Exploring Pneumonia Detection with Chest X-Ray Images

Aaryan Samanta
aaryan.samanta@gmail.com

This project uses a Convolutional Neural Network to classify chest X-ray medical images as pneumonia or normal. The purpose of using AI is to allow more efficient and effective detection. The data set used here is a public chest X-ray dataset, split into training, validation, and test data sets. The model is trained with TensorFlow and is tested for its accuracy and performance on new images of chest X-rays. The project simulates how doctors may use AI for X-ray scans to diagnose diseases.

Here, I used a CNN which is capable of binary image classification. I trained the model for classification of two image classes effectively based on image features with the use of deep learning. This kind of problem is what computer vision excels at. Training a binary classifier is an important step towards the creation of intelligent systems that can perceive visual information.

The database for this exercise was split into three folders as train, val, and test. Within each folder, two subfolders contained the name of each class (labeled as 0 and 1 for pneumonia and healthy). The data was loaded by TensorFlow's ImageDataGenerator to provide data augmentation and preprocessing. The images were resized to 150x150 pixels and normalized with rescale=1./255, so the model was able to more easily work with them. To improve generalization and to counter overfitting, zoom augmentation (zoom_range=0.2) was also used during training. The test set was 234 class 0 and 389 class 1, a fairly small class imbalance that would later affect the performance of the model.

The model's architecture had eight trainable layers. The model began with three convolutional layers of 32, 64, and 128 filters, respectively, with each of them having max pooling. The layers progressively pulled more and more abstract visual features out of images. After flattening the subsequent feature maps, there was a single dense layer containing 128 units with ReLU activation followed by an output layer with sigmoid activation function. It was trained on Adam optimizer and binary crossentropy loss, and the accuracy was tracked as the primary performance measure. It was also trained for 1,3, and 5 epochs to observe the impact on generalization performance.

Interestingly, performance metrics were the same for training for 1 and 3 epochs, which means that overfitting occurred very early. Test accuracy in both cases was 82%. Confusion matrix showed that the model was performing wonderfully on class 1 images but was appalling with class 0. Exactly, it identified only 141 out of 234 class 0 images correctly and categorized 93 wrongly. The model performed extremely well on class 1 and identified 369 out of 389 and categorized 20 wrongly. However, when using 5 epochs, although the accuracy was higher, the

model began overfitting too much, and therefore performed very poorly on the validation set, which I will discuss shortly.

To also look at model performance, I plotted training and validation accuracy and loss versus each epoch with Keras history object data. Training accuracy increased gradually, while validation accuracy peaked early and validation loss began to increase after the first epoch. This is characteristic for overfitting: the model does well on the training set but cannot generalize to new data. These learning curves were very insightful on what was going wrong and what the limitations of the model were. One of the main reasons for overfitting is that the original data being used wasn't balanced. Since there were 234 samples from class 0 and 389 samples from class 1, the model developed an algorithm bias, which caused it to overfit and perform poorly on new data.

From these findings, some tactics could be used to improve performance. First, I added dropout layers to prevent overfitting as it would randomly turn off neurons when training, causing the model to learn more complex features. Second, stronger data augmentation (random flip, rotation, and shift) would simulate a larger dataset. Finally, early stopping training can also be done to prevent the process from continuing once the model begins memorizing the training data. Specifically, when the accuracy for the validation data set starts decreasing, I used a callback function to stop the program early.

One thing that I learned through this project is that CNNs can learn visual patterns well in images and how important monitoring model performance and accuracy is. What came as a greatest surprise to me was how quickly the model started overfitting the data — even after just one epoch. I experimented with different parameters and layers until I found what worked best to prevent overfitting and to increase accuracy.

For the most part, the project was an excellent experience and I learned how to train, test, and debug deep learning models. It also showed that for high performance, it's more optimal to split the code into parts like preprocessing, parameter tuning, and analysis, rather than merging everything into one group.

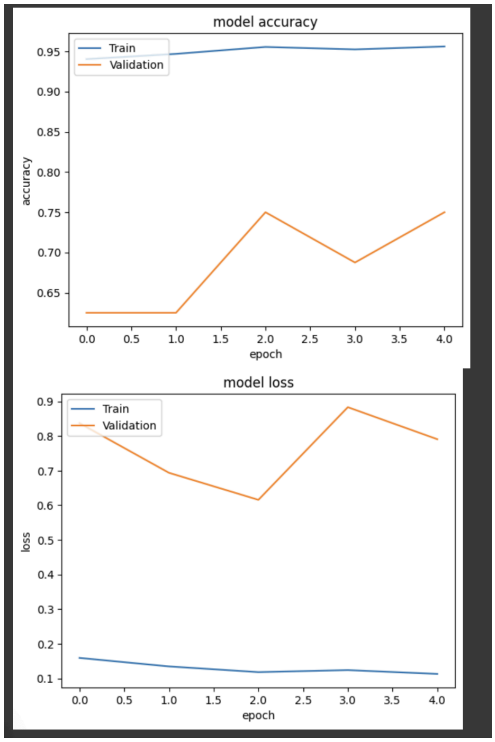

Figure 1

1 epoch

Confusion Matrix:

[[141 93]
[20 369]]

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.60	0.71	234
1	0.80	0.95	0.87	389
accuracy			0.82	623
macro avg	0.84	0.78	0.79	623
weighted avg	0.83	0.82	0.81	623

3 epochs

Confusion Matrix:

[[141 93]
[20 369]]

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.60	0.71	234
1	0.80	0.95	0.87	389
accuracy			0.82	623
macro avg	0.84	0.78	0.79	623
weighted avg	0.83	0.82	0.81	623

5 epochs

Confusion Matrix:

```
[[158  76]
 [  2 387]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.68	0.80	234
1	0.84	0.99	0.91	389
accuracy			0.87	623
macro avg	0.91	0.84	0.86	623
weighted avg	0.89	0.87	0.87	623

Figure 2