Final Project Report

IST 718 | Professor Jillian Lando

Image Classification | Radiology Images | ‘Normal’ vs ‘Pneumonia’

Meichan Huang, Matthew Pergolski, Shawn Anderson

Introduction

Pneumonia, a respiratory infection caused by bacteria or viruses, affects numerous individuals, particularly in areas characterized by high levels of pollution, unhygienic living conditions, and overcrowding, often accompanied by inadequate medical infrastructure. This infection leads to pleural effusion, a condition where fluids accumulate in the lungs, resulting in respiratory difficulties. Timely diagnosis plays a crucial role in ensuring effective treatment and improving survival rates.

Chest radiography is an essential diagnostic imaging test widely used in medical practice. The timely identification of potential findings and accurate diagnosis of diseases depicted in the images are crucial. However, the interpretation of these X-rays presents challenges and is subject to subjective variability. Thus, the detection of abnormalities through automated, rapid, and dependable methods plays a vital role in the radiology workflow.

In this study, we evaluate how effectively deep Convolutional Neural Networks (CNNs) can distinguish between normal and abnormal chest X-rays. Our comparisons strictly focus on the classification of images by algorithms. Two algorithms were tested, respectively ResNet and VGG-16. The purpose of this project is to assist radiologists and clinicians in promptly recognizing potential abnormal findings, enabling efficient work list triaging and reporting prioritization. Our business questions are:

How accurately can we classify pneumonia vs. normal chest X-rays?

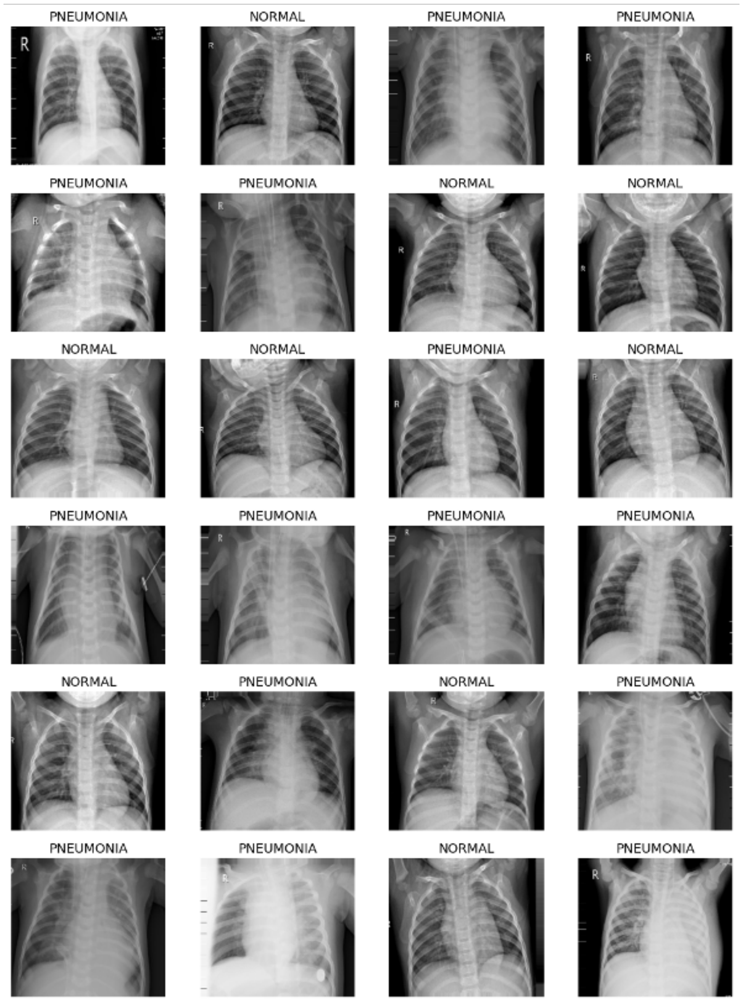
1. Which ML model best identifies/predicts the pneumonia vs. normal X-rays?
2. How accurate are the respective models?

Data Description

Dataset

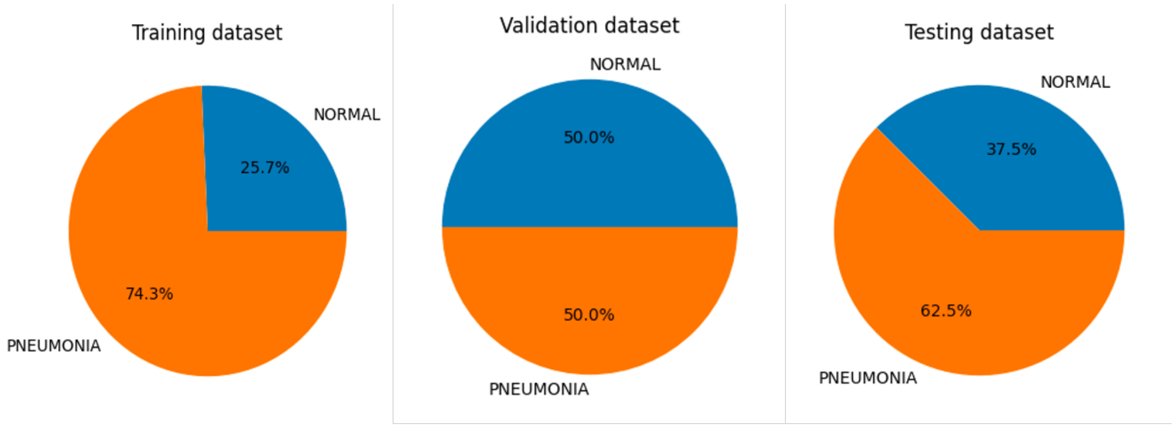
We leveraged 5,863 chest x-rays (see References) from children aged one to five years old with each image categorized by its respective diagnosis, normal or pneumonia. All patients were from the Guangzhou Women and Children’s Medical Center in Guangzhou, China. The dataset was obtained from Kaggle and was in the format of images. Therefore, data transformation is needed before we train the model.

The following is a randomly selected sample of the normal vs. pneumonia chest X-rays output from the dataset. In a normal chest X-ray, the lungs appear clear and transparent with well-defined lung markings. The lung fields show a symmetrical pattern, and there are no signs of abnormalities such as infiltrates or consolidation. On the other hand, a chest X-ray showing pneumonia may present distinct characteristics, depending on the type of pneumonia. For instance, in bacterial pneumonia, the affected lung regions often display patchy or lobar opacities, indicating areas of consolidation in typically lung lobes, where air-filled spaces are replaced with fluid or inflammatory debris. These opacities can be localized or spread throughout the lung. In viral pneumonia, the abnormalities seen on X-ray may be more diffuse and may present as interstitial infiltrates, where the inflammation affects the lung tissue between the air sacs (Tang et al. 2020). However, without trained eyes, one can assume the level of difficulties to correctly classify the cases, particularly with a large dataset of X-rays.



Dataset Descriptive Analysis

The dataset consists of three sub-datasets, namely training, validation, and testing. Overall, there were 5,216 images in the train set, 16 images in the validation set, and 624 images in the test set. The following pie charts showed the distribution of normal vs. pneumonia chest X-rays.



Considering the imbalanced nature of our data, we utilized class weight adjustment through the Sklearn library. This strategic intervention, specifically applied to the training dataset, aimed to enhance our model's performance and mitigate potential overfitting issues.

Data Analysis

We applied two models with various tuning. In the following section, we discuss the architectural features of the two models we applied, respectively ResNet and VGG – 16.

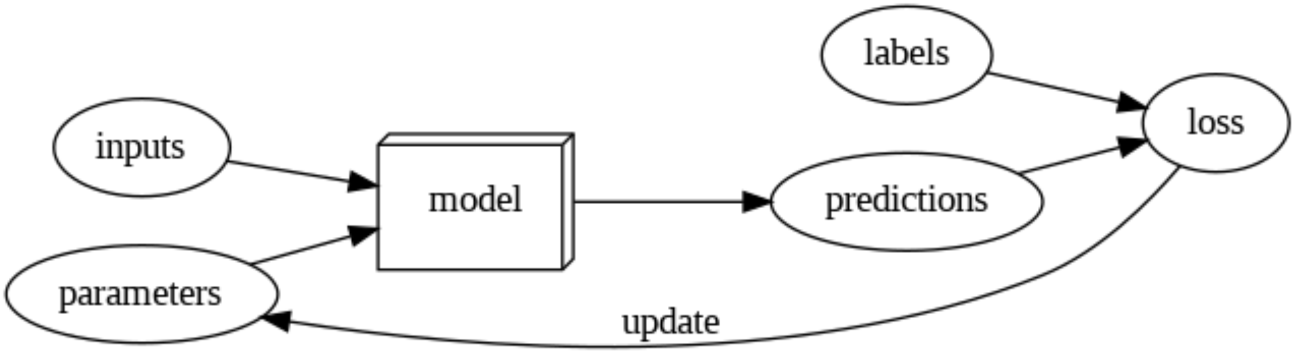
Model 1: ResNet

Within our analysis, two frameworks were used to assist the team in classifying radiology images into the categories of either ‘normal’ or ‘pneumonia:’ PyTorch and Fastai.

PyTorch is a powerful open-source library for Python, used widely for tasks that require complex computations, such as machine learning and computer vision. PyTorch offers a flexible library that leverages Graphics Processing Units (GPUs), making it particularly useful for deep learning tasks – such as radiology image classification. Ultimately, PyTorch's design has helped it become a popular choice among researchers and developers, overtaking Tensorflow recently.

PyTorch, with respect to the image classification project, uses backpropagation to achieve high accuracy results. Backpropagation is the ‘backbone’ of most neural network training. This algorithm calculates the gradient of the loss function with respect to the network's weights. This gradient is then used to adjust the weights to improve the network's predictions. Each training iteration, known as an epoch, involves an update to the model weights, contributing to the network's 'knowledge' of the data. It is ultimately an iterative process determined by the number of epochs decided when training of the model begins.

The below illustration conveys the overall process of training a model using backpropagation:



As previously mentioned, the PyTorch framework is considered a low-level programming language. In the case of PyTorch, it is termed as low-level because it offers fine-grained control over the specifics of a given neural network; however, this control often sacrifices simplicity and convenience – requiring detailed coding and a deep understanding of the underlying processes.

Fastai, on the other hand, is a high-level library built on top of PyTorch. It focuses on providing a more user-friendly code structure to the powerful features of PyTorch. It provides useful python functions and classes such as data loaders, learning rate schedulers, and more – making it easier to build machine learning pipelines. Ultimately, fastai's simplicity enables developers to build and train sophisticated models with fewer lines of code, with an option to view the underlying low-level PyTorch source code if/when needed.

Moreover, the team used both PyTorch and fastai to train on a subset of our image dataset – the training data – and tested our results on unseen data, the test set. In all, three deep learning Resnet models were trained on the dataset:

- Resnet18
- Resnet50
- Resnet152

For background, Resnet is a pre-trained PyTorch neural network architecture used in deep learning, specifically in image classification tasks. The below summary tables represent the training progress of each Resnet model. The 'epoch' column represents the number of complete iterations through the training dataset. The 'train_loss' and 'valid_loss' columns refer to the loss function's value calculated on the training and validation datasets, respectively. The 'error_rate' indicates the proportion of incorrect predictions. Finally, 'time' records the total duration of each epoch.

The team also took advantage of Google Colab's free limited-GPU access for training these deep learning models on the training dataset. Below is a snapshot of the relevant training information for the three models chosen (as described above):

Resent18

epoch	**train_loss**	**valid_loss**	**error_rate**	**time**
0	0.399708	0.207884	0.062299	12:26
epoch	**train_loss**	**valid_loss**	**error_rate**	**time**
0	0.173970	0.120216	0.033298	02:04
1	0.067924	0.048390	0.012889	02:04
2	0.035941	0.056787	0.015038	02:04

Resnet50

epoch	**train_loss**	**valid_loss**	**error_rate**	**time**
0	0.377561	0.193697	0.048335	02:18
epoch	**train_loss**	**valid_loss**	**error_rate**	**time**
0	0.190636	0.064476	0.023631	02:11
1	0.078045	0.039712	0.019334	02:10
2	0.024602	0.030338	0.015038	02:09

Resnet152

epoch	**train_loss**	**valid_loss**	**error_rate**	**time**
0	0.394098	0.165493	0.042965	02:38
epoch	**train_loss**	**valid_loss**	**error_rate**	**time**
0	0.151848	0.106010	0.034372	02:29
1	0.084596	0.049993	0.015038	02:25
2	0.026443	0.031645	0.009667	02:28

The number alongside the Resnet model name shines light on how many deep learning layers – or nodes – exist for the model. In short, the Resnet18 model consists of 18 deep learning nodes, while Resnet50 and Resnet152 contain 50 and 152 deep learning layers, respectively.

Model 2: VGG –16

The other model we explored was Visual Geometry Group-16 (VGG –16), which is a deep convolutional neural network architecture commonly used for image recognition and classification tasks. VGG-16 was proposed by K. Simonyan and A. Zisserman from the

University of Oxford (see Thakur), which was trained to classify images into 1,000 object categories for the ILSVRC (ImageNet Large Scale Visual Recognition Competition).

VGG-16 consists of 16 layers, including 13 2D convolutional (Cov2D layers, 5 max-pooling layers, and 3 fully connected layers, as shown in the below architectural structure graph.

Convolutional Layers

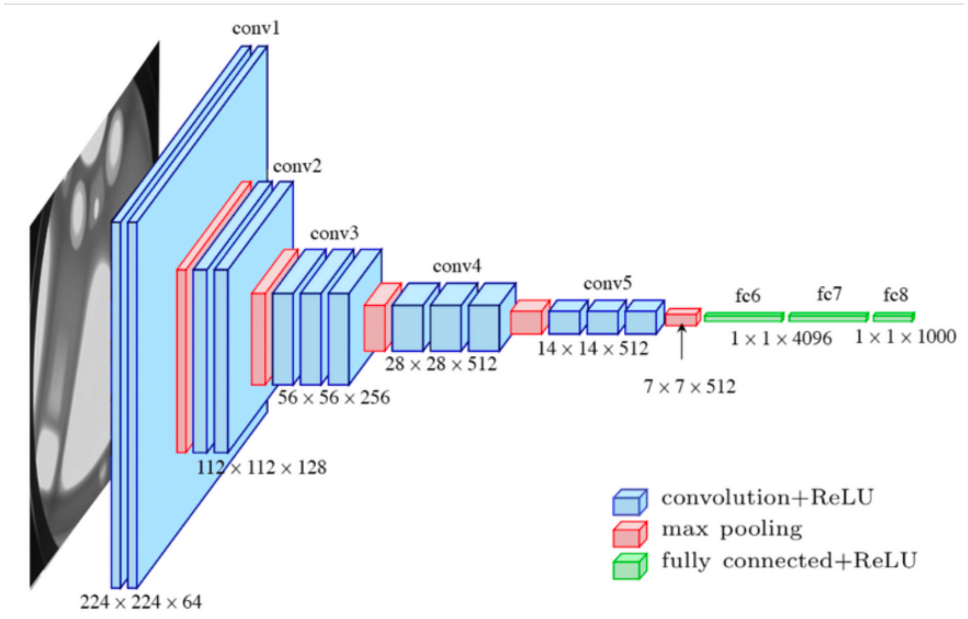


Image Preprocessing and Augmentation: In our modeling process, we deployed TensorFlow’s Keras API to implement and train the model. The raw input images were first preprocessed to be of the same size (224x224 pixels) as required by the VGG-16 model. The images were also normalized and augmented for better performance of the model.

Model Definition: The VGG-16 architecture is defined using TensorFlow's Keras API. The base model involves stacking multiple Conv2D and MaxPooling2D layers for feature extraction, followed by three Dense (fully connected) layers for classification.

Model Compilation: The model was compiled with a loss function of cross-entropy, an optimizer of Adam, and the accuracy matric to measure the model performance.

Model Training and Tuning: The model is trained on the prepared dataset using the fit() method. During the fine-tuning process, two methods were taken: (1) freezing and unfreezing the VGG-16 base model layers by setting trainable = False; (2) increasing the number of epochs the model goes through. Setting trainable = False freezes the layers in a base model, preventing weight updates during training. This approach, commonly used in transfer learning scenarios with pre-trained models, retains the model's learned features. New trainable layers are then added to the base model, and only these layer's weights are updated during training, leveraging the base model's knowledge for the new task.

The number of epochs is a hyperparameter that defines the number of times the learning algorithm works through the entire training dataset. Too few epochs could contribute to underfitting of the model. Therefore, we started with 3 epochs and increased the number of epochs to 5 in the final model of VGG-16.

Model Evaluation and Prediction: After training, the model's performance is evaluated on a test set. The model can also be used to make predictions on new, unseen images. To better determine the accuracy and decision-making process, we overlayed Gram-cam heatmap for VGG-16 predicted outcomes.

Model Results

The following section outlines the results of our two CNN-based modeling of ResNet and VGG-16.

Model 1: ResNet 18, 50, and 152
All in all, the following was achieved using the ResNet models with respective numbers of ResNet layers:

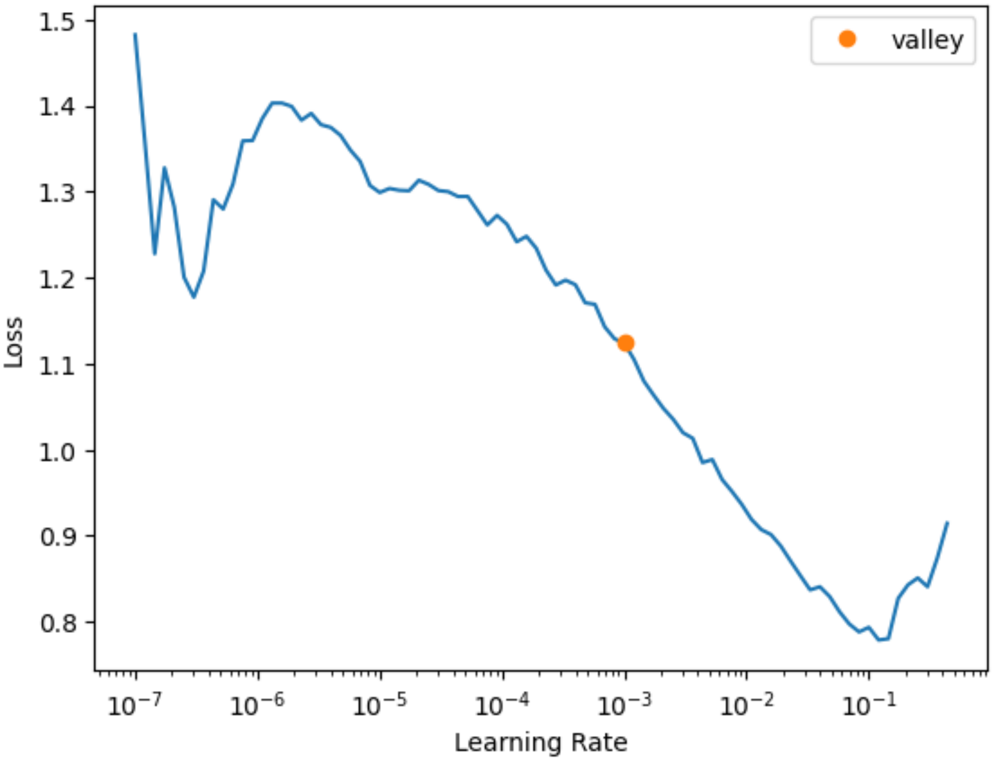
Model	**Accuracy**
RESNET18	84.78%
RESNET50	81.89%
RESNET152	85.58%

The three deep learning models – Resnet18, Resnet50, and Resnet152 – produced what the team noted as ‘respectable’ results for radiology image classification. The accuracy percentages for Resnet18, Resnet50, and Resnet152 were 84.78%, 81.89%, and 85.58%; respectively.

However, in comparing the training results to our test accuracies, a clear pattern of overfitting was observed across all three models. Overfitting can be described as when the model learns the training data ‘too’ well – often to the point where it captures not just the underlying patterns in the data, but also the noise and outliers within the training set as well. This was seen when our models' achieved accuracies close to 99% on the training data. On the other hand, when presented with unseen data in the test set, the models achieved a maximum accuracy of around 86%, highlighting the decrease in performance.

To counteract overfitting and aim for higher results, the team deployed a learning rate optimizer. An optimized learning rate can speed up the training time and improve overall model performance. Overall, the learning rate is a crucial hyperparameter that determines the step size taken during each iteration. In other words, an optimizer helps adjust this learning rate during training; making it dynamic and more well-suited for the data.

Upon leveraging Fastai's learning rate finder, which recommended a learning rate of 0.0010000000474974513. This led to a slight improvement in the accuracy of the Resnet18 model. With the optimized learning rate, the Resnet18 model was able to achieve an accuracy of 85.26%, marking an increase compared to its performance with a non-optimized learning rate.



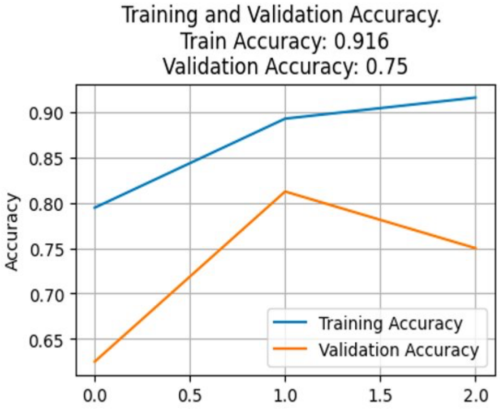
Model	**Accuracy**
RESNET18 (Learning Rate Optimized)	85.26%

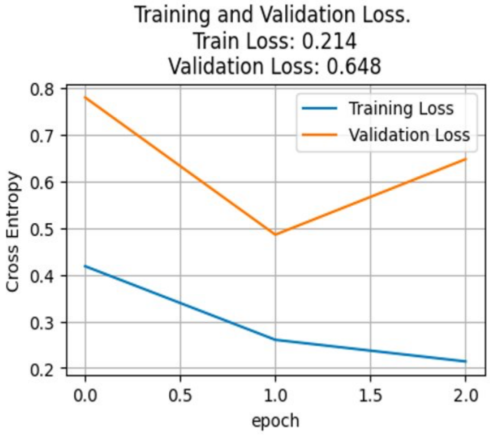
VGG –16 model results

Overall, when we load the VGG16 architecture with ImageNet weights as the base model, the VGG-16 model achieved a high training accuracy exceeding 90% with 3 epochs. These metrics offer insight into the model's performance throughout each training epoch.

```
163/163 [==============================] - 163s 930ms/step - loss: 0.4182 - accuracy:
0.7947 - val_loss: 0.7801 - val_accuracy: 0.6250
Epoch 2/3
163/163 [==============================] - 159s 975ms/step - loss: 0.2603 - accuracy:
0.8928 - val_loss: 0.4858 - val_accuracy: 0.8125
Epoch 3/3
163/163 [==============================] - 160s 982ms/step - loss: 0.2140 - accuracy:
0.9160 - val_loss: 0.6476 - val_accuracy: 0.7500
```

As we can see in the loss values, indicative of the model's predictive proficiency, it decreased over time. Lower values suggest the model's output prediction performance improved. The model's accuracy metrics, representing the fraction of correctly predicted instances, increased concurrently, further confirming this improvement. The validation loss and accuracy gave us an understanding of the model's performance on the validation dataset, which was meant to provide us with a sense of the generative capacity of the model. To our surprise, the validation accuracy was approximately 10-20% lower than that of the training dataset. However, it's important not to overlook the relevance of the results, considering that the validation dataset consisted of only 16 cases, which resulted in a reduction of generalizability and led to lower accuracy scores relative to the training dataset. This inadequate sample size in the validation set likely impeded robust statistical inference, adversely affecting the model's performance evaluation. Worthnoting, despite some fluctuations, the validation metrics, particularly in later epochs, were relatively high, pointing to the model's competency in making accurate predictions.

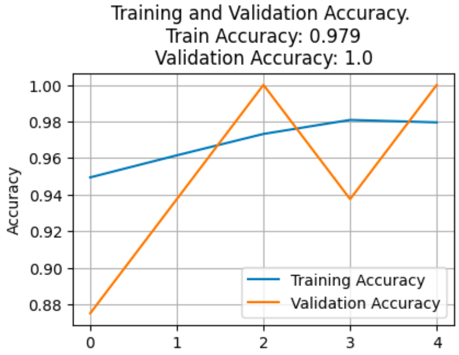


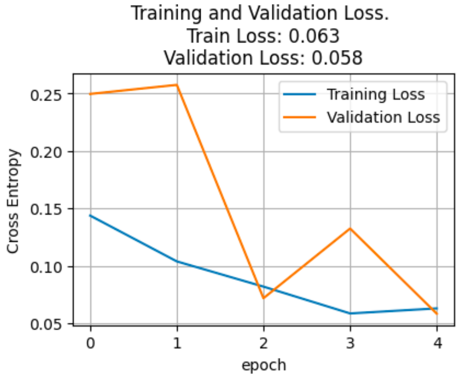


To improve the model performance, we unfreeze the base model layers and tune the epochs to 5. The base model, VGG16, is made up of 19 layers. Out of these, 4 layers have been unfrozen for training, i.e., the weights in these 4 layers were allowed to be updated during the training process. The model was trained for a total of 5 epochs, and for each epoch, the performance metrics - loss and accuracy - were recorded for both the training and validation sets. Overall, the train accuracy results improved significantly to 97.9% with a validation accuracy of 100%. The following is our final model performance in each epoch:

```
VGG16 base layers = 19 Unfreezing number of layers in base model = 4 Epoch 1/5 163/163
[==============================] - 161s 966ms/step - loss: 0.1437 - accuracy: 0.9494
- val_loss: 0.2497 - val_accuracy: 0.8750 Epoch 2/5 163/163
[==============================] - 160s 981ms/step - loss: 0.1038 - accuracy: 0.9615
- val_loss: 0.2576 - val_accuracy: 0.9375 Epoch 3/5 163/163
[==============================] - 160s 980ms/step - loss: 0.0818 - accuracy: 0.9732
- val_loss: 0.0717 - val_accuracy: 1.0000 Epoch 4/5 163/163
[==============================] - 160s 978ms/step - loss: 0.0585 - accuracy: 0.9808
- val_loss: 0.1324 - val_accuracy: 0.9375 Epoch 5/5 163/163
[==============================] - 159s 977ms/step - loss: 0.0628 - accuracy: 0.9795
- val_loss: 0.0582 - val_accuracy: 1.0000
```

The learning curve plot also showed that the model appears to be learning well from the training data, with the accuracy increasing and the loss decreasing over time. However, the validation accuracy reaching 100% in the last two epochs suggests that the model may be overfitting the training data, as it's perfectly classifying the validation data.

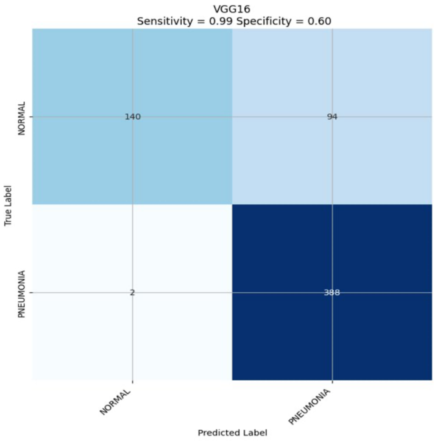




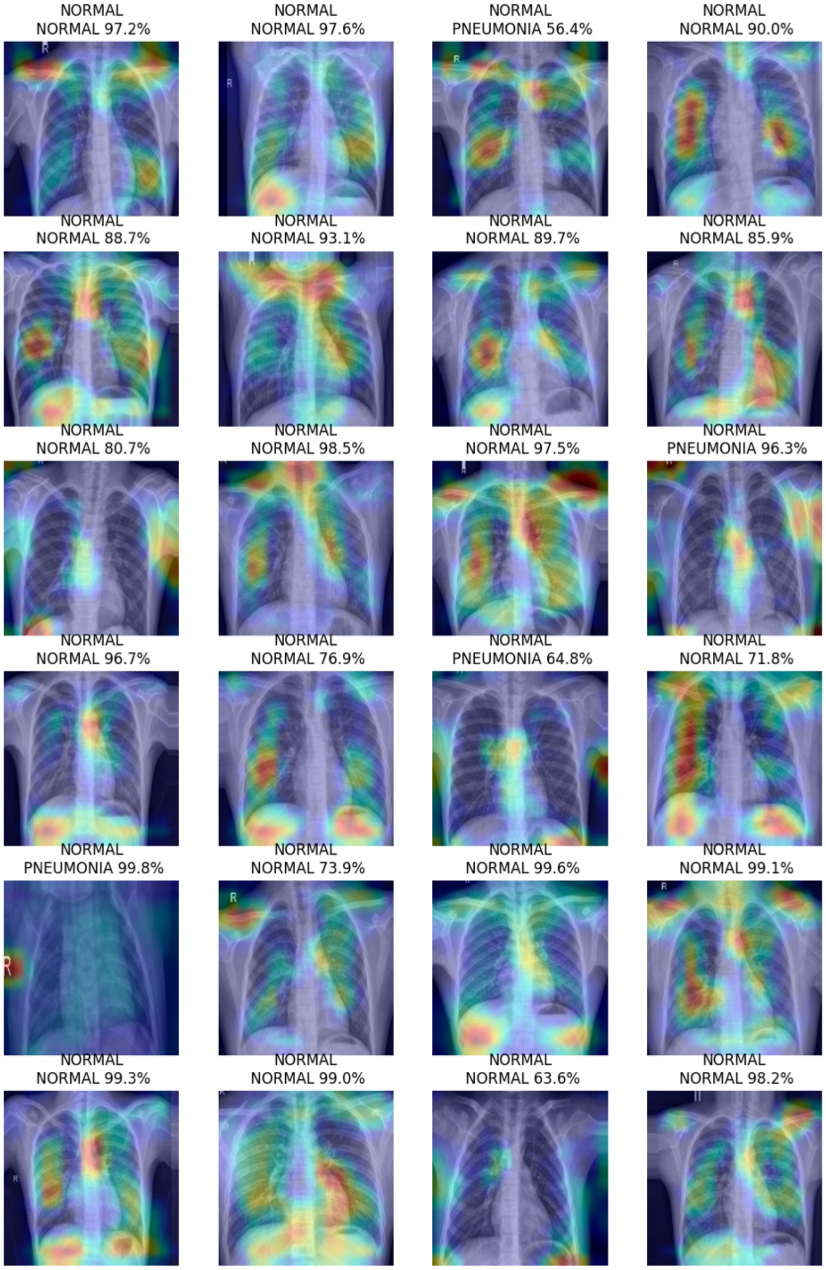
.

When evaluating the model against the test dataset, we confirmed that we encountered a similar issue of overfitting was seen in applying ResNet models to train the data. Upon testing the model on the test dataset, the accuracy decreased to 84%, indicative of mild overfitting - a limitation we previously highlighted.

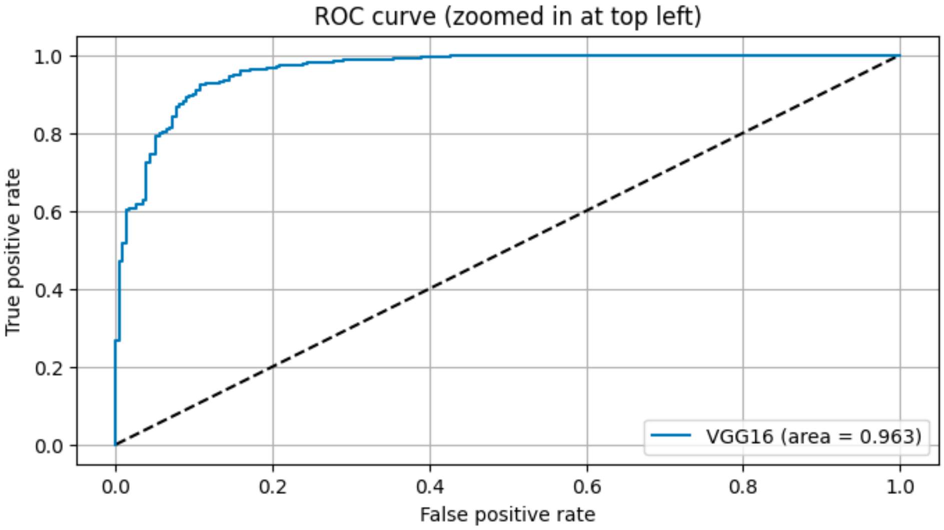
The confusion matrix of the final VGG-16 model demonstrates the model's proficiency in identifying pneumonia, with a scant two cases of false negatives. Nonetheless, the model exhibits issues with overfitting, as evidenced by the misclassification of 94 normal cases to pneumonia.



The team utilized Grad-CAM (Gradient-weighted Class Activation Mapping) to overlay the chest X-rays, highlighting how the model differentiates normal and pneumonia-affected X-rays. Grad-CAM is a technique for visualizing and interpreting a deep neural network's decision-making process, especially in image classification, which generates a heatmap underlining the significant regions or features in an image that the network primarily uses for prediction. As the results showed that the most heated regions overlayed with the lung area and upper respiratory area in most of the maps, however, not all predictions were accurate. For instance, the 3rd X-ray on the top was a normal X-ray but classified as pneumonia, and the algorithm was only 56.4% confident.



A receiver operating characteristic (ROC) curve was also plotted to examine the VGG-16 model's performance. An ideal ROC curve would closely approximate the top left corner, indicating that the classifier has a high true positive rate and a low false positive rate, regardless of the threshold. In our result, we can see that the ROC is nearly aligning with the 1.0 true positive rate on the top left, and the AUC area was close to 0.96, which indicated a near perfect accuracy using the VGG-16 model to classify the normal vs pneumonia in the children's chest X-rays data.



Conclusion

Machine Learning can assist current radiological examiners in screening for pneumonia, but our current accuracy results (~85%) do not warrant a replacement of their work. Use cases may include remote regions which lack sufficient medical staff, but which would benefit from screenings to prioritize the work of available professionals.

We recommend continued research into refining existing models and exploring new ones which would improve on these concepts with the aim in providing augmented medical services to regions which would not otherwise have sufficiently-trained people.

References

Howard, J., Gugger, S. (2021). fastai: A Layered API for Deep Learning. fast.ai. https://www.fast.ai/posts/2020-02-13-fastai-A-Layered-API-for-Deep-Learning.html

Le, K. (2021). An overview of VGG16 and NiN models. Retrieved from https://medium.com/mlearning-ai/an-overview-of-vgg16-and-nin-models-96e4bf398484.

Mooney, P. (2018). Chest X-ray Pneumonia. Retrieved from https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia.

Tang, YX., Tang, YB., Peng, Y. *et al.* Automated abnormality classification of chest radiographs using deep convolutional neural networks. *npj Digit. Med.* **3**, 70 (2020). https://doi.org/10.1038/s41746-020-0273-z

Thakur, R. (2019). Step by step VGG16 implementation in Keras for beginners. Retrieved from https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c.