DATS - 6203 Machine Learning 2

Individual Report

Jason Witry

Introduction

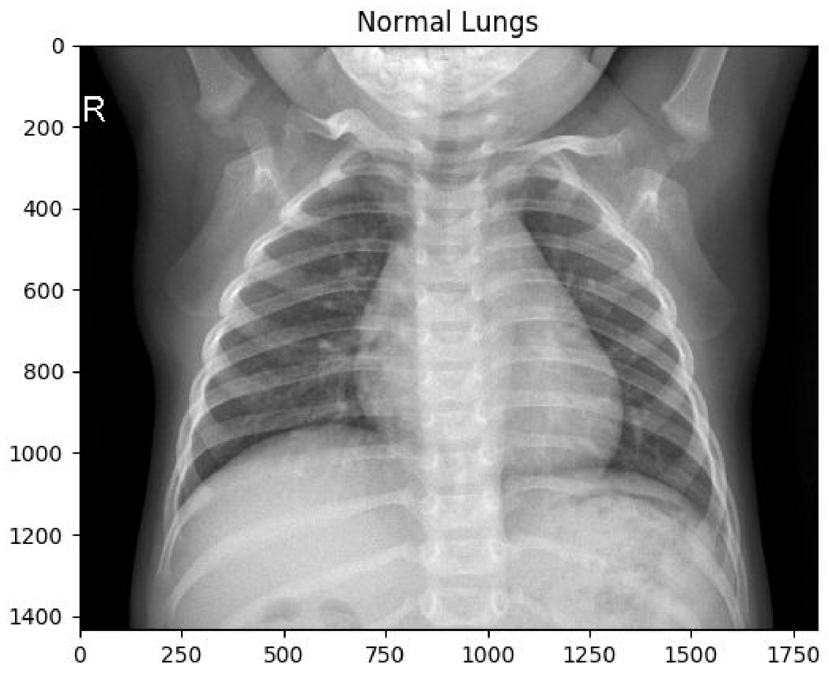
Pediatric pneumonia is consistently estimated to be the leading cause of childhood mortality (Rudan et al 2008). It kills more than malaria, HIV/AIDS, and measles combined (Adegbola, 2012). The two most common causes of pneumonia are bacteria and viruses (Mcluckie, 2009), but treatment methods differ 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.

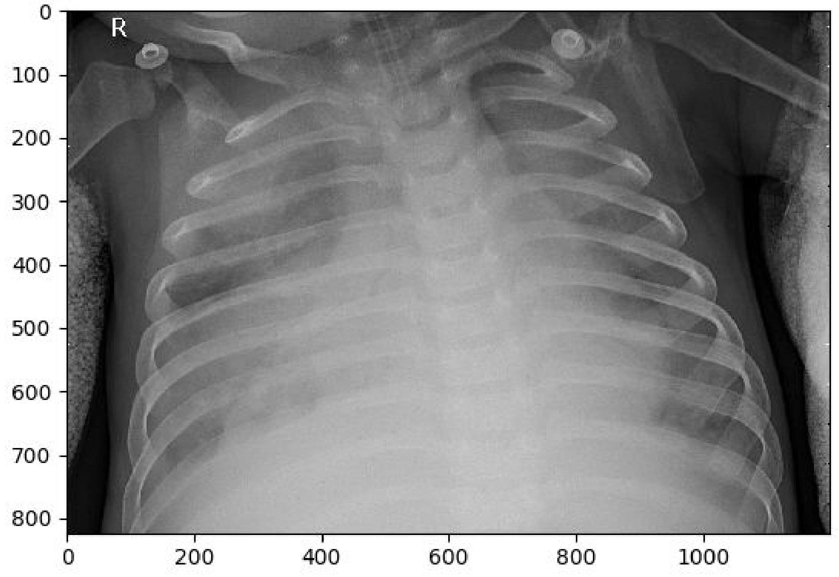
For my project, I attempt to develop an AI to do just that. The problem is fairly complex, as the division between viral and bacterial pneumonia proves difficult to model. However by experimenting with different algorithms and using transfer learning, I present a model that can solve this problem with reasonable (although improvements should be made) robustness. For this project, my group worked fairly independently, sharing necessary codes to preprocess data and run models but devising my own experiments. The report below details my personal contributions to this project.

Data

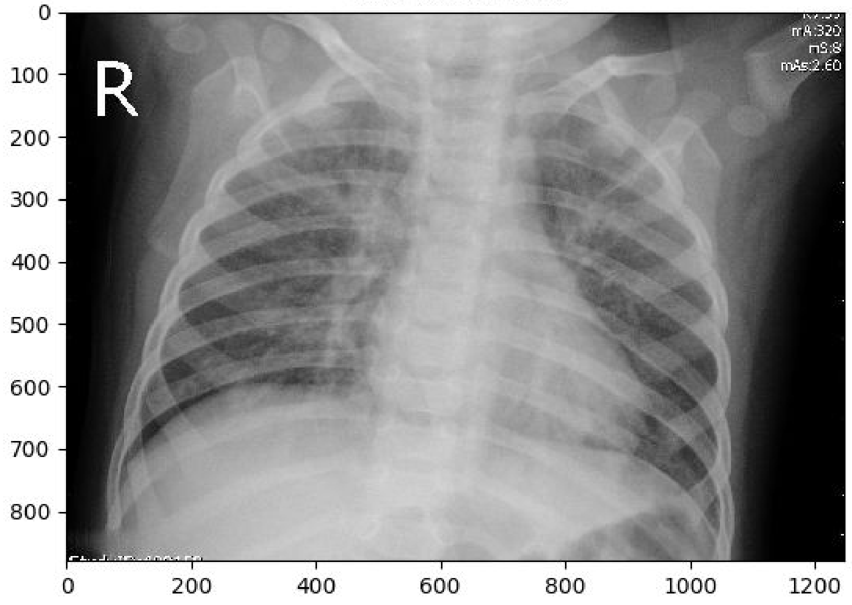
The first step for this project is to get a look at some of the training data. I loaded these images to offer a view:



Bacterial Pneumonia



Viral Pneumonia



To me, this problem already looks somewhat difficult; the lungs are behind the ribs, which may cause some interference when looking at the images. It looks as though pneumonia lungs may sag more than normal lungs; this may be something the model can (and did) pick up on when looking at the images. This would likely be due to the mucus in the lungs weighing them down. Viral and Bacterial pneumonia don't appear to have any significant differences, however, except for the high frequency filaments in the viral pneumonia image. I do not know if this is an actual difference or only appears from the small sample I looked at.

Individual Work

I wrote the data reorganizer code to resize and make my data accessible by PyTorch's ImageFolder class, wrote the ResNet models with fully connected classifiers on top, did background research on the project motivation, wrote the introduction as well as the ResNet sections and conclusions, and wrote the Gaussian and HammingWindow augmentation classes.

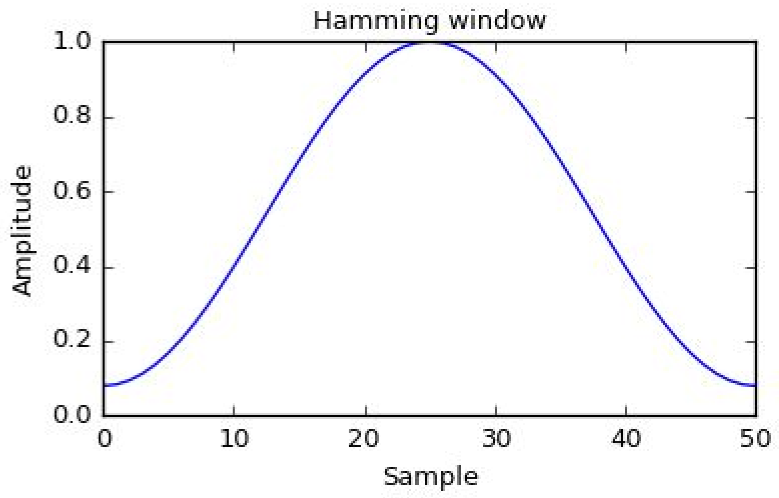
I also tried ResNet with several fully connected architectures as a classifier, but never got results better than the original ResNet, which ended up as my best model.

Specific Contributions

The first thing to do with the images was to make sure that PyTorch could load the images easily, so I wrote the data_reorganizer code. It takes in the path names of all the images, picks a random selection of training images to move to the validation set, and saves all the images back into folders where the classes are distinct, i.e. NORMAL, PNEUMONIA-BACTERIAL and PNEUMONIA-VIRAL. The data has about 5800 images, so I determined 15% of 5800 for the size of the validation set. I didn't realize, but I had made a mistake in my math somewhere; the validation set ended up only being 14.17% of the data set, which I overlooked until I had trained several models already so I left it.

Upon looking at the images of the chest X-Rays, I decided to try a transfer learning method, as Kartik had not had much success with models from scratch and transfer learning methods work well on data sets with small numbers of samples. I had been reading about ResNet, and found a tutorial on PyTorch transfer learning with ResNet. I took much of that code and modified it somewhat to fit my problem. After experimenting with ResNet some, I decided I should branch out and try some other models, so I experimented with a run of VGG16 and DenseNet121. VGG16 took ~90 minutes to run, while DenseNet121 took ~60 minutes and ResNet took ~ 30 minutes, all with the same accuracy. This can be recreated with the all_models.py script.

Upon doing more research, I came across the idea that high frequency features in the image could throw off predictions; they are present in both the viral and normal images above, called 'infiltrates'. As an aside, I tried filtering the images with a Hamming window, which cuts out high frequencies and preserves low frequencies. There is an editable parameter, r, which roughly corresponds to how spread the window is. The window looks almost like a Gaussian:



Filtering the images this way made for very poor results, however; training for 75 epochs only gave a validation loss of 0.69, and training time took twice as long due to the addition of a Fourier transform on all of the images.

In addition, I realized that with fully connected layers on top of ResNet, many of the models were overfitting rather quickly. Obviously I are adding many extra parameters, so I wanted to freeze the ResNet layers after training for a little, and then use it as a feature extractor. I did this by saving checkpoints every 10 epochs, and when I noticed overfitting I continued training the model with the ResNet parameter gradients requires_grad flag set to false.

Results

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)

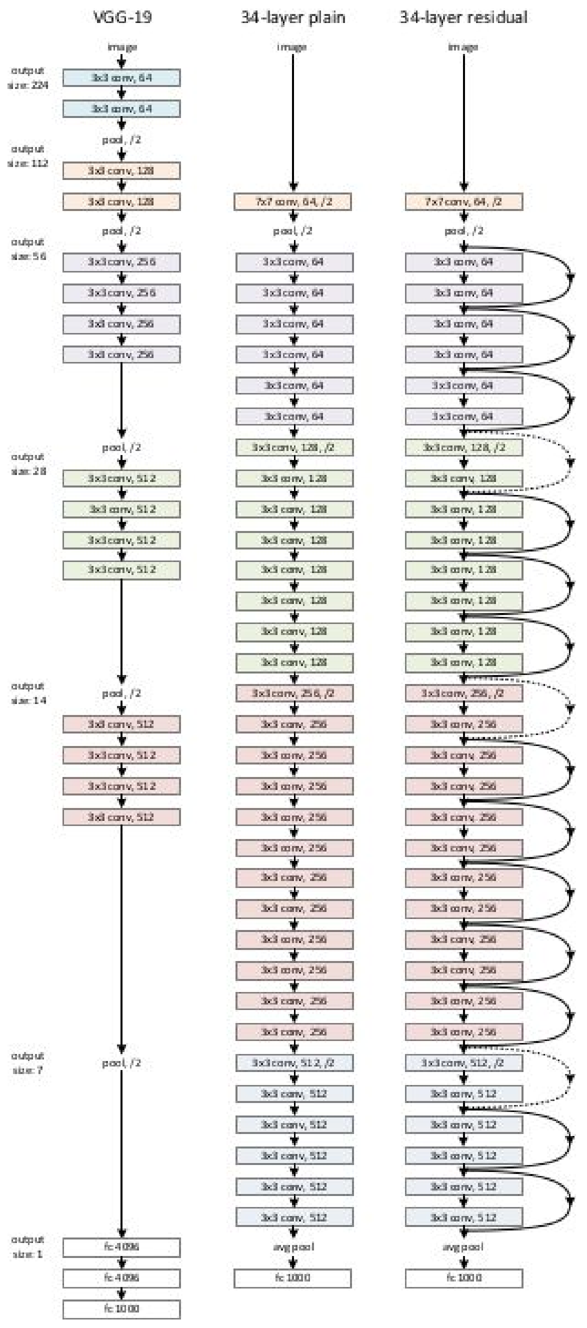


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase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

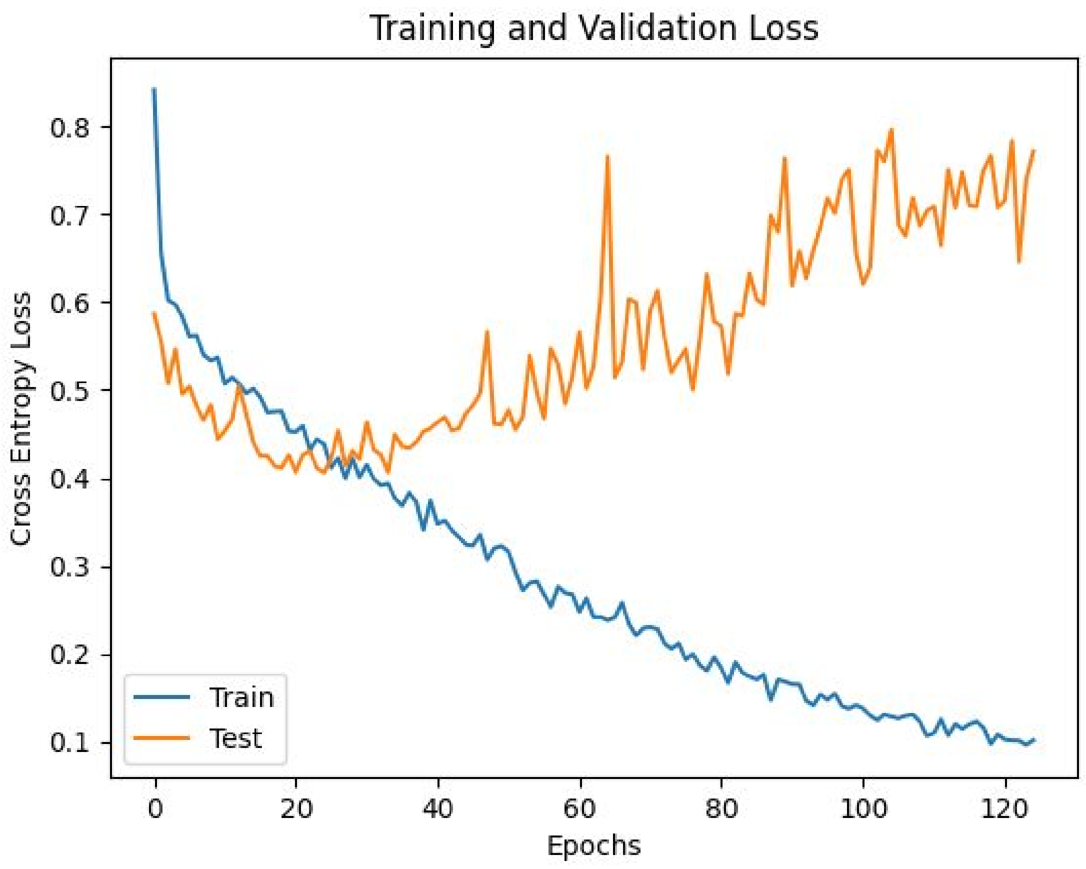
In both my experiments, these residual networks were shown to be faster than plain networks with no skip connections and achieve similar, and sometimes better, results than plain networks. Therefore, I built several fully connected classifier architectures with ResNet as the backbone, including 1-layer, 2-layer, 4-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, I 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. I 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:

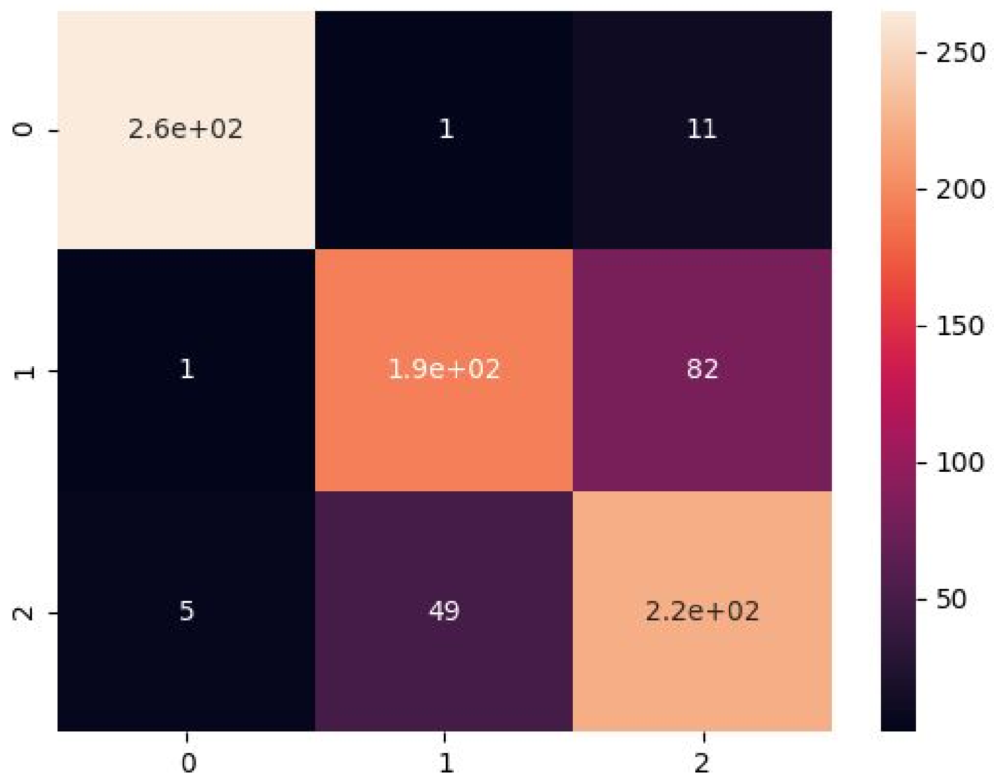
Hyperparameter	Value
Learning Rate	1e-3
Batch Size	64
Sampler Weights	Chosen at runtime to balance classes

The learning rate is the standard 1e-3 I start testing my models with, and the batch size is chosen because it gives better results than a batch size of 32, but requires less run time. I 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:

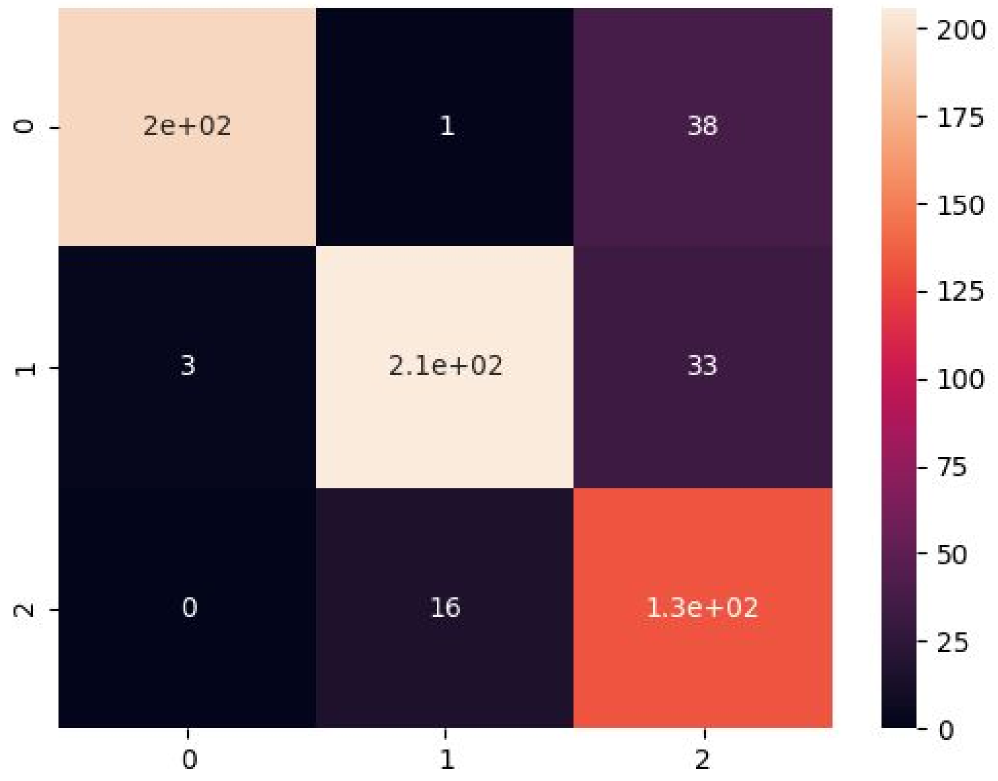


I see that the model begins overfitting after ~25 epochs, and does not reconverge. The confusion matrix for this model at 25 epochs is shown below:

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:

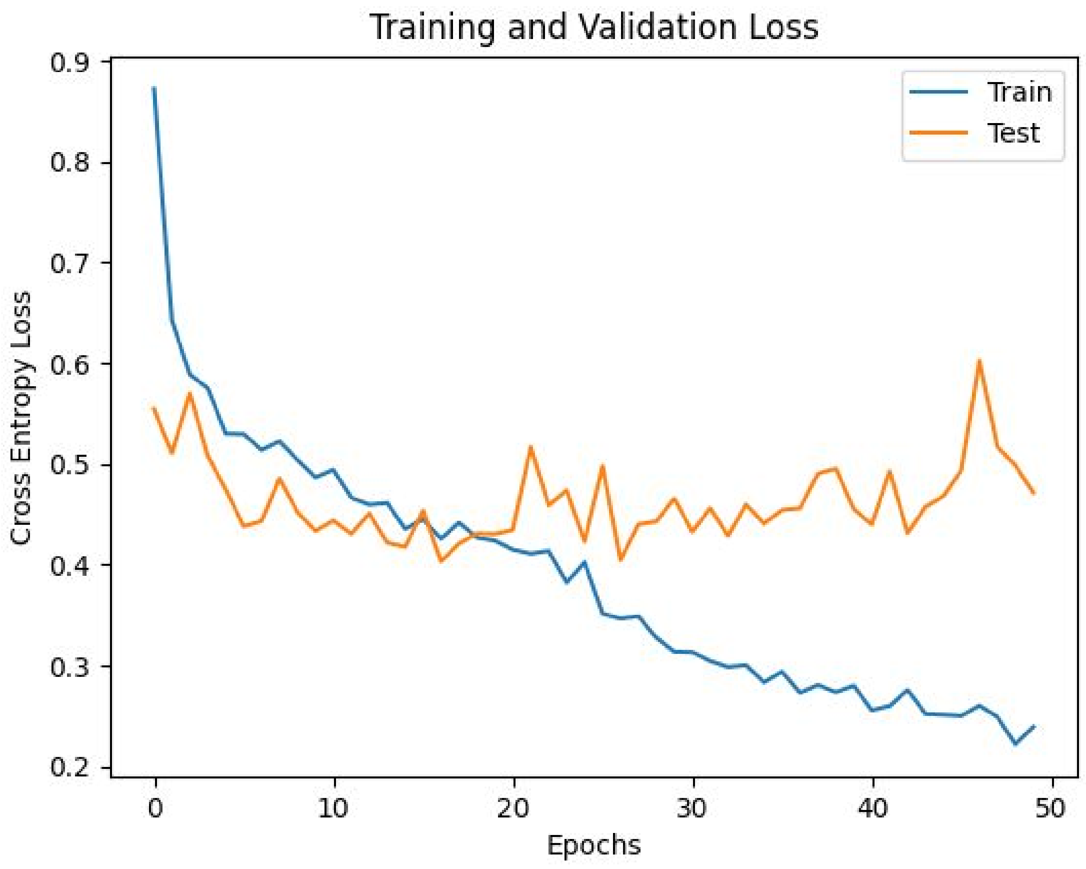
Precision	0.882
Recall	0.854
F1 Score	0.861
Cohen’s Kappa	0.781

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_fc2 Results

ResNet_fc2 was trained in full for 50 epochs. The architecture is shown below:

Layer	(Input Features, Output Features)
1	(4096,4096)
2	(4096,3)



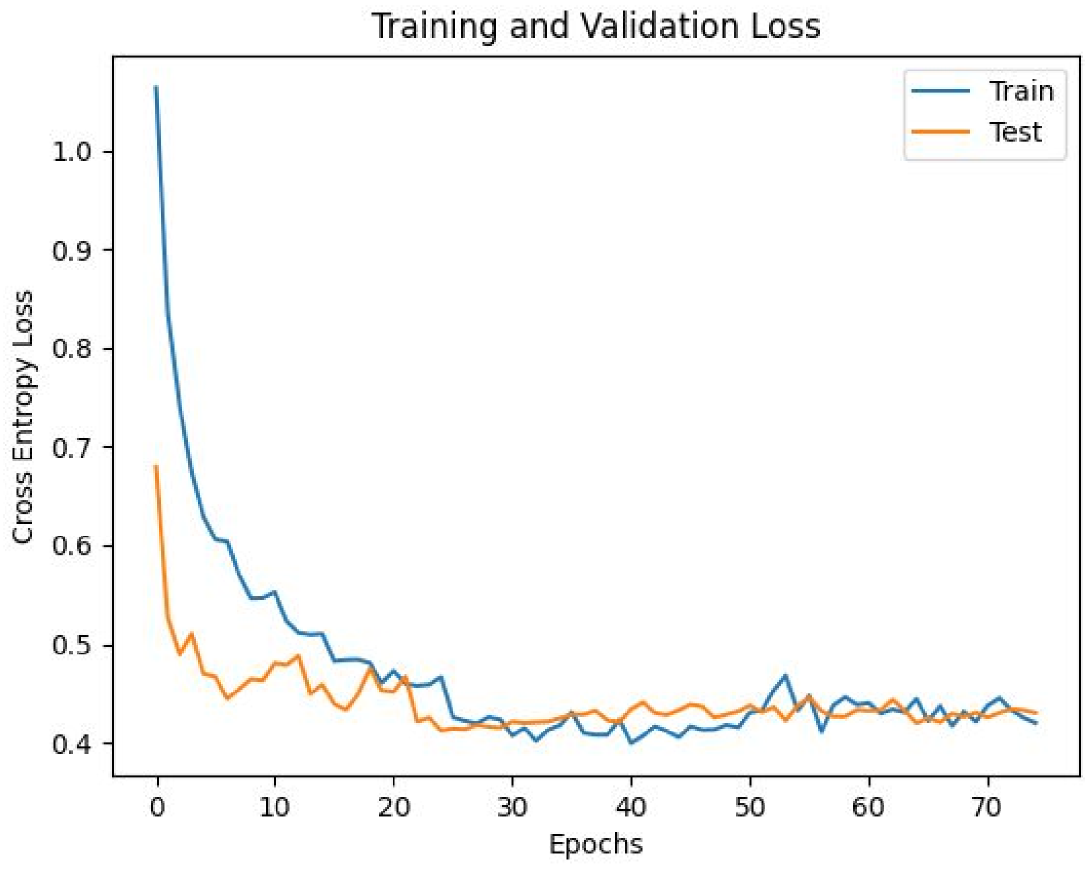
The validation and test loss are shown in the above plot. I can see clearly the model begins overfitting around 20 epochs; I realized I would need a simpler model. Here, I decided to try to train ResNet for a little, and then use it as a feature extractor to continue training the classifier. However, I know I would need more than a 2 layer classifier, so I increased the classifier layers to 4 as shown below.

ResNet_fc4 Results

ResNet_fc4 was trained in full for 25 epochs and then with only the classifier for 50 epochs. The architecture is shown below:

Layer	(Input Features, Output Features)
1	(4096, 4096)

2	(4096, 2048)
3	(2048, 1024)
4	(1024, 512)
5	(512, 3)



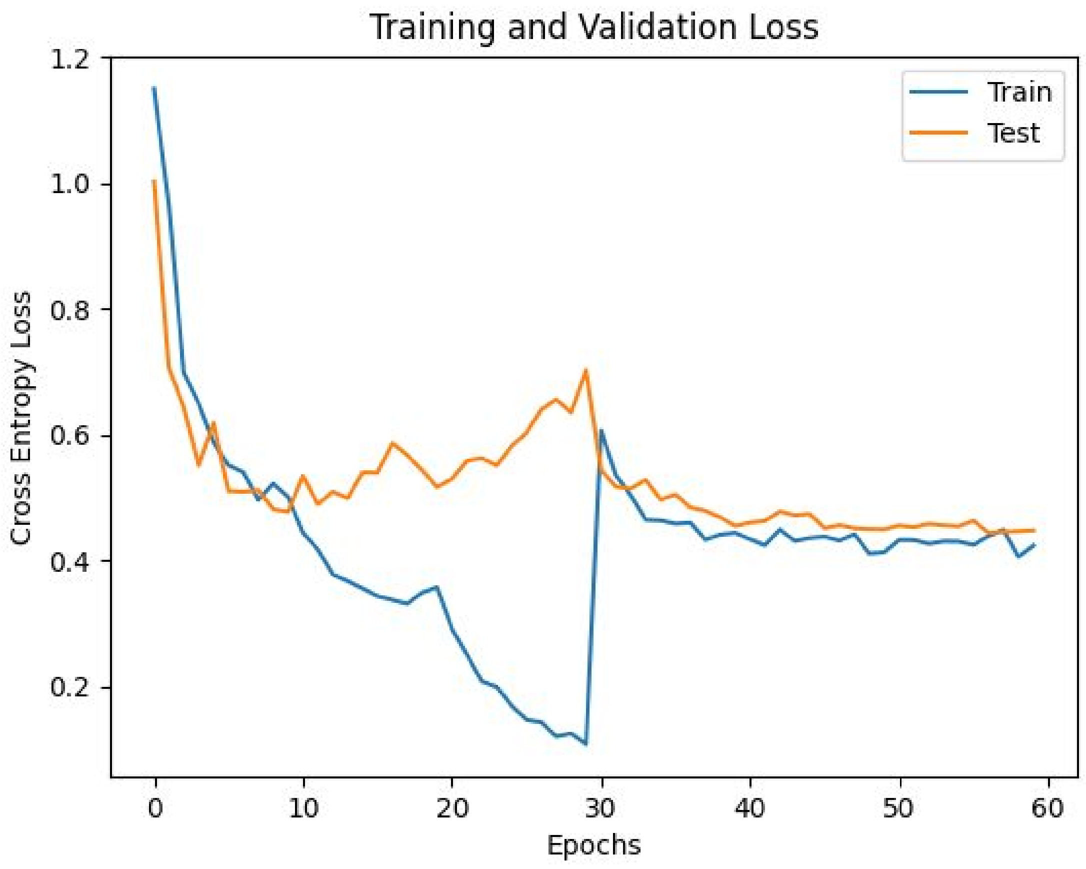
Here, I see that the model seems to converge, and not much training happens after the ResNet section is frozen at 25 epochs. So, the thought is that maybe a 5-layer architecture does not adequately model the complexity, even with ResNet as a feature extractor. So, I added another layer of 4096 neurons to increase the model complexity.

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)

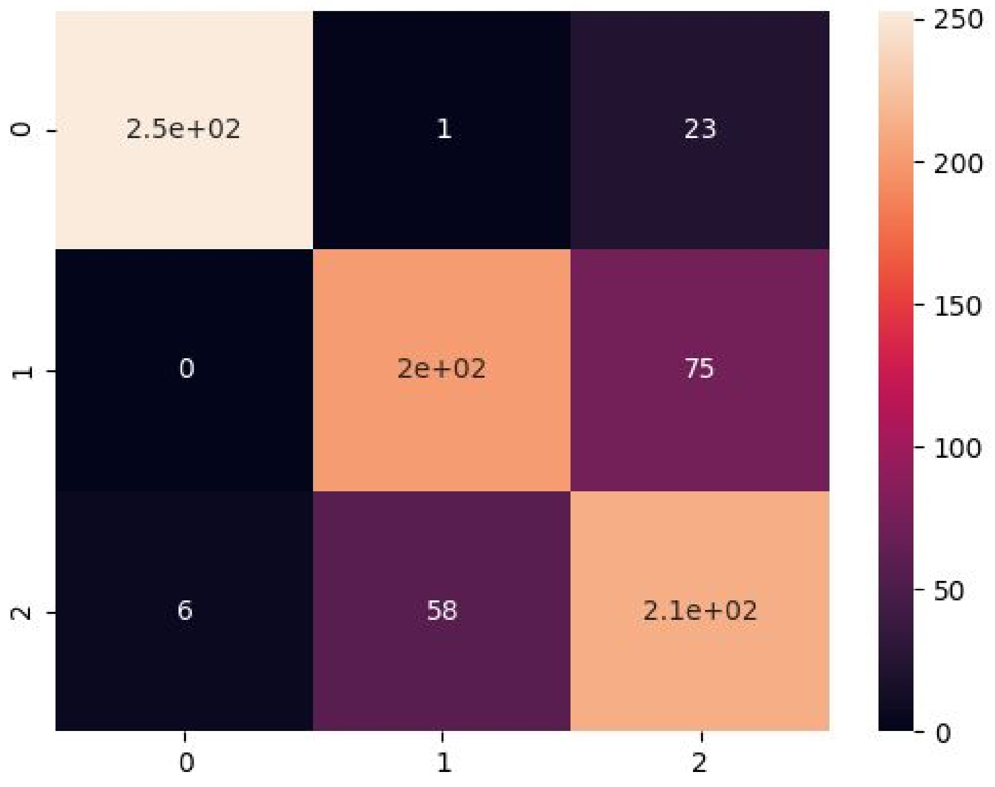
I used the same optimizer here as in ResNet_fc1. The transfer functions between the fully connected layers are all LeakyReLU, with a slope of -1e-3, and the Dropout is chosen as 0.5 for each layer. Below are the validation and training loss:



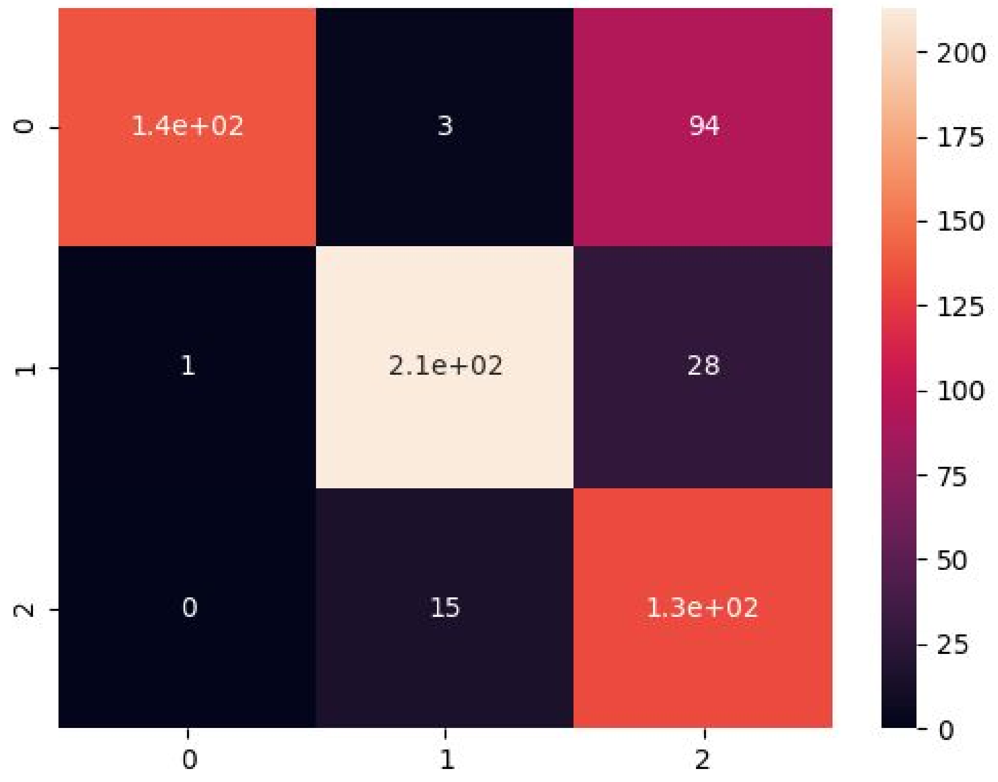
The model is overfitting at 20 epochs, and after the ResNet neurons are frozen, the training loss continues to decrease while validation (test in the figure) loss increases in a very discontinuous spike, interestingly. After, the validation and training loss settle together above 0.4.

The confusion matrices for the validation and test set:

Validation:



Test:



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

Here I see the interesting spike in training loss around 30 epochs; this was verified with another training run, where the training loss spiked at around 35 epochs. It is unclear what is happening here; a shift in the model seems to happen that lowers loss on the data it is training on somehow.

Summary and Conclusions

My best model was my ResNet_fc1, with higher precision, recall, f1 score and Cohen's kappa than any other model I trained. It has some difficulty still with viral and bacterial pneumonia in my test and validation sets, and even confuses normal lungs for viral pneumonia. However, my 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, I 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 I would like for this particular problem. Fortunately, when the true target is bacterial pneumonia the model only predicts normal for 4 of my 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, I were anticipating that a few more layers added to the classifier would improve predictions, potentially by a large margin. I never saw this increase however; the model overfit very quickly with extra layers, and when the ResNet neurons were frozen I saw the model stop training or perform in strange ways (see the validation and train loss plot for ResNet_fc6).

Future Improvements

To improve my models, I should have explored the hyperparameters more. I kept looking into different ways to augment the data or relatively simple ways to increase complexity, such as adding layers. But I forgot how important hyperparameters, especially the learning rate, are for training these enormous models.

For this project, I've learned many practical training things, such as how to keep logs and keep a notebook of what you train and why, as well as relevant knowledge about current deep learning trends and practices.

Coding Percentage

~ 250 lines of code from internet

~ 50 lines modified

~ 200 lines of my own code

250-50/(200+250) = 200/450 = 44.4%