3 respective folders; Normal, Pneumonia-Bacterial and Pneumonia-Viral.

Data Preprocessing

To ensure our validation set will give good information on how well our model will generalize, we must deal with the small amount of samples and the lack of viral pneumonia samples in the validation folder. To solve this issue, we take random samples of the files in the training set without replacement. This random sampling is done on each class so that each has 277 images in the validation set (this accounts for the 8 bacterial pneumonia and normal images already in the validation set). In total, this constitutes 14.17% of the total data, which will be a sufficient validation set. We now have a balanced validation set as well, with the same number of each image. This was the most natural way to divide the data, as any imbalances will 1) make an assumption on the distribution of our three classes and 2) make the accuracy score on the validation set less meaningful. The new distribution of our data among validation and test sets are shown below:

After Data Balancing

Folder	Class	Number
train	Normal	1075
	Bacterial Pneumonia	2264
	Viral Pneumonia	1071

validation	Normal	277
	Bacterial Pneumonia	277
	Viral Pneumonia	277

Sampler

To change the amount of samples our model sees of each class, we use a WeightedRandomSampler in the DataLoader. It takes a list of probabilities (1 for each image; these don't have to sum up to 1) that determines how likely the model is to select that image for training. This offers us a chance to either balance the training data batches exactly, or force our model to consider more of one class than another. The latter will help distinguish between classes that are difficult to separate by forcing the model to see more of those classes, provided we are careful not to overfit.

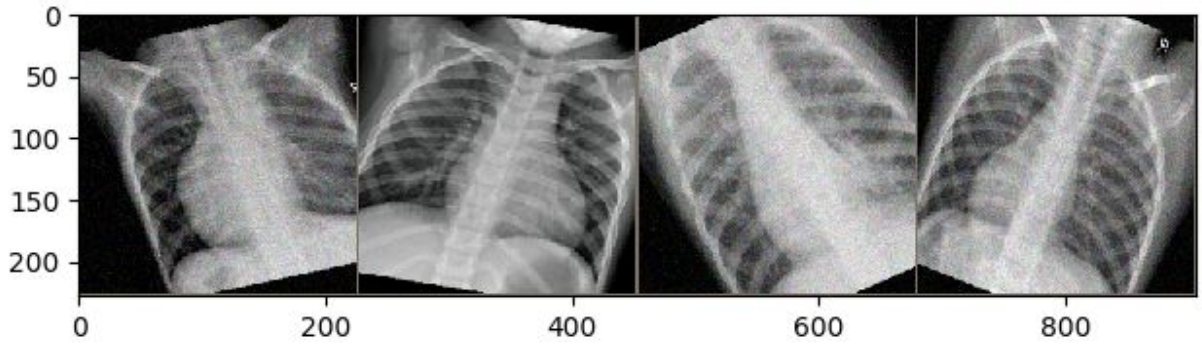
To balance the classes, we simply have to find the fraction of observations that belong to each class and assign that probability to its respective class.. Given the train data distribution, the probability for viral pneumonia and normal images will be nearly the same, and be approximately twice as much as the probability for bacterial pneumonia (our majority class).

To force our model to spend more time learning to distinguish specific classes, we can also assign each class a probability manually. When we tested this however, we did not find a probability distribution that produced better results than balanced training sets.

Transformations:

In testing our models, we found no significant difference in the accuracy of models trained with the full images (cropped to the necessary size for the network) and networks trained images resized (using OpenCV) to 256x256, although there was a significant difference in training time. To reduce this, we resized the images and saved to a new folder before training to reduce runtime.

We also used several augmentations in PyTorch to minimize overfitting in our models, such as RandomAffine, of which we used rotation (typically between -30 and 30 degrees), scaling (typically between a factor of 0.8 and 1.2) and shear transformations (typically 10 degrees). We used a RandomResizedCrop, which randomly takes a crop from the image and resizes it to 224x224 pixels. The scale of the crop was typically between half the image size and the full image size; this naturally accounts for the lung position within an image by shifting the relative location of the lung randomly. We applied Gaussian, or 'salt and pepper' noise to randomly with a 50% chance. These augmentations are shown below:



4. Modeling

Performance Index

We chose to use Cross Entropy Loss for all of our models below, a standard for classification problems such as this. Cross Entropy Loss for multiclass classification is given by the equation below:

$$-\sum_{i}^{M} y_i \cdot log(p_i)$$

where M is the number of samples, y is a vector with 1 for the true label and 0 for all other entries and p is the output probability vector from the model. Using Cross Entropy loss, we can maximize the likelihood that our output distribution matches the target distribution; as Cross Entropy Loss is common in the community, we do not explore other potential loss functions.

Hyperparameters

Our hyperparameters will be determined experimentally whenever possible, and taken from research standards when not. We will start with hyperparameters that have been successful in similar models; for example, the first learning rate for ResNet will be 1e-3, available from an example of transfer learning with ResNet in Pytorch (https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html) that worked fairly well; it is also a suggested learning rate for image classification. As such, we use this as the starting learning rate for many of our models. This is one area where our models were lacking, a sufficient exploration of hyperparameter space

Optimizers

We experimented with several different optimizers including Stochastic Gradient Descent (SGD), Adam ,etc. For our best model using ResNet (described further in the Final Model Description section), we use SGD, as this is what was used in the original training of ResNet (Kaiming 2015). Also, SGD with well-tuned parameters may converge to better minima than adaptive optimizers (see discussion in Ashia et al. 2018). Adam may have generated better results however, especially with our lack of exploring the hyperparameter space.

Custom CNNs

We tried several different models in an attempt to minimize the Cross Entropy Loss performance index, and maximize the precision and recall. We started by building our own custom architectures, but given the relatively small data set, training a CNN from scratch did not produce results as sufficient as pretrained models. In the first few efforts, we tried to develop a CNN from scratch which contained various layers. The initial trials consisted of CNN architectures with upto 7 layers with activation functions, BatchNormalizations, Pooling layers and a Linear output layer. The best model among these yielded a minimum CrossEntropyLoss of 0.9 (which is considerably high for this problem statement) and a validation accuracy of upto 72.57 %. However, achieving higher accuracies is difficult for a CNN built from scratch with the relatively small data set we have. A solution is transfer learning, which can generalize better than scratch CNN models with fewer samples.

We started to experiment with Pre-Trained Networks from Pytorch, such as ResNet18, ResNet34, DenseNet121, VGG16 and Inceptionv3. Among the first few were Resnet18, Resnet34 & DenseNet121. The Resnet pre-trained models showed much better results that gave a validation accuracy in the range of 69% to 78%, and that took less time to train (30 minutes (ResNet18) vs 90 minutes (VGG16) vs 60 minutes (DenseNet121)), so we explored ResNet more deeply (see ResNet; next section).

The ResNet18 seemed like the learning was saturating closer to 15 epochs with the above parameters and started to overfit the training data from ~17th epoch. The training loss fell rapidly after this and provided a training accuracy of 90% around the 20th epoch.

Once ResNet18 gave a better result, I continued pursuing more complex versions of these pre trained models which have more number of trainable parameters. I tried Resnet34 which gave better results but not better than ResNet18. I also noticed that the Resnet34 model seemed to overfit the data much faster than the previous model. The training stopped and saturated around the 9th epoch at a validation accuracy of 72% and continued to stagnate around the same value for the rest of the training cycle while the training loss rapidly grew until the 20th epoch. Finally, The loss minimized at ~0.98 and gave a validation accuracy of 74%.

CNN Layers	Batch Size	LR	Epoch	Changes	~Validation Loss	~Validation accuracy
Ch = 16,16 (2 Layers, k = 3, MaxPool)	128	0.001	30	SGD	2.23	39%
16,16 (4 Layers, k = 3, MaxPool)	128	0.001	30	#layers	1.94	48%
16,16 (7 Layers, k = 2, MaxPool)	128	0.001	30	#layers #kernels	1.21	69%
16,16 (12 Layers, k = 2, MaxPool)	128	0.001	30	#layers	1.37	49%
16,16 (16 Layers, k = 2, MaxPool)	128	0.001	30	#layers Adams	1.21	57%

16,16 (7 Layers, k = 2, MaxPool)	64	0.001	30	Batch_size	1.77	57%

To create a more powerful model that can distinguish viral and bacterial pneumonia, we tried training ResNet34, which has more convolutional layers than ResNet18 and should be able to distinguish the viral and bacterial classes more easily.

ResNet

Transfer learning with CNNs generally achieves better results in much less training time than models trained from scratch, due to 1) the models have established kernels for basic shapes and 2) have been trained on a variety of classes (1000s in the case of ResNet); therefore, it is easier to create more generalizable models. ResNet, or Residual Network, fashion each layer's output as the sum of the layer activation and the input, as in the following graphic (Kaiming 2015)

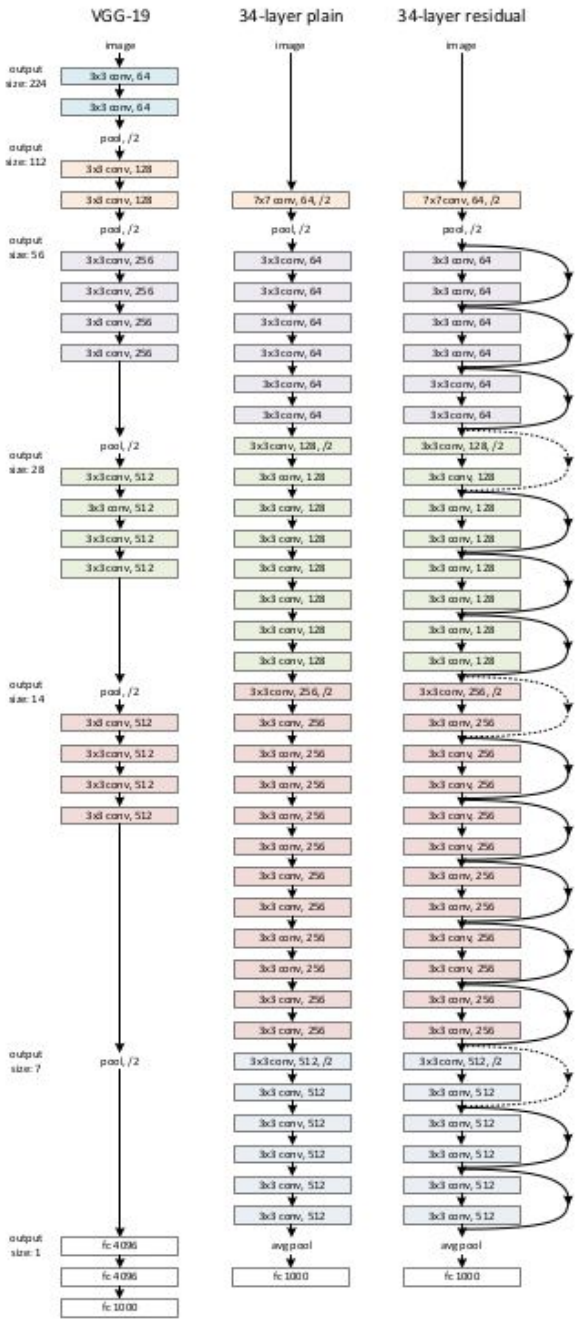


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

:

The black curves passing around the layers are called ‘skip connections’, or residuals; essentially, the skipped layers are not learning the mapping from input to output, they are learning to map the input to the residual of the input and the output. For example, let H(x) be the function a subset of a network is supposed to approximate (not necessarily the entire network, but a few layers). If it is hypothesized that the model can asymptotically approximate H(x), it can be equivalently hypothesized that the model can approximate H(x) - x, or the residual. Kaiming 2015 let these layers (typically 2 convolutional layers, as shown above) approximate the residuals directly, letting H(x) - x = F(x), where F(x) is the output of the layers. This gives the desired function, H(x) = F(x) + x, hence the ‘skip connections’.

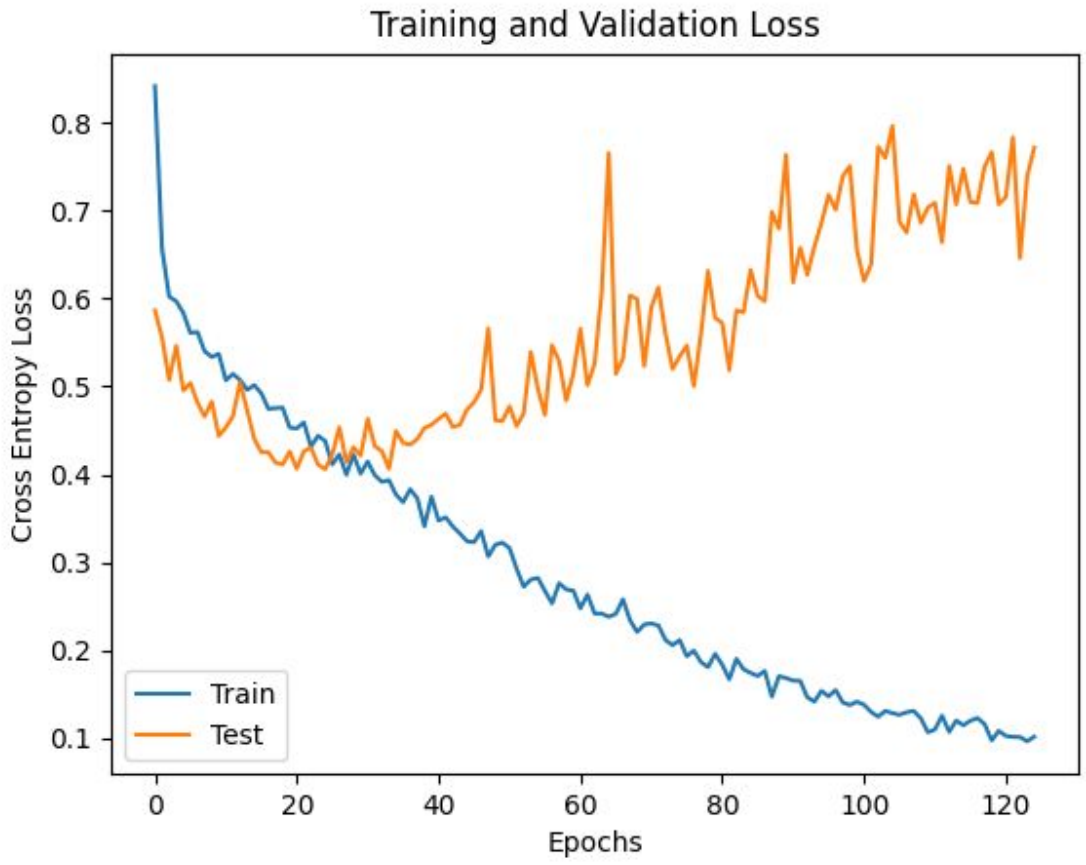
In both our experiments, these residual networks were shown to be faster than plain networks with no skip connections and achieve similar, and sometimes better, results than plain networks. Therefore, we built several fully connected classifier architectures with ResNet as the backbone, including 1-layer, 2-layer, 5-layer and 6-layer networks named ResNet_fcN, where N is the number of layers in the fully connected classifier. For any network with 2-layer and above classifiers, training the entire network overfit very quickly. Therefore, we froze the ResNet neurons after seeing signs of overfitting, and continued training the fully connected classifiers. However, the network performance stagnated in each of these models on both the training and validation sets, and none outperformed ResNet with one fully connected classifier (ResNet_fc1).

ResNet_fc1 Results

ResNet_fc1 was trained for 125 epochs total and overfitting began at epoch 25. We trained the network for another 100 epochs to be sure the network would not converge to a smaller minimum. It was trained with the following hyperparameters:

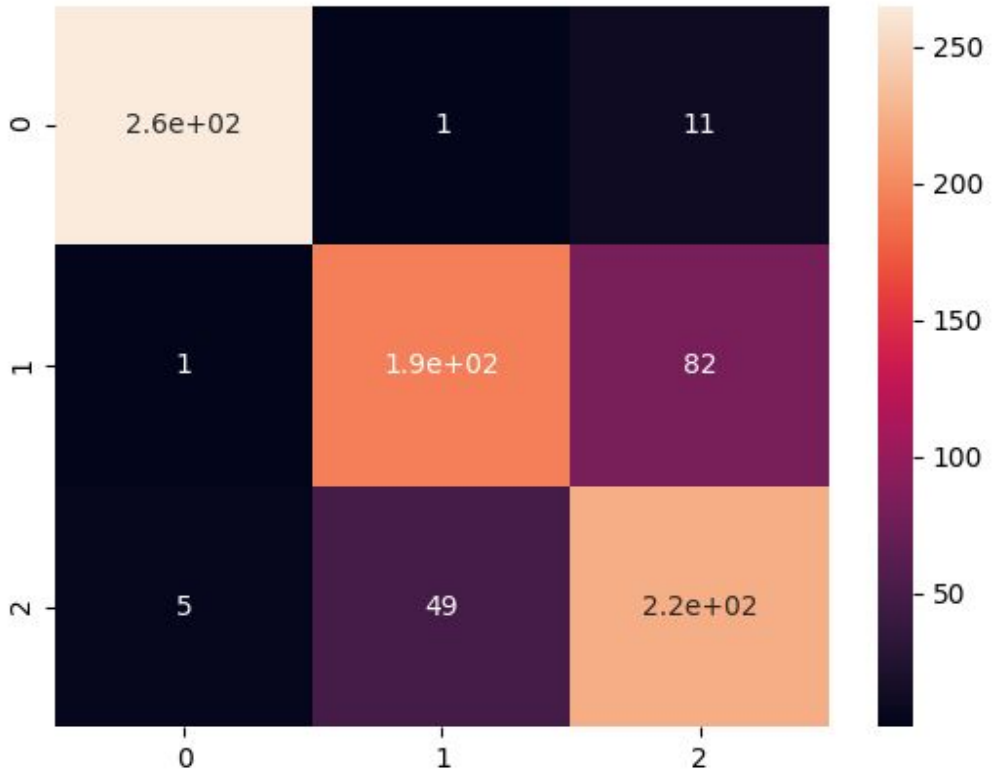
Hyperparameter	Value
Learning Rate	1e-3
Batch Size	64
Sampler Weights	Chosen at runtime to balance classes

The learning rate is the standard 1e-3 we start testing our models with, and the batch size is chosen because it gives better results than a batch size of 32, but requires less run time. We used SGD as the optimizer, with a momentum of 0.9 and no weight decay. The validation (labelled test here) and training loss are shown below:

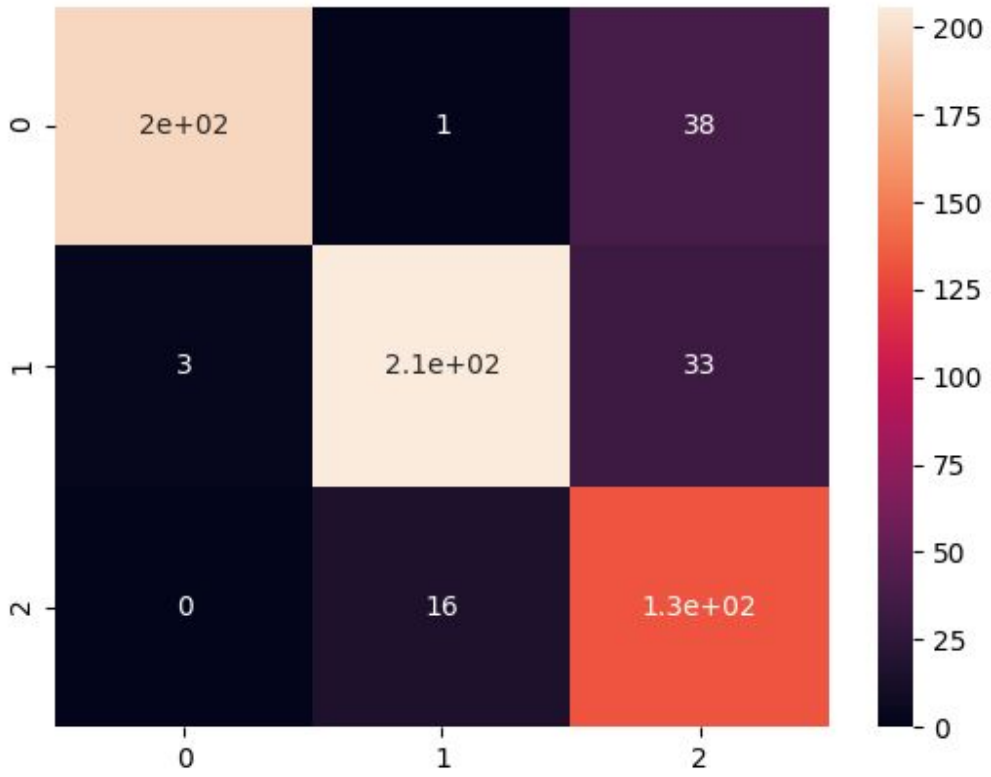


We see that the model begins overfitting after ~25 epochs, and does not reconverge. The confusion matrix for this model at 25 epochs is shown below:

Validation:



Test:



In the confusion matrices above, 0 represents normal, 1 bacterial pneumonia, and 2 viral pneumonia.

Several metrics for this model are shown below:

Validation:

Precision	0.826
Recall	0.821
F1 Score	0.821
Cohen's Kappa	0.731

Test:

Precision	0.882
Recall	0.854
F1 Score	0.861
Cohen's Kappa	0.781

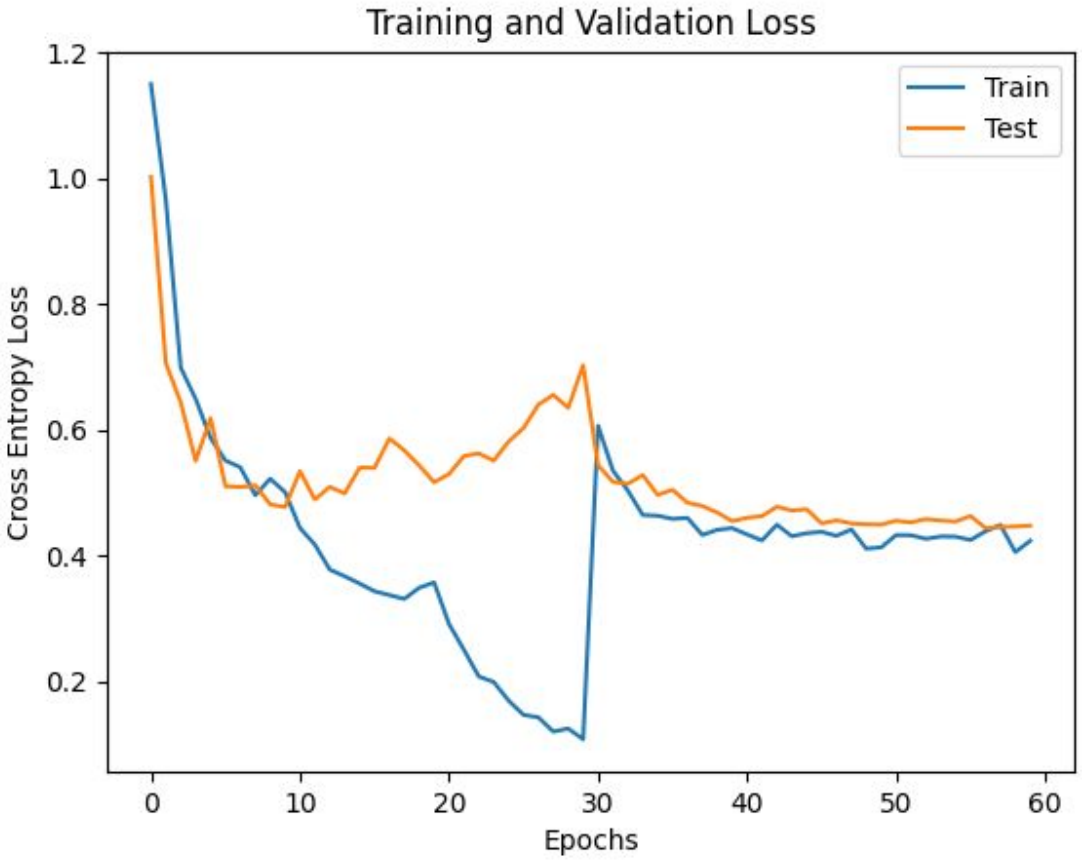
This model is a decent predictor of all classes, with relatively high scores across all metrics. The confusion matrix shows that the model easily classifies normal cases from those with pneumonia except for mistaking some normal cases for those with viral pneumonia (38 images). It has more difficulty distinguishing viral pneumonia from bacterial pneumonia, shown as the 2x2 portion of the confusion matrix in the bottom right corner.

ResNet_fc6 Results

ResNet_fc6 was trained in full for 20 epochs, and then ResNet was frozen while the classifier trained for another 40 epochs. The architecture of the fully connected classifier network is shown below:

Layer	(Input Features, Output Features)
1	(4096,4096)
2	(4096, 4096)
3	(4096, 2048)
4	(2048, 2048)
5	(2048, 1024)
6	(1024,3)

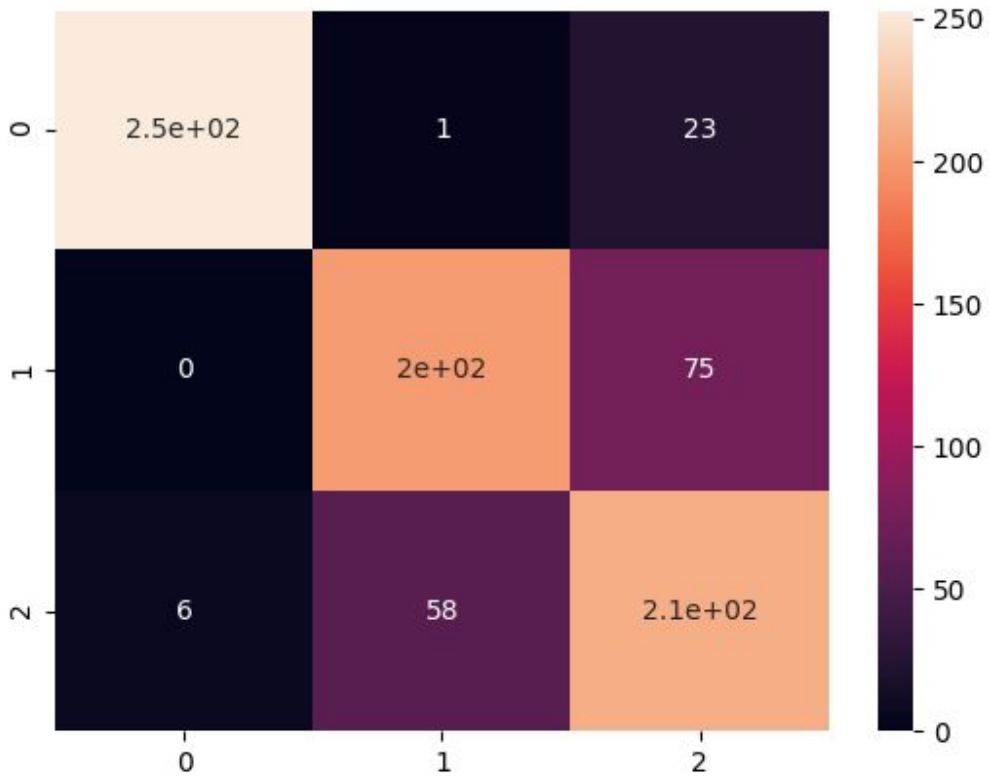
We used the same optimizer here as in ResNet_fc1. The transfer functions between the fully connected layers are all LeakyReLU, with a slope of -1e-3, and the Dropout is chosen as 0.5 for each layer. Below are the validation and training loss:



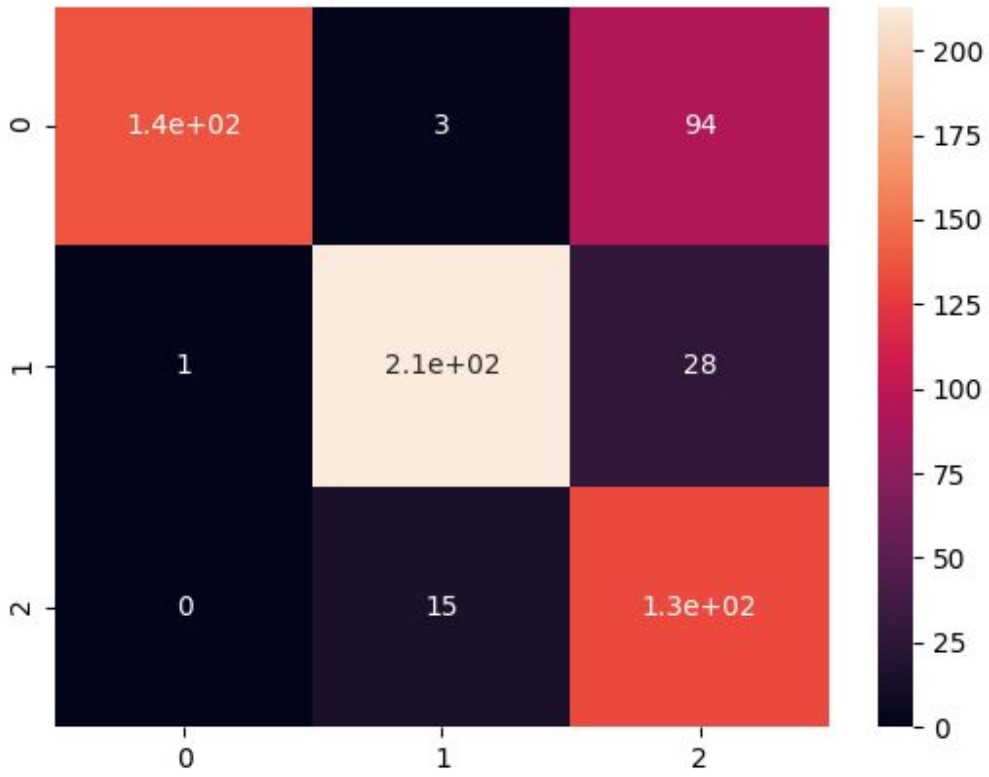
The model is overfitting at 20 epochs, and after the ResNet neurons are frozen, the training loss continues to decrease while validation (test in the figure) loss increases in a very discontinuous spike, interestingly. After, the validation and training loss settle together above 0.4.

The confusion matrices for the validation and test set:

Validation:



Test:



We can see that this model has similar issues with normal vs. viral pneumonia and bacterial vs. viral pneumonia as ResNet_fc1, albeit worse.

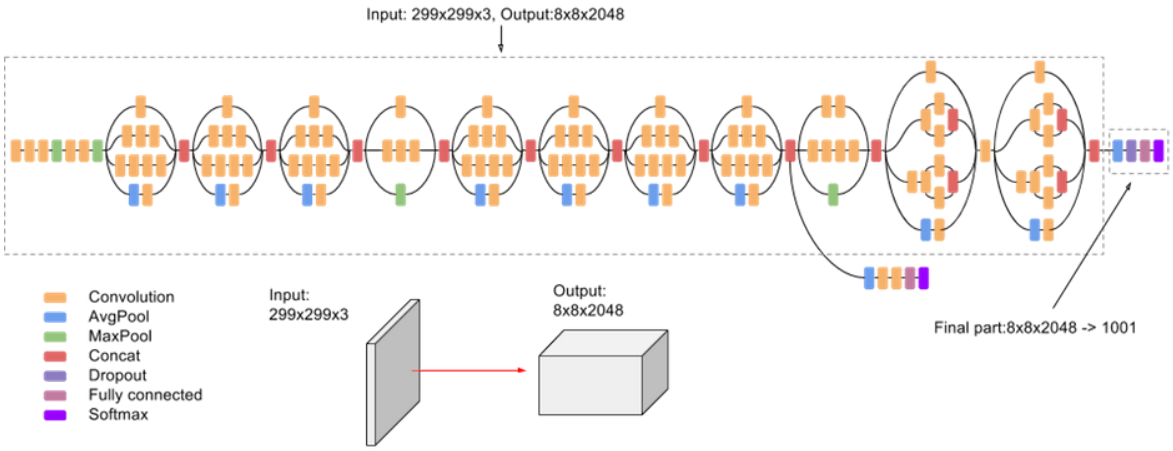
VGG16 and Inception V3

Since the ResNet18 gave us a good result after data balancing, we want to try whether adding inception V3 and VGG 16 to ResNet will show better results with only the original dataset instead of after data balancing.

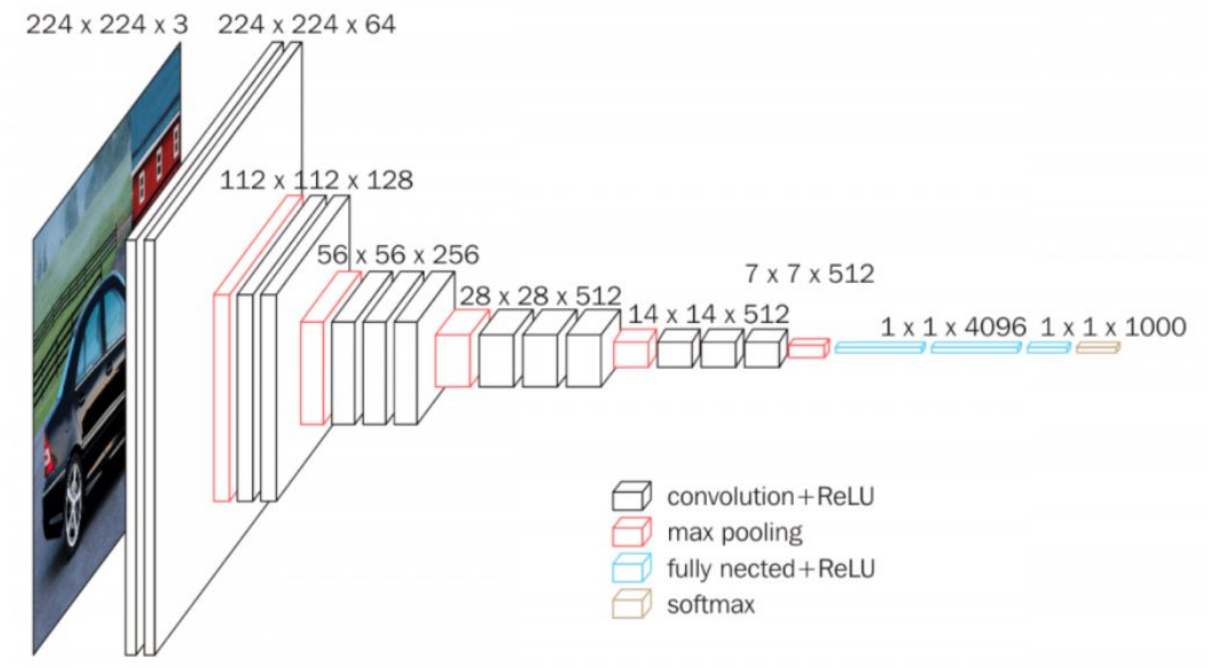
Inception V3 model can be utilizing the added computation as efficiently as possible by suitably factorized convolutions and aggressive regularization. Similar to vgg16m, it is also a vision model architecture. The great thing about VGG16 is it focused on having convolution layers of 3x3 filter with a stride 1 and always used the same padding and maxpool layer of

2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture.

Inception V3:



VGG16:



We use a similar augmentation with CNN model but added 'transfer.ColorJitter' to make the picture more recognizable. For our model, we only use the parameters with [brightness = 0.5, contrast = 1.5]. We set up the parameters are 128 batch size, 0.005 LR, 0.001 decay and 25 epoch. First, we tried to use WeightedRandomSampler to ensure that each batch sees a proportional number of all classes.

```
sampler = WeightedRandomSampler(weights, num_samples=int(0.7 * len(datasets['train'])), replacement=True)
```

The WeightedRandomSampler obtains the list of target classes and shuffle. Then, get the class counts and calculate the weights and class by taking its reciprocal. Next, it assigned the weight of each class to all the samples. Finally, obtain the corresponding weight for each target sample. Therefore, we received our data proportion is 1:2:1.

However, after using the sampler, our training accuracy became around 90% but the test accuracy was only around 50%. That means the training is overfitting. Therefore, we changed to the general DataLoader which the result didn't overfit; and We then received the train accuracy was around 79% and the test accuracy was around 60%.

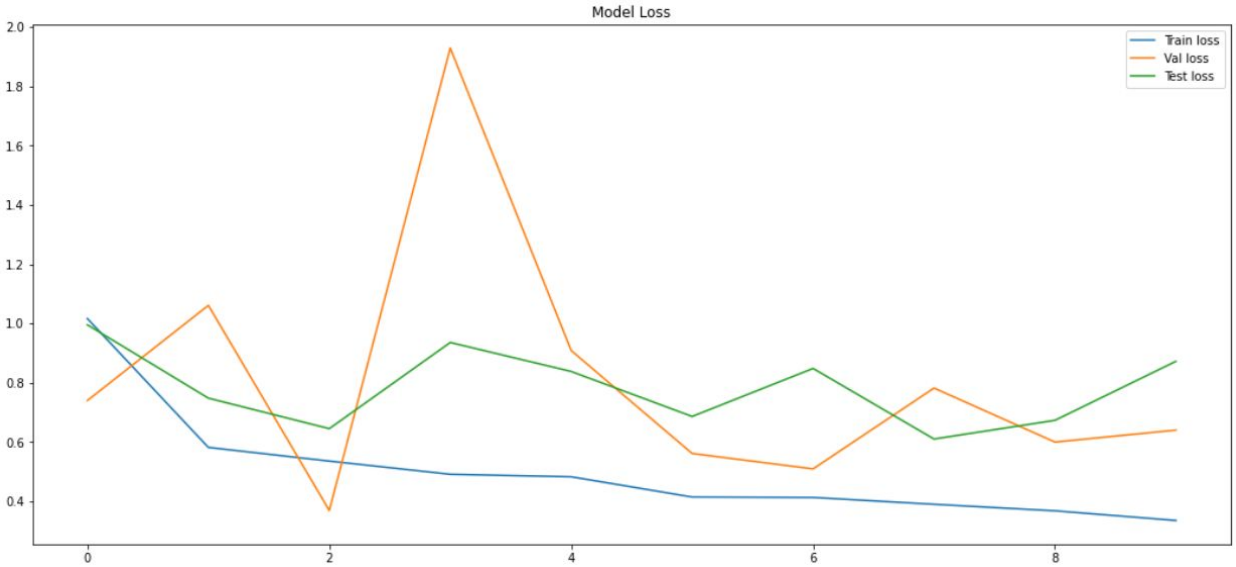
```
dataloaders = {mode: DataLoader(datasets[mode], batch_size=batch_size, shuffle=True) for mode in modes}
```

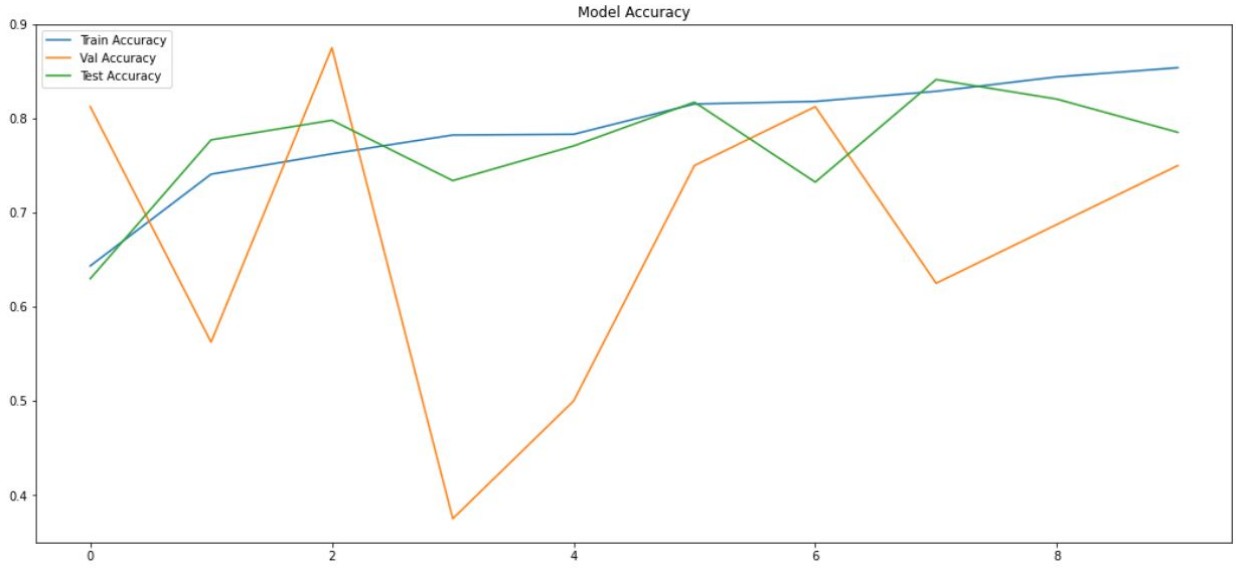
We also tried to change the epoch to 50 to see whether the accuracy will increase with that. However, the result was more stable around 20 epochs. Following step, we tried SDG and Adam as the optimizers. Looks like only Adam optimizers for the two models turned out better results around 80% of training and validation accuracy and 72% of test accuracy. Nonetheless, validation accuracy was very unstable because the original validation folder only contains 16 pictures with no viral pneumonia. Moreover, we read about some discussion only that vgg16_bn() is better than vgg16 based on people's experiments. However, we tried it but didn't see the result significantly change with our dataset.

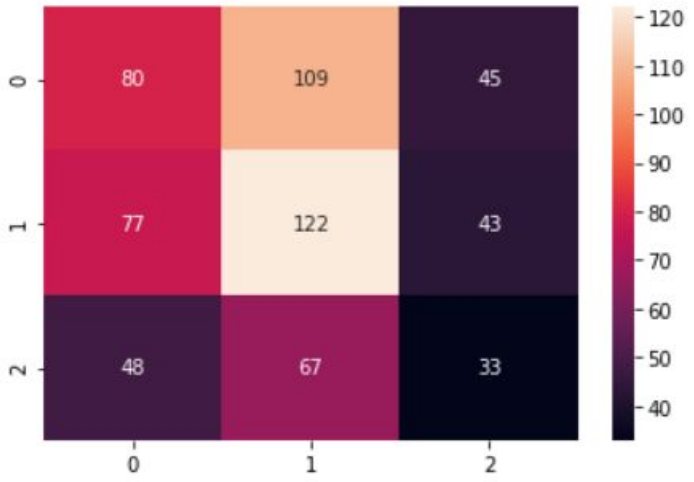
Result

The following 3 graphs and confusion matrix are the best result we train with Inception V3 and VGG16 models. The reason why the val loss is unstable is because the original dataset

validation folder only contains 16 pictures with no virus image. Not enough samples makes the model unpredictable.







5. Conclusions

Our best model was our ResNet_fc1, with higher precision, recall, f1 score and Cohen's kappa than any other model we trained. It has some difficulty still with viral and bacterial pneumonia in our test and validation sets, and even confuses normal lungs for viral pneumonia. However, our ultimate goal is to reduce the recall of each class to the smallest it can possible be, especially in the case of bacterial pneumonia. When bacterial pneumonia (generally more dangerous for children than viral pneumonia) is the true target, we want to predict bacterial pneumonia as much as possible. The recall for the bacterial pneumonia class for ResNet_fc1 is around 73%, which is not as high as we would like for this particular problem. Fortunately, when the true target is bacterial pneumonia the model only predicts normal for 4 of our images in both test and validation, as this would be the worst-case scenario (predicting normal when the child has bacterial pneumonia); at least if viral pneumonia is predicted, the child will likely be under observation at the hospital.

Given ResNet_fc1's success, we were anticipating that a few more layers added to the classifier would improve predictions, potentially by a large margin. We never saw this increase however; the model overfit very quickly with extra layers, and when the ResNet neurons were frozen we saw the model stop training or perform in strange ways (see the validation and train loss plot for ResNet_fc6).

In general, we were able to get better results from transfer learning than custom architectures trained from scratch. This is not surprising, given the relatively small amount of data.

Balancing each batch performs the best out of any other distributions we chose, despite the potential in exposing the model to more viral and bacterial pneumonia images in order to better learn the distinction between the two. However, most of the rebalancing of the dataset was done trying to train the ResNet models with fully connected classifiers, so separate class distributions may provide other models with significant improvements.

One potential reason the fully connected layers are not training well may be the high dropout. 0.5 is fairly high, and may be inhibiting learning. In general, the hyperparameters of the ResNet model should be tuned.

6. Further Improvement

The raw dataset only contains two labels - Normal and Pneumonia. Originally, the model only needed to identify whether the picture is normal or pneumonia. During the training, we try to separate Bacteria and Virus which might cause false acceptance rate increase. For the further model, we will add one model that only training the two pneumonia labels. Instead of having three categories in one model, we will have two models that both have two categories.

The most difficult class to distinguish from one another is consistently viral and bacterial pneumonia in our models. Signal processing techniques that can potentially accentuate the differences between viral and bacterial pneumonia would make it easier for our models to distinguish these classes. We could attempt a number of color, contrast, or saturation changes or filtering the images in order to find either known or potentially unknown features of viral or bacterial pneumonia. Another solution may be to add more layers to our models in order to map the data onto a space where the decision boundary is not as complex, but this proved difficult in practice. Increasing the layers of the fully connected classifier appended to ResNet did not improve the validation accuracy, nor did training the 34-layer version of ResNet. It is possible that even larger models may improve our results, but these models become more difficult to train and overfit more quickly. A

hyperparameter exploration of these models may yield successful results, however we leave this for future work.

Also, figuring out the correct batch size and epoch might improve our result. The article "Advanced Guide to Inception v3 on Cloud TPU" shows that the inception v3 has attained greater than 78% accuracy in about 170 epochs on the ImageNet data set for all the examples that they train. In absence of overfitting, this model should be able to perform at least as well on our data, but we would need to test out to 170 epochs, which would require approximately 4 hours of runtime.

Of course, more chest X-Ray data would create a better model, but an even simpler option would be to gather recorded clinical data, such as patient temperature, blood pressure or even doctor's notes. This type of information along with these CNN models to improve the automated diagnosis.

The simplest way to improve our model, however, would be to take our ResNet models and tune the hyperparameters; the rapid overfitting in these models is likely due to a relatively high learning rate (1e-3) and no regularization. Both of these could also improve the accuracy of our model.

7. References

https://neurohive.io/en/popular-networks/vgg16/

https://cloud.google.com/tpu/docs/inception-v3-advanced

https://towardsdatascience.com/pytorch-basics-sampling-samplers-2a0f29f0bf2a

https://arxiv.org/abs/1512.03385 - Deep Residual Learning for Image Recognition, Kaiming et al. 2015

https://arxiv.org/abs/1705.08292 - The Marginal Value of Adaptive Gradient Methods in Machine Learning, Ashia et al, 2018

https://pytorch.org/docs/stable/torchvision/transforms.html

https://pytorch.org/docs/stable/torchvision/models.html

https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

https://www.kaggle.com/c/cdiscount-image-classification-challenge/discussion/40249