Final Project Report for CS 184A/284A, Fall 2019

Detecting Pneumonia from Chest X-Rays Using Deep Convolutional Neural Networks

Raghav Verma, 67109270, raghavv@uci.edu

Code: https://github.com/rghvv/DCNN-Pneumonia-Diagnosis, **Dataset:** https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

1. Introduction and Problem Statement

Pneumonia is a very common disease that affects the lungs and can be deadly to if left unchecked. The Pneumonia Chest X-Ray dataset contains over 5,800 black and white images of chest x-rays split into two classes - 'pneumonia' and 'normal'. The problem is to build a model capable of accurately classifying any given image of a chest x-ray on whether it shows symptoms of pneumonia or not.

2. Related Work

Convolutional neural networks have been proven to be successful in analyzing medical images in the past. Models have been able to reach an AUC of up to 0.93 [1] on diagnosing radiological pathologies on datasets such as CheXpert [2], outperforming most professional radiologists. This has can be leveraged to assist doctors in providing faster and more accurate diagnoses. It has numerous real world applications in time critical situations such as emergency wards and trauma centers as well as resulting in better patient health outcomes.

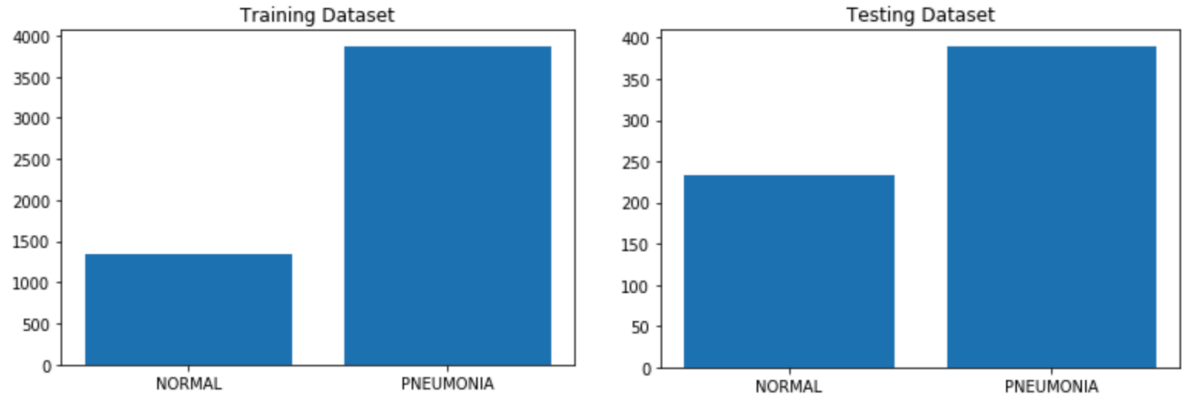
My model attempts to use convolutional neural networks and past knowledge of what has worked in medical image analysis to classify images of chest x-rays on whether they have pneumonia. I evaluated my performance on the testing dataset using various metrics including precision, accuracy, recall and F1 score.

3. Data Sets

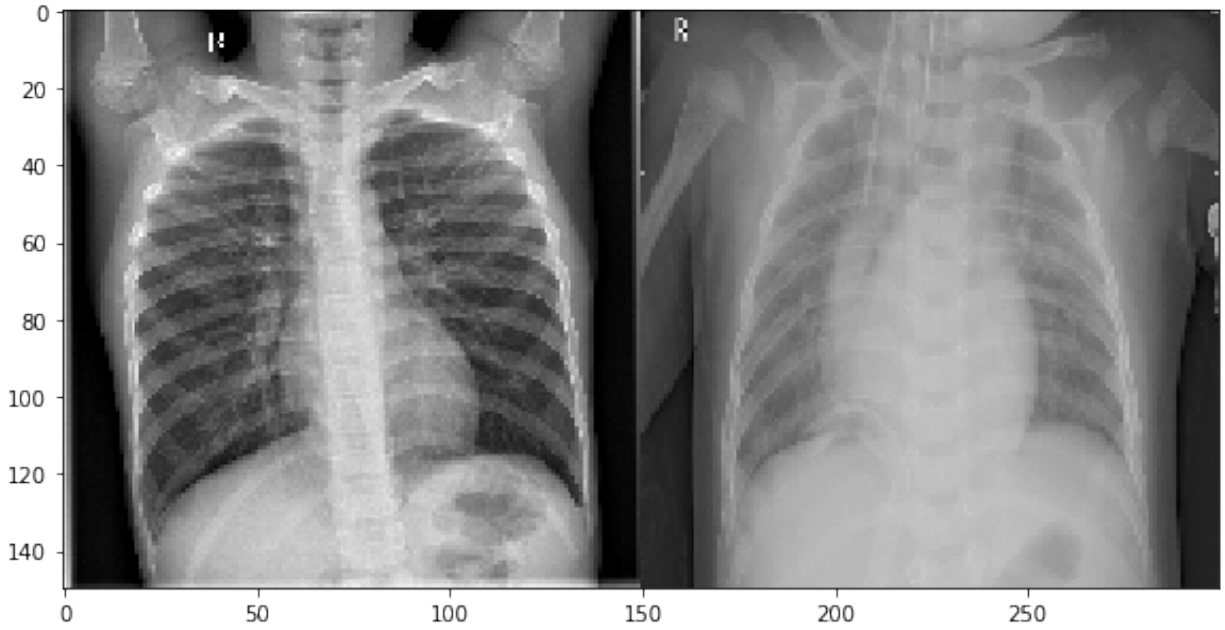
I used the "Chest X-Ray Images (Pneumonia)" dataset from Kaggle [3].

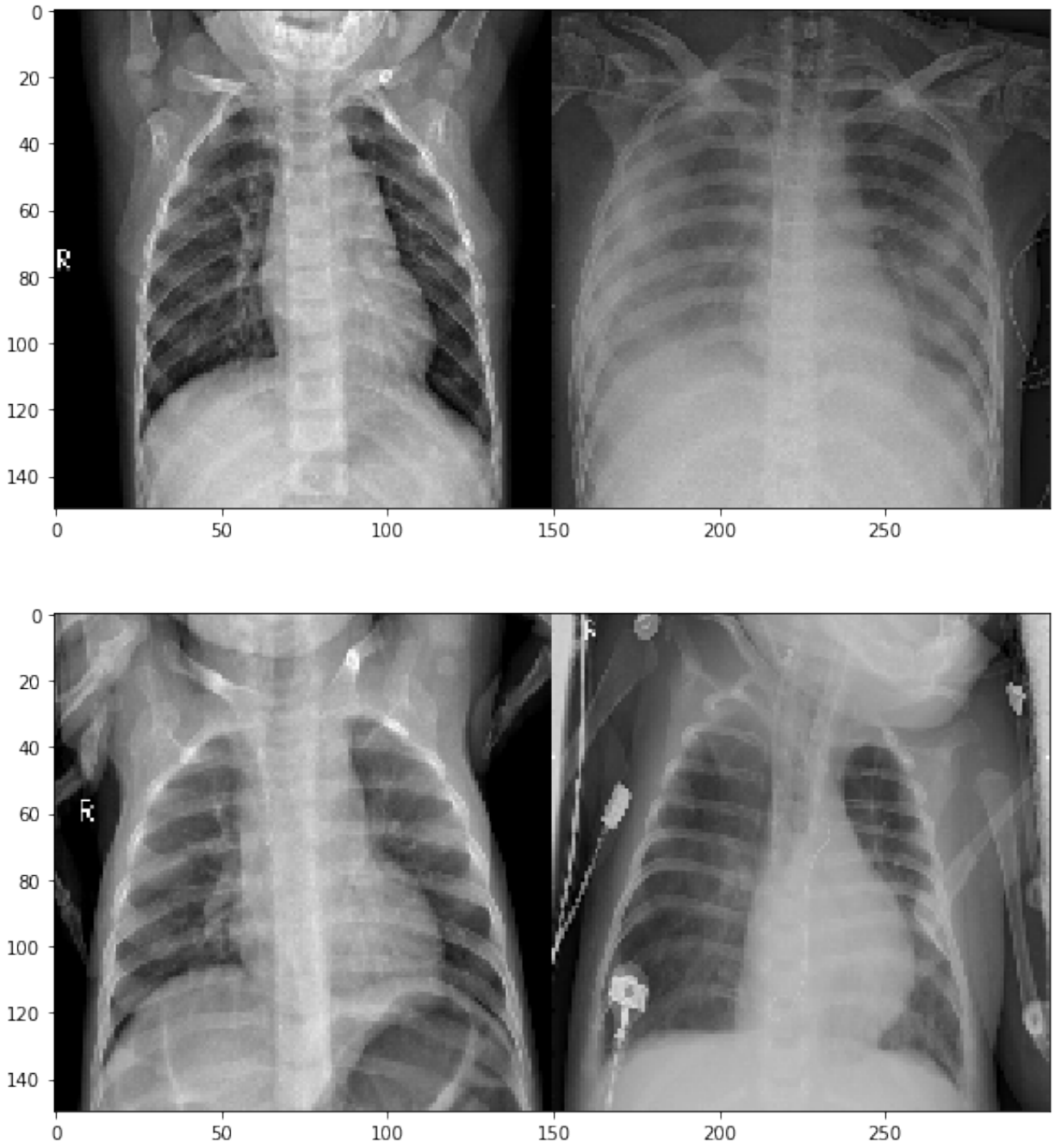
The dataset is sourced from pediatric patients between one and 5 years of age from the Guangzhou Women and Children's Medical Center, Guangzhou, China. All imaging was performed as part of patients' routine clinical care. For building the dataset, all chest radiographs were initially screened and all low quality or unreadable scans were removed. Two expert physicians then graded and verified the diagnoses of each scan, and a third expert did the same for the testing set to account for grading errors.

The size of the dataset is 1.2 GB and it contains 5,836 JPEG images of chest x-rays. The images are black and white, 1000px in each dimension, and each image is in one of two classes - ‘pneumonia’ and ‘normal’. The dataset is has a split of 5,216 training, 16 validation, and 624 testing images and has a significant class imbalance in the training (1,342 ‘normal’; 3,876 ‘pneumonia’ images) and testing (234 ‘normal’; 390 ‘pneumonia’ images) splits.



From visual inspection, images that do not indicate that the patient has pneumonia show clear lungs. The images that do indicate that the patient has pneumonia show slightly opaque lungs that are not as discernible. The images below demonstrate this phenomenon - healthy lungs are on the left, and lungs exhibiting symptoms of pneumonia are on the right.





4. Description of Technical Approach

Since convolutional neural networks are the standard way of interacting with image data, where a window slides over and interacts with subsets of pixels of an image repeatedly. I began by examining various deep models that have been successful for image classification such as Google's InceptionV4, Oxford's VGG19 and Microsoft's ResNet. Transfer learning, or fitting a well-performing pre-trained model

to the current data, is a viable strategy for achieving good performance. Implementations exist that use transfer learning to achieve accuracy as high as 0.98 for this dataset. However, this strategy is not as interesting as almost all of the work of constructing and training the network is abstracted out. Hence, I opted to create my own neural network from scratch.

The dataset has significant class imbalances so I began by using the ImageDataGenerator class from Keras in Python to generate samples that are similar to the training data. The images are augmented in various ways, including rescaled to 255th of its original size, zoomed in and out of by a factor of 0.3, and flipped horizontally. I also rescaled the image sizes to 150px on all sides as processing 1000px would be too computationally intensive.

The model has 30 layers consisting of 5 convolutional blocks, each with a convolutional layer with a 3x3 filter, a max pooling layer for reducing dimensions and a batch normalization layer for normalizing the input and scaling the activations. Following are flatten layers to collapse the input into a one-dimensional vector, fully connected 'dense' layers and dropout layers to prevent overfitting.

The model has a batch size of 32 and is trained over 12 epochs with two functions that are called after every epoch - 'ReduceLROnPlateau' which decreases the learning rate by a factor of 0.3 when the training accuracy does not improve for two epochs, and 'ModelCheckpoint' which saves the iteration of the model with the best validation accuracy for evaluating the model's performance on the testing dataset.

Due to the class imbalances, various metrics such as precision, recall, accuracy, and F1 score are used for evaluating the model's performance.

5. Software

Major software dependencies are:

- Python 3.5+
- Keras for creating the neural network
- Numpy for manipulating data and math
- Scikit-learn for metric selection
- Pandas for reading data from disk
- Matplotlib for visualizing performance
- Mlxtend for plotting the confusion matrix

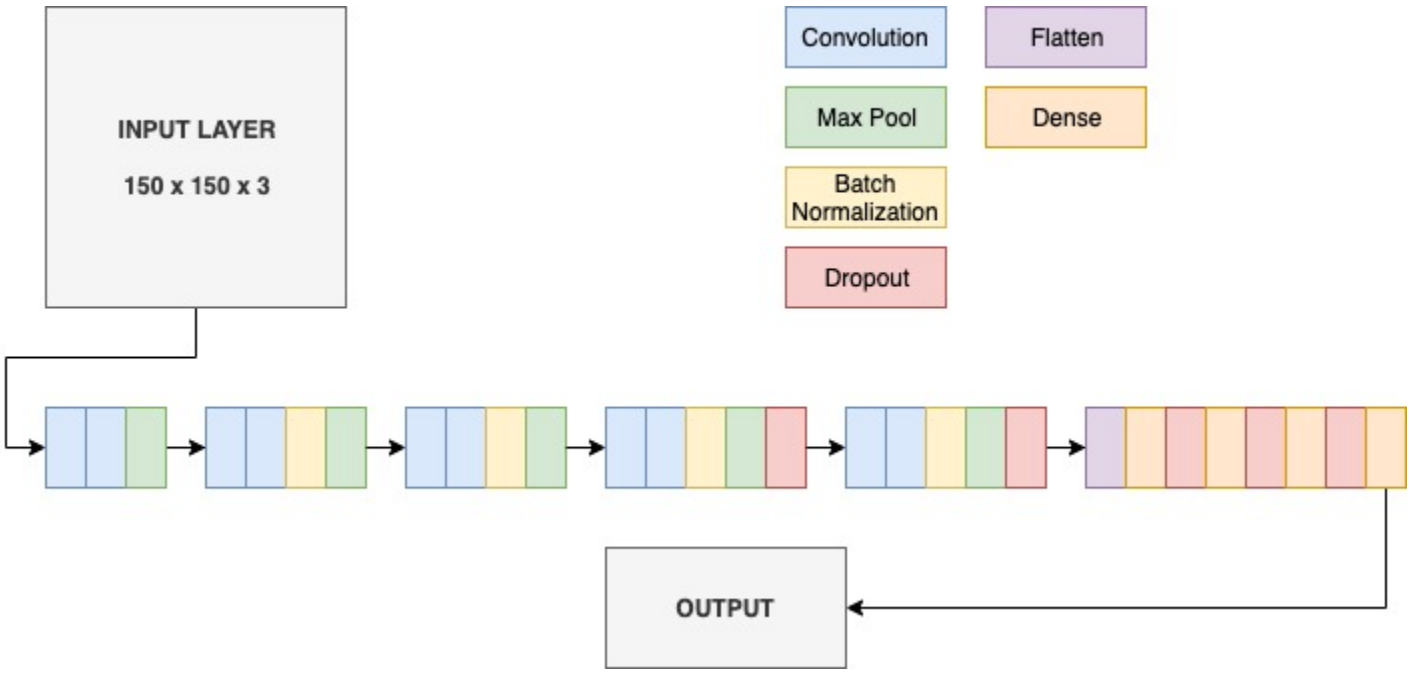
I chiefly looked at the Keras code repository [4] for finding information about creating custom convolutional networks such as creating a convolutional base, hyper-parameter tuning and metrics for evaluation.

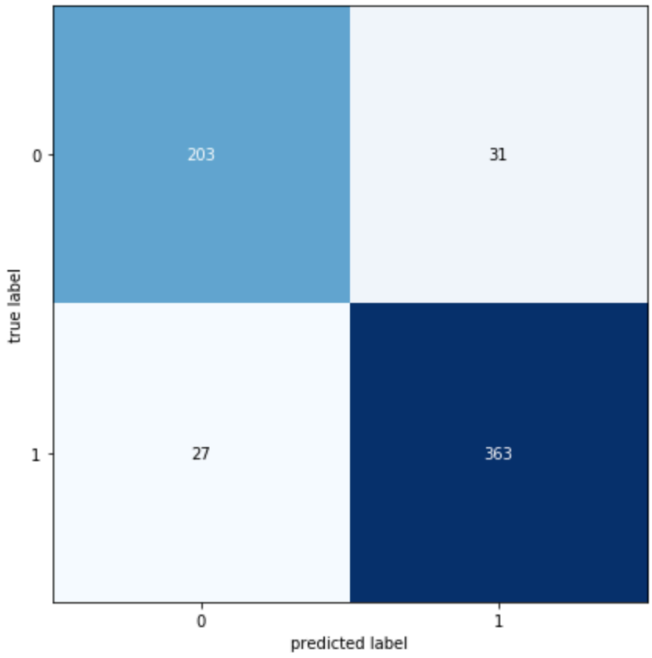
6. Experiments and Evaluation

Due to the class imbalance in the testing set, only accuracy was not a sufficient metric for evaluating the performance. After adding precision and recall, I rationalized that false negatives (or misdiagnosing someone as not having pneumonia when they actually do) was intuitively the worst case scenario. Hence, false negatives had to be minimized and recall maximized (since recall accounts for false negative in its denominator). I decided that optimizing the neural network for maximum recall would result in the best performing model.

However, while I achieved a very high recall rate (0.96), the neural network's accuracy (0.69) and precision (0.71) were mediocre. I noticed that there were just 6 false negatives (0.9% of the testing set) but 168 false positives (26.92% of the testing set) from the resulting confusion matrix, when it became apparent that optimizing for recall was not an effective strategy. By having a high confidence interval for classifying an image as negative (or 'normal'), the model decreased both false and true negative predictions. So, although recall is an interesting metric, the model must be evaluated holistically using a variety of metrics.

After significantly increasing the overall complexity of the model by adding layers and tuning hyper-parameters, it achieved a validation accuracy of 0.8721 and training accuracy of 0.9459. It also got an F1 score of 0.9260, accuracy rose to 0.9070, precision to 0.9213. Even though recall fell to 0.9307, the final model is demonstrably better than any previous iterations.





Accuracy = 90.705%

Precision = 92.132%

Recall = 93.077%

F1 Score = 92.602%

Training Accuracy = 94.594%

Prior to reaching this solution, I experimented with various other strategies and combinations of training parameters. These included image augmentation techniques such as stretching, vertical flipping, rotating, blurring, and histogram normalization; batch sizes of 16 through 256; choosing RMSprop, Gradient Descent, SGD as optimization algorithms; and picking weights. The changes that contributed to most to increasing the model's performance are:

- Adding Batch Normalization to each convolutional block.
- Adding dropout layers to final two convolutional blocks.
- Replacing some Conv2D layers, the traditional convolutional layer, with SeperableConv2D layers which is a depthwise spatial convolution followed by a point wise convolution that blends outputs, resulting in faster operations.
- Choosing Adam as the optimization algorithm.
- Choosing cross entropy as the loss function.

7. Discussion and Conclusion

Given more time, some ideas I would like to explore are:

- Investigating the specific samples that are being classified incorrectly.
- Implementing a blended/stacked ensemble of models, including pre-trained and novel models as ensembles of multiple models are often better than any single model.

- Finding more/better data: Since the dataset only contains samples from a specific region and from children between the ages of one and five, it is not representative of the larger world. Labelled images chest radiographs symptoms of diseases other than pneumonia can be used to construct a model that can identify the presence of many diseases. Given enough data, an accurate generalized chest radiograph-analyzing neural network can be created.

Despite having a relatively simple architecture the model is able to achieve a good performance on the given dataset, but there is still room for improvement - both in the model's performance and the scope of the project itself.

References:

[1] "*Interpreting chest X-rays via CNNs that exploit disease dependencies and uncertainty labels*", Pham, Hieu; et al. https://arxiv.org/abs/1911.06475

[2] "*CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison'*, *Stanford ML Group*. https://stanfordmlgroup.github.io/competitions/chexpert/

[3] Kaggle Chest X-Ray (Pneumonia) Dataset. https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

[4] Keras User Guide. https://www.tensorflow.org/guide/keras/overview