

USING DEEP LEARNING TO DETERMINE CHEST X-RAYS

By Ryan Franco



December 15, 2021

DR. WANG

CS677

*The Description of the Problem*

Pneumonia is an illness that affects millions of people every year, in a spectrum that ranges from mild to severe. It is particularly deadly to children and elderly adults. However, it can be difficult to classify whether someone has Pneumonia or not. Diagnosis is usually based on symptoms, but the symptoms themselves are usually not enough. There are other viruses that exist that could show whether a person could potentially be infected with pneumonia or not. One clean-cut way to determine whether someone is fine is if they are examined internally through an x-ray machine. Therefore, it becomes necessary to be able to identify whether someone has Pneumonia from an x-ray image as validation. Machine Learning has led to advances in the field of medicine that allow it to become easier to figure out diagnoses. It is easy to collect data through images and train a model that could determine whether someone is sick. This program will attempt to check to see if someone is sick through a dataset that is given.

*Dataset used*

The dataset, based on people who have received x-rays from the children’s hospital in Guangzhou, China (listed at https://www.kaggle.com/tolgadincer/labeled-chest-xray-images) contains 5856 images in total. 5466 of them are for training the data, and 390 of them are for testing the dataset. The data appears to just need two columns for the data frame: The image itself, wrapped in a link, and a binary statement whether it is Pneumonia or not. Although the actual images are both bacteria and Pneumonia, we will show them both as Pneumonia in this example. Looking at the actual images, they can be a little tough to decipher for the average human to find out which one has it from black and white images. The Pneumonia in both the training and the test datasets is slightly higher which means the dataset is slightly imbalanced, but the real-life reason for that is because most of the patients who get screened are the ones who have symptoms, rather than those who are healthier than that.

*Candidate ML models*

The model that I have chosen is a straightforward Convolutional Neural Network. To get ready for it, I had to train the data. I used train\_test\_split on the images to randomly split the train 80-20%. Then, I used ImageDataGenerator to rescale the images, with a batch size of 32 images and a random number to generate from the images to train my model. I used something like what we used in Lab 2 for CS677, with the activation of the last layer being changed into a sigmoid. The reason why this was done was because this model is going to be binary, as it is going to produce either 0 or 1. The other thing I changed was something which reflects the image size that I planned on using, a 224 x 224 x 3 image. This is largely to start with conserving the value, although it made the run time a little longer even with the validation steps and epoch steps reduced. The number of parameters for this model is 5,982,241. It uses 2 convolutions and 2 poolings. After the first epoch, the binary accuracy was 82%, and then it would increase to around 93% by the tenth one. It took a grand total of about 17 minutes for this program to compile. After testing the accuracy on the train side of things, it would lead to 95% accuracy.

*Baseline Models*

To compare my model, I used a baseline model and transfer learning to figure out what the accuracy of one would be. I chose VGG-19, a model based on having an extensive amount of Convolutional Networks. It had about 20 different convolutional networks. I used transfer learning, which is the process of using one problem to be able to solve another. In order to get it to work with my model, I replaced the output and gave it the same output as my previous model. It constantly goes through multiple convolutions and pooling, but without any real padding. There were 21,630,145 parameters in this specific model. After the first Epoch, the binary accuracy was about 72%. I was using the adam optimizer. It would eventually increase to about 92%. The one tricky thing with this model is that it would take a very long time to compile. Each epoch was about 26 minutes, and it had 10, for a grand total of 4 hours to compile. My base model is strong, but the model I presented and fitted with my own data was a little bit stronger, which is a good sign.

*Metrics*

Lastly, we will talk about the performance metrics we will use to figure out the effectiveness of this model. We start with the base accuracy score of the test after converting it to binary. The result of the first model was 82%. The VGG model came up as 81%. Another way to determine the model is through a F1 score. You want to figure out the precision and the recall. Precision is the number of true positives divided by True and False Positives. Recall is number of true positives divided by True and False Negatives. F1 score is when you use 2 \*(Recall\*Precision)/(Recall+Precision). I used a classification report to get the F1 score on each model. Precision seemed relatively strong while the recall could use some fine tuning. The last metric I wanted to look at was the Area Under the Receiver Operating Characteristic Curve. Basically, it tests to see if it can read the accuracy of the false positives in the same way of F1 Scores are. In this kind of problem, a false negative is way worse than a false positive. So having a low recall and an AUC isn’t particularly great. If there’s one way that this model could potentially improve, it would be through that element of trying to improve False Negatives. However, it is a great start and I feel like it can be even more effective later on.