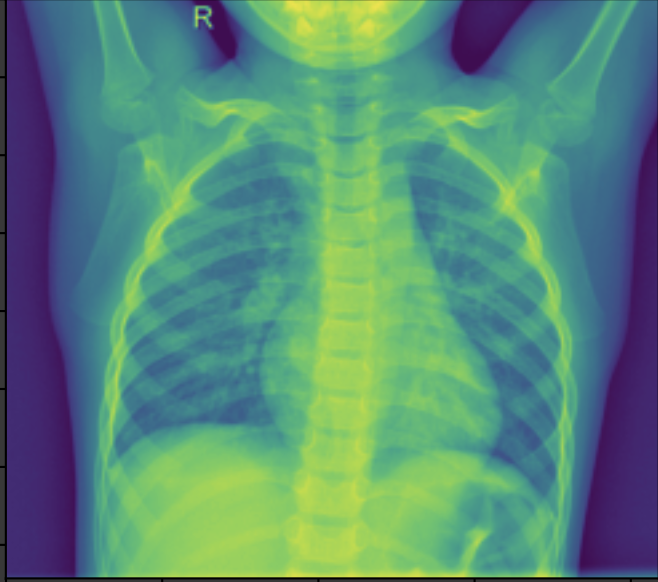
**Final Report**

**Business problem and application**

Recently, identifying medical diagnoses and treatable diseases by image-based deep learning has become more and more critical. Our project will focus on the similar topic, predicting normal or pneumonia based on their X-ray images. This is important because, the WHO estimates that over 4 million premature deaths occur annually from household air pollution-related diseases, including pneumonia. For these populations, the accurate and fast diagnosis means everything. It can guarantee timely access to treatment and save much-needed time and money for them. The implementation can support hospitals and doctors by preselecting the x-rays, but this will only be possible for a reliable and highly accurate algorithm.

For our project, we will build an algorithm to automatically identify whether a patient is suffering from pneumonia or not by looking at their chest X-ray images. The more accurate our model is built, the more time and money to save for this group of people.

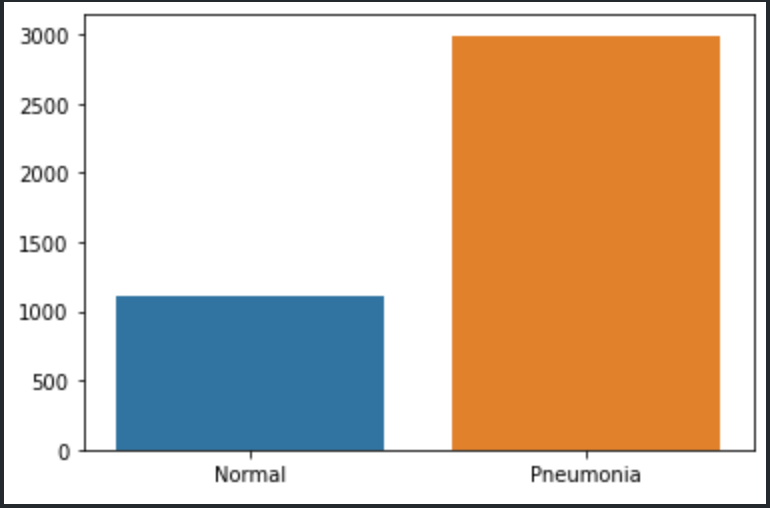
This type of model, and machine learning in general have great value for the health care sector. The ability to have a second opinion when it comes to diagnosis improves patient outcomes. On the business side, there is great opportunity in selling these tools to health care providers. Also, a software product that includes deep learning algorithms provides a win-win to pharmaceutical companies who want to sell more prescriptions faster. Better patient outcomes leads to more customer satisfaction after someone with pneumonia is diagnosed and prescribed a treatment. In addition, as the model is exposed to more training data, a pharmaceutical company can potentially use this data and the accuracy rates to improve treatments, the model, and their understanding of the disease.

**Data** 

We choose the Chest -Ray Images [dataset](https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia) from Kaggle. The datasets contain three folders: train, test and validation, and each folder has x-ray images in two categories: normal and pneumonia. In total, there are 5863 X-Ray images (JPEG). All chest x-ray images were obtained from the Guangzhou Women's and Children's Medical Center, and the images included the anterior chest and posterior back of patients aged one to five years. For better analysis of chest x-ray images, all images have undergone an initial quality control screening and low quality or unreadable scans have been removed.

**Analysis methodology**

*We plan to use CNN, classification model and Tensorboard to conduct the analysis. Data processing and EDA will also be applied. In the end, we will check the accuracy for our prediction.*

*A convolutional neural network (CNN) is a deep learning algorithm that takes an input image, assigns importance (learnable weights and biases) to aspects/objects of the image, and is able to distinguish the differences among them.*

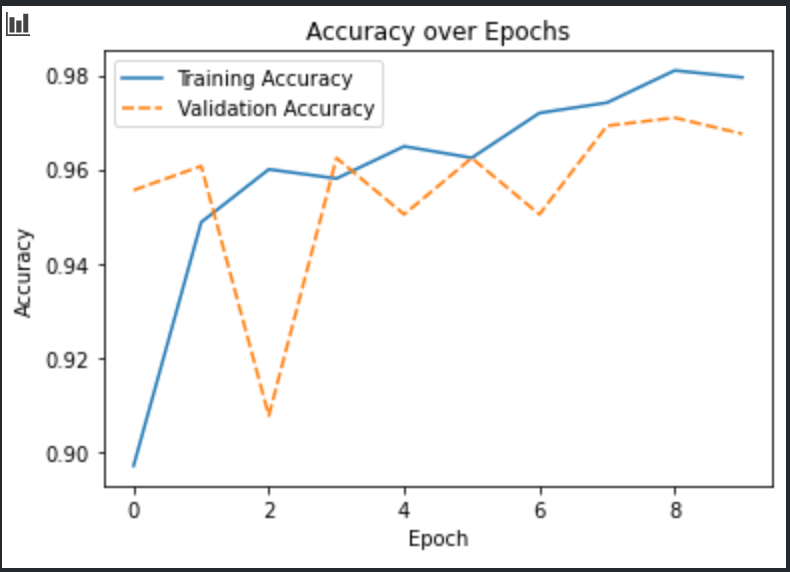
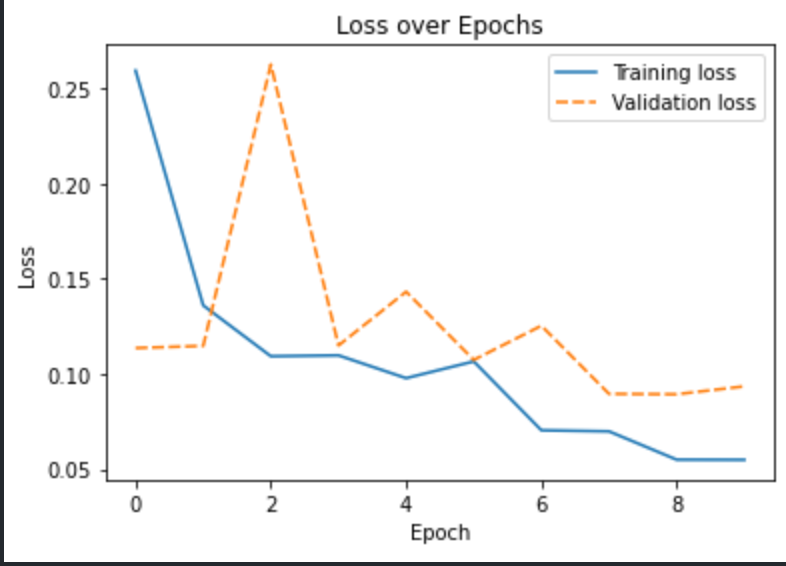
Firstly, we wanted to get an overview of our data and started with some simple EDA. We looked at the distribution of our dataset and found out it’s unbalanced. We can see in the graph on the right side that there are more pneumonia cases than normal cases in general. We also checked for the sizes of the pictures and saw that we would need to reshape the images for our network. For example, under the train dataset, the height and width for normal cases are in the range from 1100 to 1500 and 1300 to 1800. And pneumonia cases’ height and width are in the range from 600 to 900 and 900 to 1400. The predefined train, validation and test datasets didn’t have the split we wanted so we combined the data and used a 70/10/20 split. For our first try we reshaped the image to a size of (224, 224) but then we decided to choose a bigger size (800, 800) because the details are important to detect pneumonia. Before building our own network we wanted to see how the pretrained MobileNet would perform on this task. Afterwards we also built our own convolutional networks. Starting with a simpler one and then improving the complexity.

**Analytical findings**

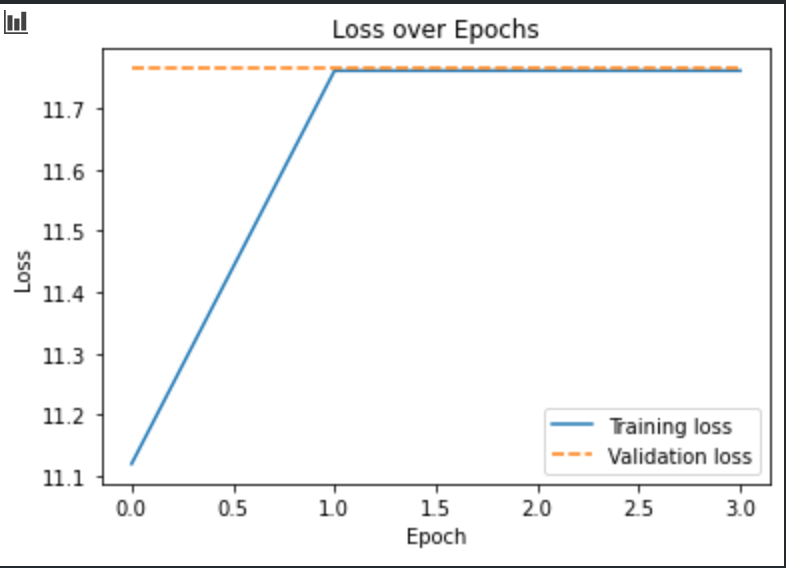
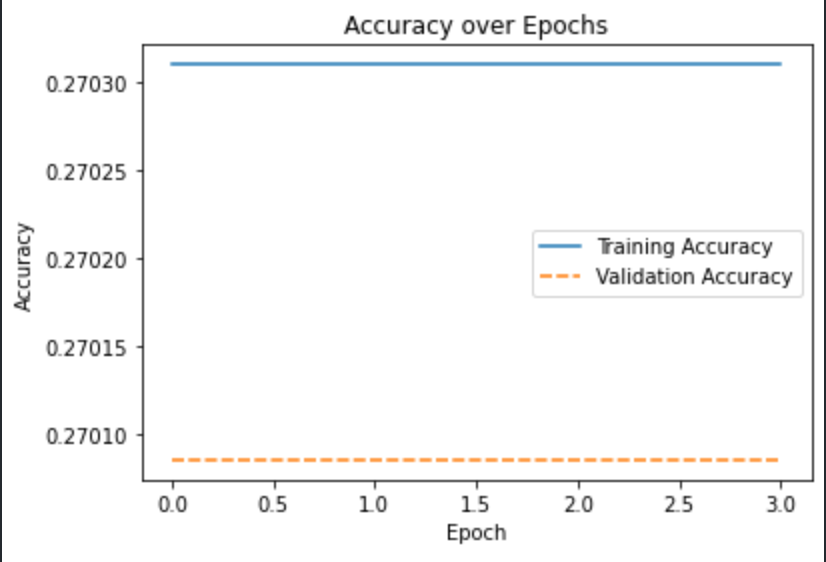
After creating multiple models and improving those, we still got our best result from the pretrained MobileNet with an accuracy of 96% on the test data set. This network is by far the most complex and the one pre-trained with the most images. The networks we built ourselves achieved an accuracy around 73% which is still a decent accuracy keeping in mind that we only have over 5000 images. A lot of our models underfit and got stuck at an accuracy of around 27% after one epoch. We tried to solve this problem by increasing the learning rate in case this underfit was due to a local minimum. We also tried to change the optimizer. We still got some networks stuck at the same accuracy. It’s also interesting that our very first try actually got an accuracy of 73% for the test data set and following models where we tried to improve the model by adding more layers got a lower accuracy of 27% on the test data set. (This model seems to be stuck at a local minimum.) We ran the same architecture just with more epochs again and the model got an accuracy of 73%. After adding another convolutional layer to the network and decreasing the learning rate our accuracy is stuck again at 27%. For the next model we only changed the loss to binary cross entropy loss and it improved the performance again to 73%. In the 6. model we added four more convolutional layers and increased the learning rate again, our accuracy wasn’t different from the one before. It seems like either there is another local minimum at 73% or this is the best accuracy we can get with our data (unlikely, since we got an accuracy of 96% with the MobileNet, but it could be the best performance we can reach with the level of complexity we used). We were wondering if the small size of the images had an impact on the classification so we decided to increase the picture size by almost 4 times. We used one of our more complex models again and decreased the learning rate. The accuracy got stuck at almost 73% after the second epoch, so we decided to increase the learning rate again. Unfortunately, it didn’t have an impact on the accuracy at all. As one of the last steps we decided to use Tensorboard to visualize the weights during training. We looked at some of the information we got from the Tensorboard but due to the time frame of this project we were not able to implement anything from the results of the Tensorboard.

Lastly, we tried a different optimizer with a small and high learning rate and were able to improve our training accuracy by 0.5% to 73.5%. We also rotated some pictures to get an even better accuracy, but our accuracy went back down to 27%.

In addition, we wanted to look at the loss and accuracy of the models over the epochs and decided to compare our best performing model, the MobileNet, with one that is stuck at 27%. The two graphs below show the accuracy and loss over the epochs of the MobileNet. The lines of the training data are increasing steadily for the accuracy and decreasing for the loss. The lines for the validation set look very different from the ones of the training set. But overall we can see that the accuracy increased a bit and the loss decreased.



The next two graphs show the accuracy and the loss for one of our models that got stuck at 27%. For the accuracy graph we can see that both the training and validation lines are completely horizontal at around 27%. The accuracy neither increases or decreases. For the Loss over the epochs the lines for the validation data is again a horizontal line, but the line for the training data increases sharply between Epoch 0 and 1. These two graphs are a good example of how you don’t want your accuracy and loss to look like.



**Results Summary:**

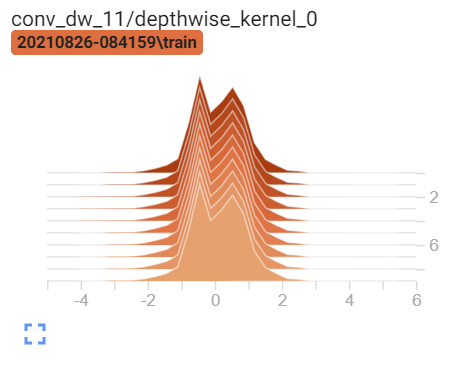
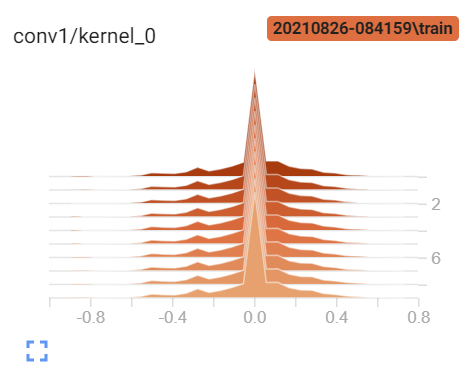
Below is a table summarizing the results of several models that we ran. As mentioned in the previous section, our best performing model out of sample is the MobileNet model with frozen layers.

| **Model** | **Train** | **Validation** | **Test** |
| --- | --- | --- | --- |
| **Mobile Net Frozen** | **0.982** | **0.968** | **0.964** |
| **Mobile Net Unfrozen** | **0.951** | **0.937** | **0.931** |
| **Smaller Network** | **0.729** | **0.729** | **0.729** |
| **Additional Convolutions** | **0.457** | **0.270** | **0.270** |
| **Adagrad optimizer** | **0.270** | **0.270** | **0.270** |
| **SGD Optimizer and sigmoid** | **0.729** | **0.729** | **0.729** |
| **Smaller LR** | **0.270** | **0.270** | **0.270** |
| **Larger LR** | **0.729** | **0.739** | **0.729** |
| **Larger Image Size** | **0.734** | **0.625** |  |

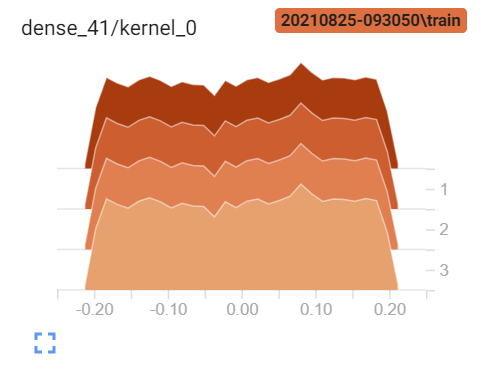
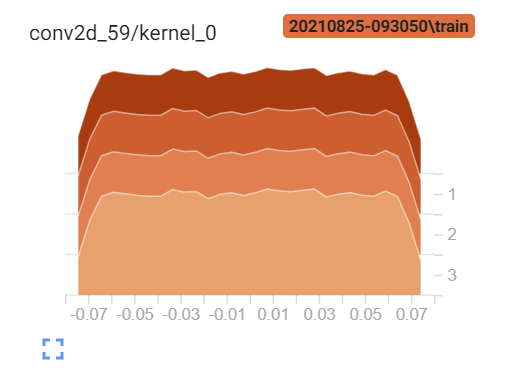
**Implementing TensorBoard**

TensorBoard is a tool for visualizing the values of weights in a trained neural network model. Based on the results above, we felt that we may be underfitting, or experiencing dead neurons that would lead the network to no longer learn more after the first epoch. Below are a few charts that we pulled from TensorBoard, showing the distribution of the values of the weights for a few of the convolutional layers. Overall, we felt that the values for the weights were much larger in the MobileNet model compared to the custom model using the RMSprop optimizer. We feel that potentially, these very small positive and negative numbers could be causing the model to not improve after the first epoch. Also, the fact that the distributions seem to be very flat in the custom model indicates that there is not a lot of information contained in the weights. We would expect to see more peaks and valleys if a layer is passing along useful signals. In the future, we could use this information to design a better custom network that could approach or surpass the accuracy that we saw with MobileNet. Potentially, adjusting the network architecture, trying different optimizers, or changing activation functions could address these problems.

**MobileNet - frozen layers**



**Custom Model - RMSprop and additional convolutions**



**Conclusion**

Overall, we can conclude that it is possible to create different NN with relatively good performances. Our best network reached an accuracy of 96%. But putting it in context, the performance of the network is not nearly good enough. The (training) data is too small to get a stable and acceptable result/accuracy. The field of which this algorithm would be used is the medical field and an accuracy of 96% is not high enough to actually rely on it. With an accuracy of 99% or over, the algorithm could help doctors to make a preselection by marking x-rays with obvious pneumonia and x-rays where it’s hard to detect. A doctor could then look at the preselected x-rays and be more efficient because he/she does not need to look at all of the x-rays. This would save time and people who are actually sick can get needed help sooner. For the given time and data, we think, we can still be happy with our outputs. We learned a lot about how to build and change features within the network. Lastly, we still want to mention a couple limitations of this project. On the one hand there is the time limitation due to the class schedule and on the other hand we only had over 5000 pictures to train our model with. The data set is imbalance and therefore, we had a lot more images of pneumonia than healthy lungs. In addition, the sizes of the images differ a lot between pneumonia and healthy lungs. This could be a problem because when we resize the images (224X224 for the MobileNet), there could potentially be differences in resolution across classes. For example, one class could have higher resolution images than the other, leading to bias in the model, or poorer performance than expected out of sample.