

Assignment 1 Report

Eashan Singh

Computer Science, BS

eashan.singh1@knights.ucf.edu

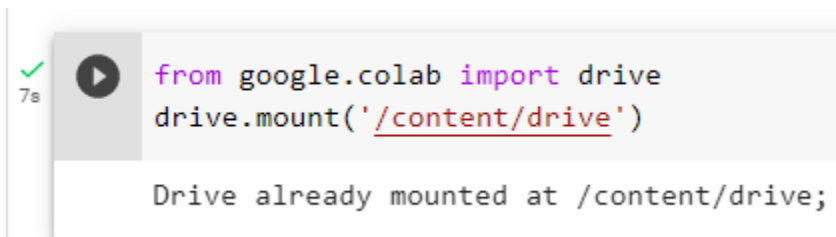
Link to associated code: <https://github.com/eashansingh905/chest-xray-pneumonia-classification>

Project Workflow

The associated code to this report can be found here: [link](#). The link to the GitHub repository has 2 .ipynb files “P1.ipynb” and “P2.ipynb” representing solutions to Task 1.1 and Task 1.2 respectively.

I wrote my code on Google Colab and utilized the free GPU provided by Google for training and evaluating the models.

The dataset used for this project is originally sourced on Kaggle but for faster access and reduced overhead (continuously downloading data with every runtime) I have downloaded the chest_xray dataset to my Google Drive and mounted my Google Drive onto the Python notebook for direct access to the dataset.



The screenshot shows a code cell in Google Colab. On the left, there is a green checkmark and a play button icon. The code inside the cell is: `from google.colab import drive` and `drive.mount('/content/drive')`. Below the code, the output text reads: "Drive already mounted at /content/drive;".

For this assignment, I am using the Pytorch deep learning framework.

Task 1.1: ResNet-18 Classifier

1. Objective

The objective of Task 1.1 is to employ an “off-the-shelf” CNN architecture for the classification of chest x-ray images for the presence of pneumonia. The dataset being used for this task is linked [here](#). The model is to be trained from scratch.

2. Model Architecture

For this assignment, the model of choice which I have decided to employ is the ResNet18. The ResNet architecture does a good job of addressing the problem of vanishing gradient for high number of layers with the help of the Residual Block, which enables residual learning.

Further, skip connections allow the network to train/converge faster than other deep learning architectures. We have seen a rise in the number of computer vision applications that employ residual blocks and thus I wanted to try this architecture for this medical imaging problem.

Due to the limited computational resources available to me, I have decided to use ResNet18 over a deeper network like ResNet50 for reduced FLOPS (floating point operations). The ResNet18 has around 11 million trainable parameters compared to the next biggest, ResNet34 which has about 24 million trainable parameters.

For this task, the model’s weights will not be pre-trained, and will start randomized, so they are trained from scratch. The final fully connected layer will have 2 output layers for this binary classification task, representing the presence or absence of pneumonia.

3. Training

3.1 Hyper-parameter Tuning

3.1.1 Batch Size

For training and test, I am using a batch size of size 32. For validation, there are only 16 images so I use all images in a batch size of 16. To optimize GPU performance, I wanted to pick a size that is a power of 2, but not too large as to slow down my training time. Size 64 and 128 batch sizes were also tried but were significantly slower than size 32. While convergence with smaller batch is quicker, the cost in training process is generally noisier.

- The training set has 5216 images thus we have $5216 / 32 = 163$ batches
- The test set has 624 images thus we have $624 / 32 = \text{approx. } 20$ batches

For training the batches are randomized from epoch to epoch.

3.1.2 Learning Rate

The optimal learning rate I found was 0.001. I know that generally larger batch sizes require larger learning rates for faster convergence and since my batch size is relatively small, I should be looking at a learning rate between $1e-5$ and $1e-3$. After experimenting with a different range of values I found that 0.001 gave good results relative to training time. 0.001 gave me the lowest loss vs training time sacrifice.

3.1.3 Optimization Algorithm (Optimizer)

I am using the Adam Optimizer for this problem. Due to the computational limitations, I figured that fast convergence would be a priority even though SGD may bring slightly more optimal results. Especially since the weights are not pre-trained, I thought this decision made the most sense to speed up training.

3.1.4 Loss Function

Cross-entropy loss has been widely recognized for classification tasks in computer vision and thus is my choice for loss calculations.

3.1.5 Epochs

One epoch of ResNet-18 with the above hyper-parameters and chest x-ray dataset takes about 2.15 minutes on Google Colab with GPU. This is quite long especially since we are doing 163 batch calculations during training. For this reason, to optimize for time and training, I have decided to use 5 epochs for evaluations so training time is roughly 10-12 minutes.

```
100%|██████████| 163/163 [01:56<00:00, 1.40batch/s]
Epoch: 1, Time: 2.1612709363301597, Loss: 0.17818601429462433
```

3.2 Data

3.2.1 Dataset

| Dataset | NORMAL | PNEUMONIA | Total |
|---------|--------|-----------|-------|
| train | 1,341 | 3,875 | 5,216 |
| val | 8 | 8 | 16 |
| test | 234 | 390 | 624 |
| Total | 1,583 | 4,273 | 5,856 |

This chart gives a breakdown of the dataset that this project will be dealing with. As you can see, there is a clear disparity in the class sizes of pneumonia and normal data. For training, the set of pneumonia images is 2.88 times larger than the normal images. For test it is 1.66 times larger.

To tackle this class imbalance, there are a few options that are worth considering. The first of which can be undersampling the pneumonia class images. This is a technique used to balance uneven datasets by decreasing the size of the majority class. Another option is to oversample the minority class by creating synthetic images, however this strategy can be prone to overfitting. Finally, another strategy can be using data augmentation. In this approach, you modify pre-existing images by creating synthetics with the slightest effects. Examples include geometric manipulation, color changes, kernel filters, etc.

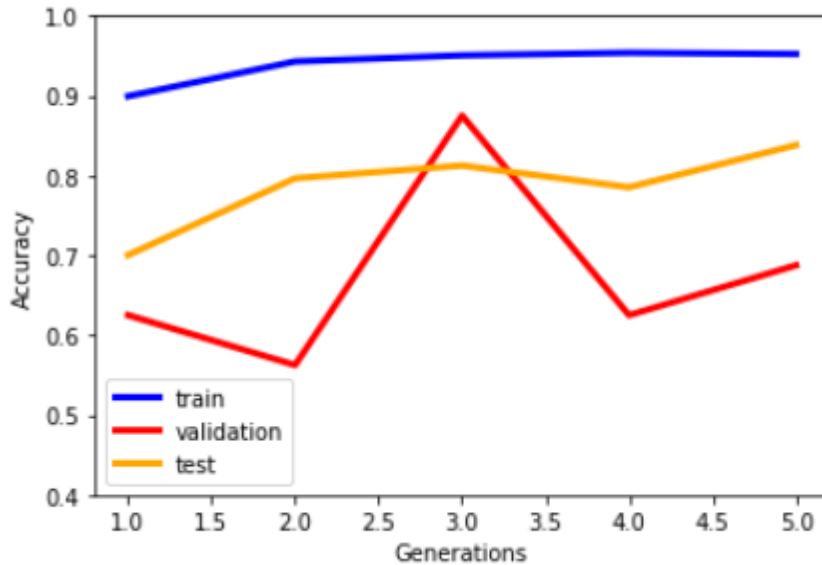
For the sake of the limited number of data points provided and to reduce our chances of overfitting, I have opted to avoid undersampling the pneumonia class or oversampling the normal class. The reason for this being: overfitting in either direction can lead to quite catastrophic results in the application of pneumonia classifications. We want to limit our false positives as much as we can, especially the case of no-detection when there is in fact pneumonia present. I will rely on data augmentation techniques instead to create meaningful variances that can help improve the classification ability.

3.2.2 Data Normalization

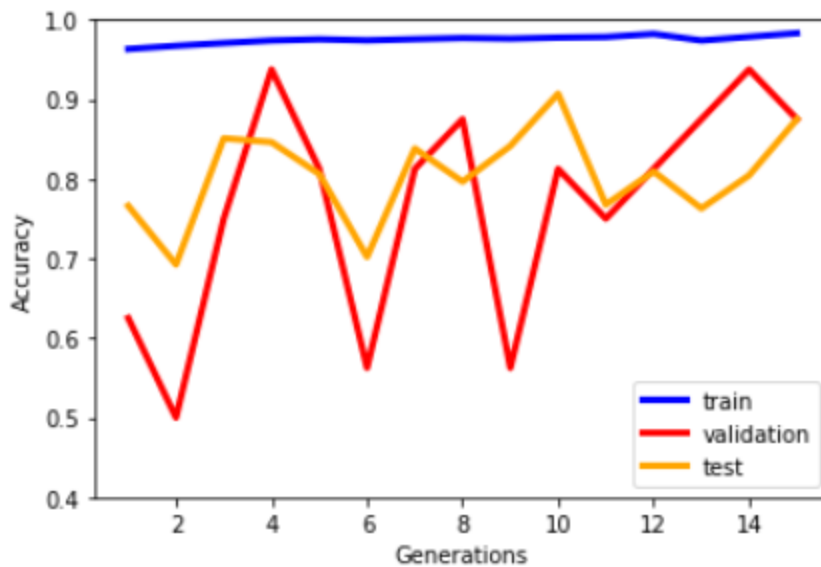
Normalization for image classification tasks is advisable because they allow for fair comparisons in images of varying levels of intensity, textures, and quality. Thus, I employ batch normalization using the mean and standard deviation of the dataset.

- Mean = [0.5303,0.5303,0.5303]
- Standard Deviation = [0.1944,0.1944, 0.1944]

3.3 Initial Raw ResNet18 Results (no Augmentations)



I ran the ResNet18 with no augmentations to see baseline performance purely based on hyper-parameter tuning. The above graph shows the results I achieved over 5 epochs. Because the validation test set is very small(16 images), every misclassification affects the validation(red line) by 6.25%, making it very noisy. As we can see, even with no pretrained weights, the training set is able to achieve 90% accuracy from the first epoch. Over the course of 5 runs the training accuracy is able to reach 96%. The test accuracy starts at 71% and is able to reach **83% accuracy** in just over 5 epochs!



To further study the baseline performance, I allowed my computer to run over night to see how it would perform over 15 epochs. From the observed chart, we can see that the training data has a very strong fit while the test/validation data fluctuates quite a lot(noisy).This may be a symptom of overfitting and will be something I look into for the modifications.

Here is the performance of the best epoch:

```
100%|██████████| 163/163 [01:47<00:00, 1.52batch/s]
Epoch: 9, Time: 1.9819676319758097, Loss: 0.022429557517170906
Train_acc: 0.977569018404908(5099/5216), Val_acc: 0.8125(13/16)
Test_acc: 0.907051282051282(566/624)
```

In epoch 9, our training accuracy was 97.7% and our **test accuracy was 90.7%.**

I will look to improve on this in the next section.

4. Modifications/ Improvements

4.1 Data Augmentation

To improve the baseline performance of the ResNet on the dataset, I performed some data augmentation to make my model more robust to variance in the data. Taking inspiration from [1] Rahman et. al (2020) who performed data augmentations on chest x-rays for improved test accuracy, I employed the following techniques:

4.1.1 Rotation

[1] Rahman et. al performs rotations of the chest ray images at 45 degrees clockwise. However, since our dataset is pretty small, and the dataset does not seem to exhibit images with 45 degree slants, I will limit the rotation range from (-25, +25) degrees. Essentially allowing the image to be tilted 25 degrees in either direction of the y-axis.

4.1.2 Color Jitter

Color jitter allows for variations in brightness, hue, and saturation. This is a useful augmentation to use because these are arguably the most important properties to control when comparing different types of x-ray images. Different levels of brightness for example may indicate to the model that certain regions of an x-ray are of special interest.

4.1.3 Horizontal Flip

Horizontal flip allows for flipping an x-ray image along the y-axis so that we see a mirror of the photo. We more or less expect symmetric properties along the y-axis since we are looking at chest images. Vertical flip on the other hand (along the x-axis) would not

make any sense because we do not expect x-ray images to be symmetrical along the opposite axis.

4.2 Regularization

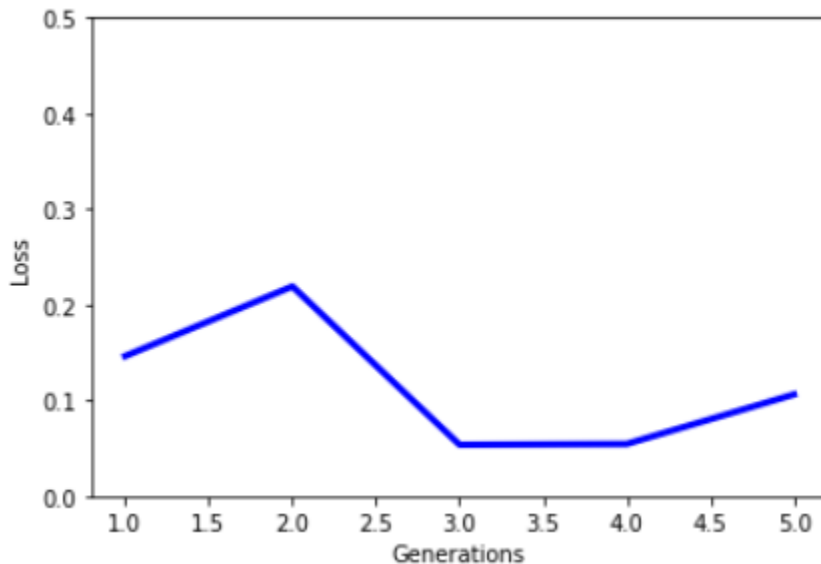
4.2.1 Weight Decay

As we observed in our initial test run of the ResNet18 model above, we noticed that the model fit very well to the training data however the validation and test sets were extremely noisy. To help us reduce some of the overfitting tendencies we see, I will introduce a small weight decay parameter of 0.0001 to the Adam Optimizer so we can prevent weights from taking on large values. This should help prevent overfitting and allow the validation/test sets to fit the data better.

5. Results

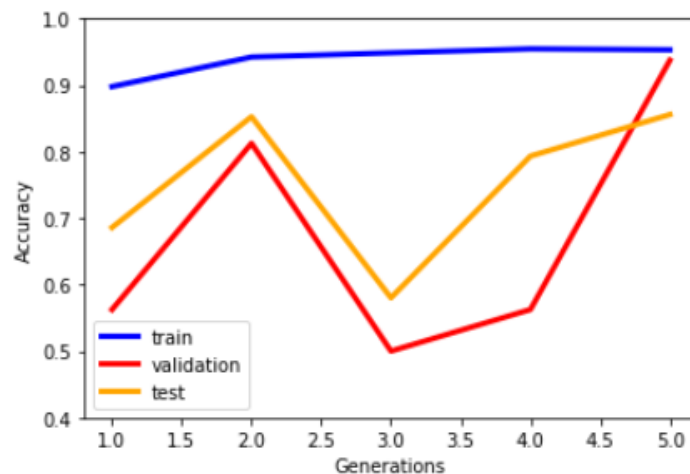
The results below show the performance of our model after performing various forms of data augmentation and model regularization strategies. Specifically, we now augment the dataset to have variant rotational images that span from -25 to +25 degrees from the y-axis, we introduce random color jitters to images to change properties like brightness, hue, and saturation, and we also allow for horizontal flips that flip a photo along the y-axis. We also introduce some model overfitting strategies such as weight decay to make sure that our model can accurately read the validation/test set.

5.1 Training Loss Curve



The training loss curve as shown above represents the training loss over 5 epochs. Our model starts off with fair training loss around 15 percent and improves over time. Unlike our previous model which fit too tightly to the data, this model is slower to conform to the training data allowing for a better representation of the test data. If we were to perform more than 5 epochs, we would see the training loss improve.

5.2 Training / Validation/ Test Accuracy



Now, looking at the accuracies of the training, validation, and test results we can make some important observations

1. The validation and test accuracies align more closely than before. In the baseline approach, we saw that the accuracy and test results over the 5 epochs had much noisier fluctuations. This tells us that our model regularization is working well to prevent overfitting of our training data too since the validation test set is usually only a significant indicator of the test set when the weight changes in training are not too drastic.
2. Our training accuracy continues to remain over 90%, but the rate at which it goes up is slower, another testament to good model weights. Our test results are able to achieve their best results in the 5th epoch where it achieves an **accuracy of 85.5%**! Given more epochs, the accuracy would continue to up. This trumps our baseline results of 83% accuracy prior.

Below, is a picture of the progression of training results over the course of 5 epochs.

```
100%|██████████| 163/163 [01:50<00:00, 1.48batch/s]
Epoch: 0, Time: 2.0363056023915607, Loss: 0.14627984166145325
Train_acc: 0.8978144171779141(4683/5216), Val_acc: 0.5625(9/16)
Test_acc: 0.6858974358974359(428/
624)
```

```
100%|██████████| 163/163 [01:49<00:00, 1.49batch/s]
Epoch: 1, Time: 2.017255504926046, Loss: 0.2191540002822876
Train_acc: 0.942292944785276(4915/5216), Val_acc: 0.8125(13/16)
Test_acc: 0.8525641025641025(532/
624)
```

```
100%|██████████| 163/163 [01:48<00:00, 1.50batch/s]
Epoch: 2, Time: 2.0153933207194012, Loss: 0.053560521453619
Train_acc: 0.9486196319018405(4948/5216), Val_acc: 0.5(8/16)
Test_acc: 0.5801282051282052(362/
624)
```

```
100%|██████████| 163/163 [01:48<00:00, 1.50batch/s]
Epoch: 3, Time: 2.014026474952698, Loss: 0.054385725408792496
Train_acc: 0.9545628834355828(4979/5216), Val_acc: 0.5625(9/16)
Test_acc: 0.7932692307692307(495/
624)
```

```
100%|██████████| 163/163 [01:48<00:00, 1.50batch/s]
Epoch: 4, Time: 2.0147287527720135, Loss: 0.1061338260769844
Train_acc: 0.9532208588957055(4972/5216), Val_acc: 0.9375(15/16)
Test_acc: 0.8557692307692307(534/
624)
```

5.4 Classification Accuracy Performance Metrics

The section below provides more performance metrics.

| Metric | Performance |
|-------------|-------------|
| Accuracy | 0.806 |
| Specificity | 0.487 |
| Precision | 0.764 |
| Recall | 0.997 |
| F1 Score | 0.865 |

Data generated from sample run of best model

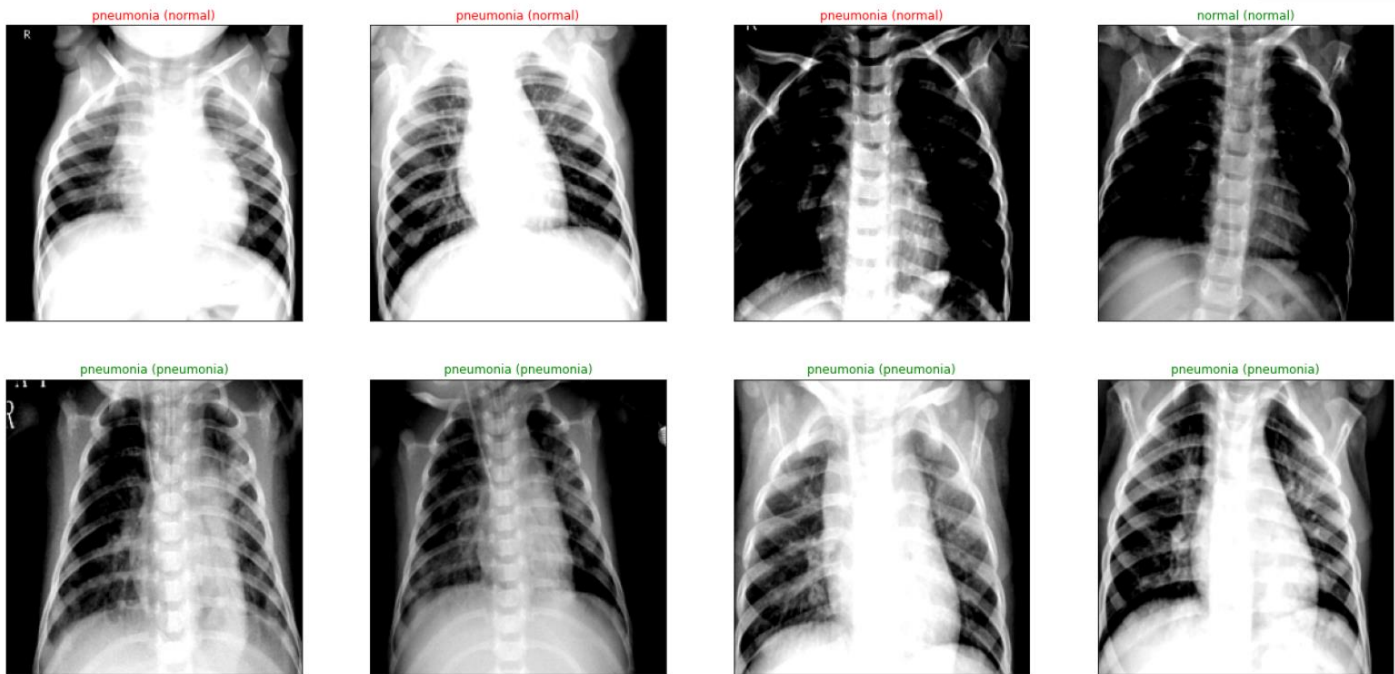
5.5 Confusion Matrix

| | Normal Pred. | Pneumonia Pred. |
|-----------|--------------|-----------------|
| Normal | 114 | 120 |
| Pneumonia | 1* | 389 |

In the context of this problem, we want to limit the number of False Negative values, or in this case, the number of pneumonia patients who are classified as normal. In our code, only 1 case of this was found.

6. Failure Cases

Below are some sample cases of a classification for descriptive study purposes



The green labels signify successful classifications. The red labels show failure cases. The label within the parenthesis is the actual label and the label outside is the prediction made by the classifier. The model does very well at classifying pneumonia images as pneumonia but suffers in sometimes mistaking a normal image as a pneumonia case.

7. Future Testing

If I were to redo this project, I would try more class imbalancing strategies to make sure that the classifier is less biased towards classes that are larger. Data augmentation was used as a technique in this project, but it is worth exploring other routes like upsampling or maybe even downsampling of the majority class.

Task 1.2: Transfer Learning

1. Objective

For this task, our goal is to leverage the pretrained ImageNet weights on the ResNet18. Specifically, the aim is to achieve the best test accuracy possible with these learned weights.

2. Model Architecture

2.1 ResNet18 + Pretrained Weights

Similar to Task 1.1 we will be leveraging the ResNet18 for image classification of pneumonia in chest x-ray images. This time however, we must work with the learned weights from the ImageNet dataset and try to get the best accuracy for classification.

2.2 Transfer Learning Motivations

The main motivation for using Transfer Learning is that training models from the ground up is a computationally heavy and redundant task. A lot of the learning of weights being done for different architectures is common, therefore enabling sharing of these weights allows for more people to have access them. Especially on models that are more computationally demanding, we may not have the resources to train from the ground up and achieve the same accuracy.

In the context of medical imaging, clinical practices would be incentivized to leverage pretrained models for downstream products/tasks that do not need to be trained from scratch all the time. It is very time intensive and in a

world that is moving to “edge technology”, we want to make technology as useable as possible to as many groups as we can.

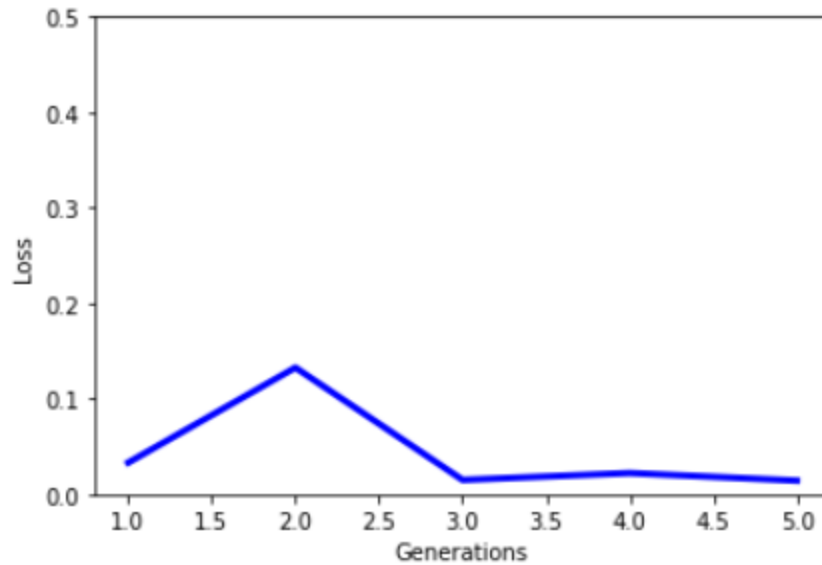
3. Training & Modification to Task 1.1

Using the same procedure mentioned in the Task 1.1, I was able to get similar results, surprisingly. The augmentation techniques were left unchanged. However, there were 2 changes that I had to make:

- Decrease of ***Learning Rate***. Something I observed from the training and validation accuracy this time around compared to Task 1.1 is that training accuracy converged much faster than the validation/test accuracy. In some cases, the test accuracy would get worse after 3-4 epochs. To account for this, I decided to decrease the learning rate by 10-fold to 0.0001. I believe the rapid changes in the validation/test accuracies was because we were making updates too quickly. Decreasing the learning rate helped to solve this problem.
- No **Weight Decay**. I introduced weight decay in Task 1.1 as a model regularization technique due to overfitting. After reducing the learning rate, I no longer found it necessary to incorporate the gradual weight decay as I did in Task 1.1.

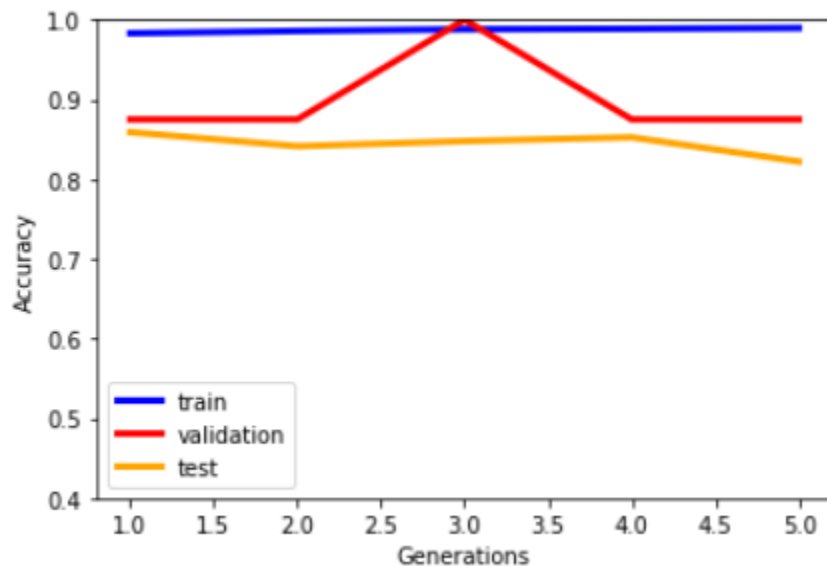
4. Results

4.1 Training Loss Curve



Training loss across 5 epochs of the best solution. After 3 epochs we can observe that training accuracy has preserved well around 0.01.

4.2 Training / Validation / Test Accuracy



For the transfer learning task, my model was able to achieve best results at a training accuracy of 0.98 and **test accuracy of 0.858**.

A progression of the training results over 5 epochs are shown below.

```
100%|██████████| 163/163 [01:49<00:00, 1.49batch/s]
Epoch: 0, Time: 2.0632766683896384, Loss: 0.03299155458807945
Train_acc: 0.9831288343558282(5128/5216), Val_acc: 0.875(14/16)
Test_acc: 0.8589743589743589(536/624)
```

```
100%|██████████| 163/163 [01:48<00:00, 1.50batch/s]
Epoch: 1, Time: 2.0527915080388386, Loss: 0.1325448900461197
Train_acc: 0.9860046012269938(5143/5216), Val_acc: 0.875(14/16)
Test_acc: 0.8413461538461539(525/624)
```

```
100%|██████████| 163/163 [01:49<00:00, 1.49batch/s]
Epoch: 2, Time: 2.059904666741689, Loss: 0.01485507283359766
Train_acc: 0.9881134969325154(5154/5216), Val_acc: 1.0(16/16)
Test_acc: 0.8477564102564102(529/624)
```

```
100%|██████████| 163/163 [01:49<00:00, 1.49batch/s]
Epoch: 3, Time: 2.064146049817403, Loss: 0.022058766335248947
Train_acc: 0.9886886503067485(5157/5216), Val_acc: 0.875(14/16)
Test_acc: 0.8525641025641025(532/624)
```

```
100%|██████████| 163/163 [01:49<00:00, 1.49batch/s]
Epoch: 4, Time: 2.0617646257082622, Loss: 0.014132199808955193
Train_acc: 0.9896472392638037(5162/5216), Val_acc: 0.875(14/16)
Test_acc: 0.8221153846153846(513/624)
```

4.3 Classification Accuracy Performance Metrics

The section below provides more performance metrics.

| Metric | Performance |
|-------------|-------------|
| Accuracy | 0.822 |
| Specificity | 0.529 |
| Precision | 0.779 |
| Recall | 0.997 |
| F1 Score | 0.875 |

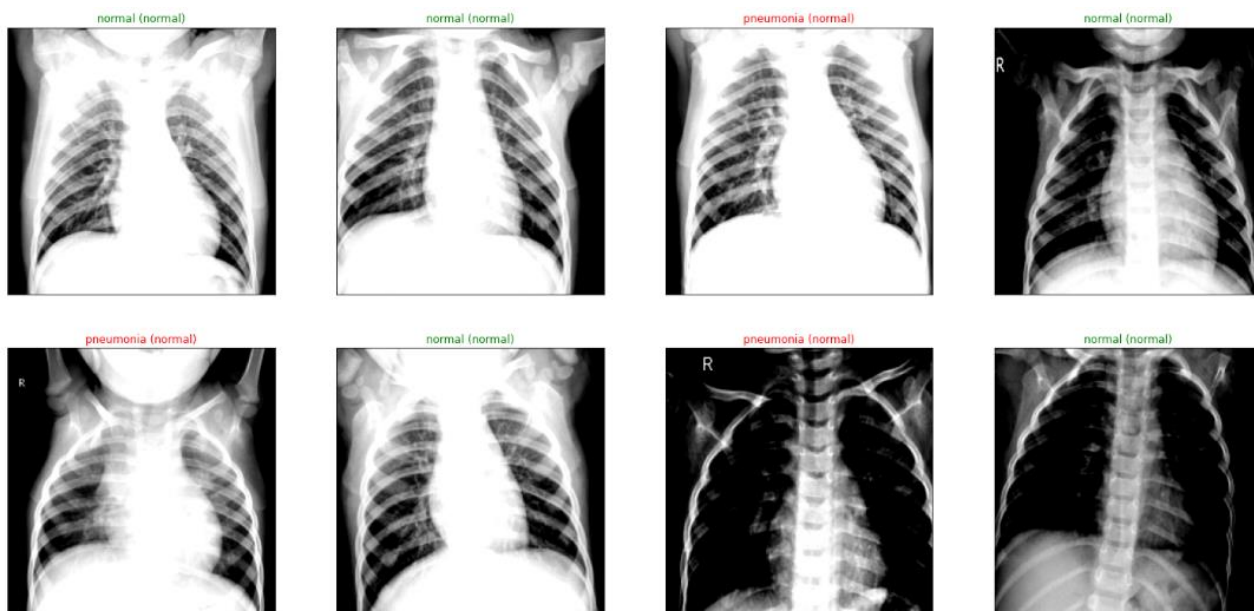
Data generated from sample run of best model

4.4 Confusion Matrix

| | Normal Pred. | Pneumonia Pred. |
|-----------|--------------|-----------------|
| Normal | 124 | 110 |
| Pneumonia | 1 | 389 |

One again, similar to task 1.1, the model does very well at avoiding False Negatives and does a predominantly good job recognizing Pneumonia. For some special instances of normal chest x-ray images, it mistakenly classifies them as Pneumonia.

5. Classification/Failure Cases



This photo shows 8 classifications made by the pretrained model on various types of images. Something interesting I noticed in this classification is the levels of brightness in areas towards the middle of the chest are closely associated with pneumonia classifications. The incorrect classifications in this image are a result of a similar problem I suspect where they mistake brightness with pneumonia. Further testing is required to see if any augmentations are creating more bias about class characteristics amongst specific image types.

6. Future Testing

For a future direction, I want to try more regularization approaches to see if I can prevent premature overfitting. The augmentations and regularization I used for both parts worked very well but training accuracy very quickly went about 0.95, while there was a lag in the validation and test results. Something I'd also like to try is freezing layers up until the fully connected layer to see how performance is affected. Finally, with more computational resources it would be interesting to see how I can improve results with a larger number of epochs. I restricted myself to 5 for this project but I may invest in Google Collab Pro for more computing resources.

References

[1] Rahman, T.; Chowdhury, M.E.H.; Khandakar, A.; Islam, K.R.; Islam, K.F.; Mahbub, Z.B.; Kadir, M.A.; Kashem, S. Transfer Learning with Deep Convolutional Neural Network (CNN) for Pneumonia Detection Using Chest X-ray. *Appl. Sci.* **2020**, *10*, 3233. <https://doi.org/10.3390/app10093233>