**Pneumonia detection report**

With the given data, the images were already separated into 2 categories. Normal and Pneumonia, with the Pneumonia containing bacteria and virus categories. There were 273 images of normal lungs, 530 images of bacteria and 187 images of virus. With the assumption that this was all the dataset that I am given and allowed to use, I separated the dataset into 7:3 ratio, with 7 being the training dataset and the 3 being the testing dataset. These separated datasets can be found under heartlab/chest\_xray/TEST or TRAIN.

The first task that I undertook was utilizing the given code so it could separate and get the training and testing dataset. I modified the original create\_datasets() into two separate functions, create\_training\_datasets() and create\_testing\_datasets(). These functions would output an array which contains the processed image and the integer no. associated with the class, with normal = 0, virus = 1 and bacteria = 2. With now the datasets imported, I wrote some code to process the data. Since the array from the create\_datasets() function outputted the image data and the integer label associated with the class, I separated the image and the label and appended it to new lists. Then the image data was normalised and resized for deep learning and the labels were turned into binary class matrix using the to\_categorical() function so it could be used in categorical cross entropy in our neural network. The neural network used in this test was found online (see code) and was slightly modified for my needs.

With the images and the labels separated and ready to be inputted, I decided to use the ImageDataGenerator to generate shifted/modified images of the dataset. As the dataset was small, I believed that using the ImageDataGenerator would help with the generalisation and try combat overfitting issue. This ImageDataGenerator was also used to create the validation dataset by modifying the test dataset as I did not want to separate the already small test dataset into even smaller test/validation datasets.

Even with the modified dataset, the model still struggled, giving a final accuracy of 17 – 20% and showing signs of overfitting. I tried to change the epoch and the batch size and got minimal results. I decided to use class weights, as there was a massive imbalance between the 3 classes (out of 717 images total, normal had 273, bacteria had 530, and virus had 187). The class weights were implemented by dividing the no. of total samples to the number of samples in that class. For example, bacteria would be 717 / 530 = 1.3528 class weight. These weights were put into a python dictionary and inputted to our model.fit() function. Also, the learning rate was reduced by using the ReduceLROnPlateau and monitoring the validation accuracy, as our model tended to stagnate in validation accuracy with each epoch. When the validation accuracy would stagnate, the learning rate would be reduced to combat the overfitting issue.

Even with all these measures in place, unfortunately I was not able to get an accuracy on the testing dataset greater than 53%. I was not able to put a lot of time into fine tuning the model any further due to time constraints. Also, I was not able to take advantage of cuda as my laptop had no GPU and it would take close minute per epoch while training this neural network. From the graphs below, you can see that as the epoch increases, the loss decreases while the validation loss remains around the same. The accuracy increases and the validation accuracy increases up to epoch of 2 and then remains constant at 0.5347. This shows that our model is potentially overfitting on the training dataset and not able to generalise / classify well on the previously unseen datasets.

Here are the results:

Loss of the model is = 1.1054431200027466

Accuracy of the model is = 53.474318981170654 %

Overall, the overfitting issue was a big problem while completing this test. I tried altering the optimiser, the neural network, activation functions, batch size, epoch etc, but the best way to increase the accuracy would be having a larger dataset. By no means I am an expert on machine learning, and I may have made a mistake in one of the parameters. However, if the dataset is increased, the neural network could have more training data which could help with generalisation and overcome the overfitting issue. Also, the division of testing/training data could have been done more randomly which may have helped with this issue. It may be that these sets of training/testing data have some sort of bias which may impact the accuracy of this neural network.