DATS - 6203 Machine Learning 2

**Pediatric Pneumonia Detection with CNN**

Automating Medical Diagnoses

horizontal line

# Placeholder image

# Introduction

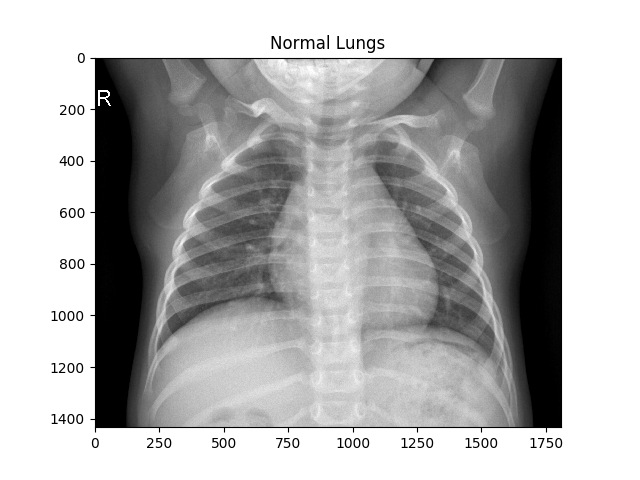
Pediatric pneumonia is consistently estimated to be the leading cause of childhood mortality (Rudan et al 2008). It kills more than malaria, HIV/AIDS, and measles combined (Adegbola, 2012). The two most common causes of pneumonia are bacteria and viruses (Mcluckie, 2009), but treatment methods differ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

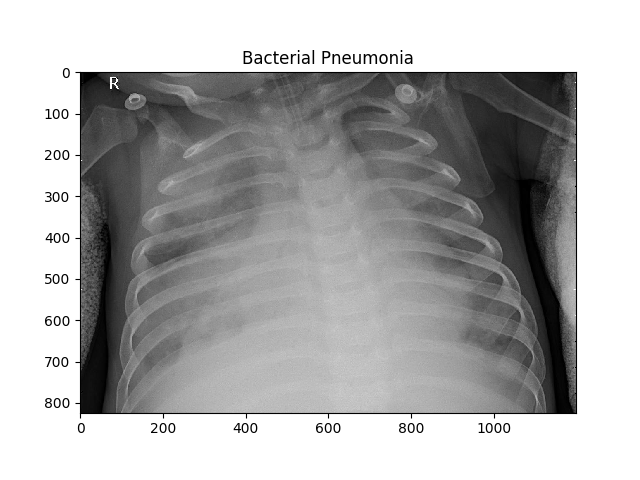
Because of the complications from incorrect diagnoses (false positive diagnoses of pneumonia can lead to wasting hospital resources, missing another potentially severe diagnosis; confusing bacterial pneumonia for viral pneumonia will lead to no treatment, and puts the child at risk for death) and the 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.

# 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 (X by Y) in size.

Below are example X-Rays from each class:





### 

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.

# Model Development

## Data Preprocessing

First, 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.

Also, we can make loading the data simpler using Pytorch’s ImageFolder data-loader, which 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. So, we move the data into 3 respective folders (Normal, Pneumonia-Bacterial and Pneumonia-Viral) to make data loading simple.

**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 data 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. (INSERT CODE BLOCK?). Given the train data distribution, the probability for viral pneumonia and normal images will be nearly the same, and be 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. This could be useful for training the model to distinguish viral and bacterial pneumonia, the pair of classes most difficult to distinguish.

In order to observe whether our data is balanced or imbalance; we use WeightedRandomSampler to ensure that each batch sees a proportional number of all classes. 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.

**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 so that the images were ready when loaded into Pytorch.

## Final Model Description

For training our dataset, we used Inception V3 and VGG16 two convolutional architecture models. 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.

## Training Process

We started with the instinct to use CNN for modelling. 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, ResNet 34, 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. The validation accuracy for ResNet with a single layer classifier seems to top around 0.8 before overfitting begins. Viewing the confusion matrix showed that the model was having trouble distinguishing viral and bacterial pneumonia, but had found some distinction between Normal and both types of Pneumonia. Therefore, we tried to rebalance the data so that the model saw the same amount of bacterial and viral images, and half as many normal images. This created some overfitting in our model, as the test loss began slowly rising while the training loss continued to decrease.

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. (KARTIK RESNET34 TRAINING)

This led to the crucial question we needed to answer to build a successful model; how can we separate viral and bacterial pneumonia in our model without overfitting or losing the distinction between normal and pneumonia cases? To answer this, we tried custom architectures built on top of ResNet18 and VGG16, such as the following:

Next we added inception V3 to RestNet and VGG16 to see if we can receive better results with only the original dataset instead of after data balancing. We use the similar augmentation with CNN model but added transfer.ColorJitter to make the picture more recognizable. In the beginning, we set up the parameters are 128 batch size, 0.005 LR, 0.001 decay and 25 epoch. Then we received the train accuracy was around 79% and the test accuracy was around 60%. After that 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 10 - 20 epoch. Following step, we tried SDG and Adam for the optimizar. Looks like Adam optimizar 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 tried vgg16\_bn() and the result doesn’t change significantly.

## **Loss Functions**

## **Optimizers**

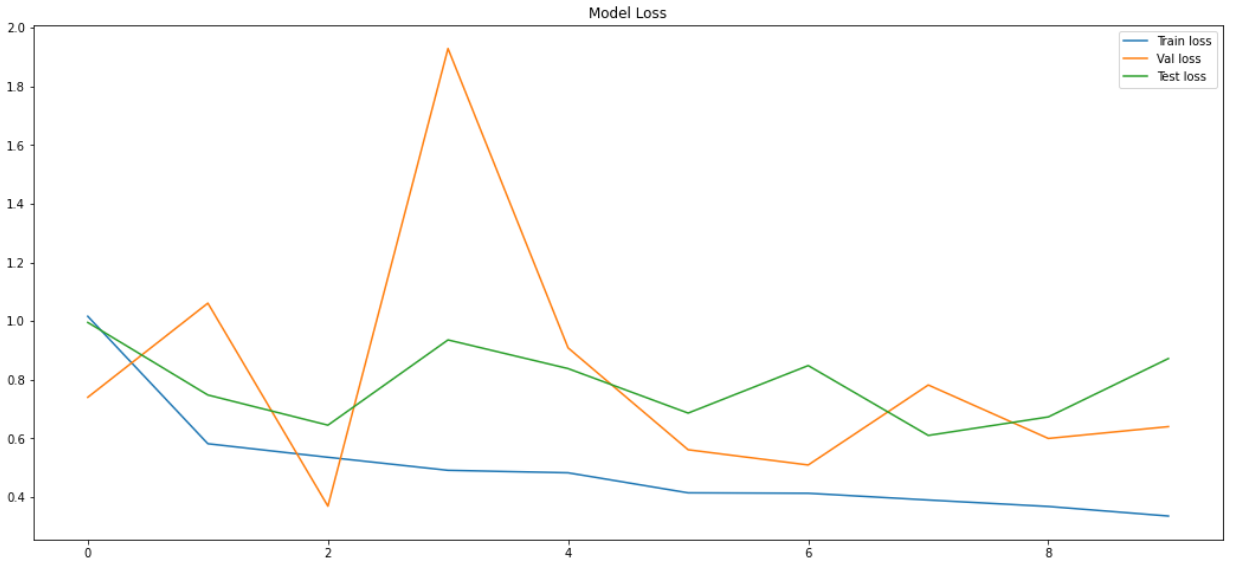
## Hyper Parameters

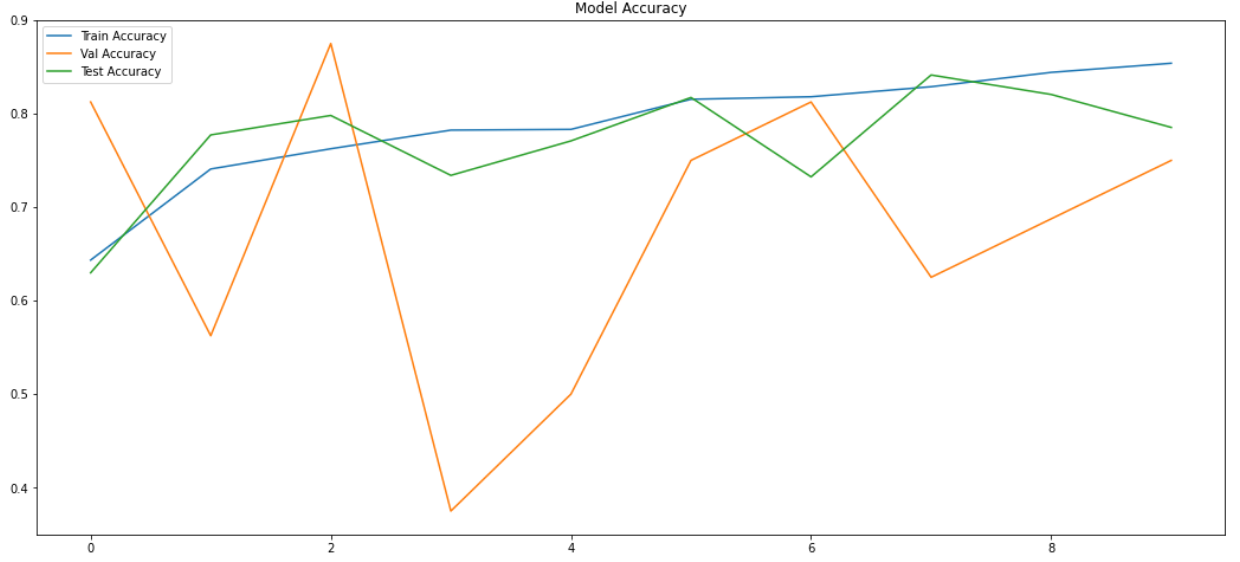
* Batch Size
* Learning Rate
  + LR Scheduler
* Dropout

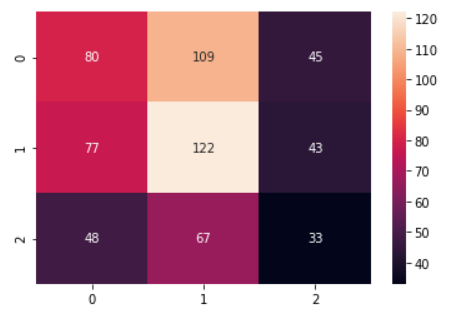
**Overfitting**

## Result

### Original Dataset







### Balanced Dataset

# 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.

Moreover, we will try to modify the ColorJitter parameter. ColorJitter can randomly change the brightness, contrast, saturation and hue of an image.

# Reference

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

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

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