Machine Learning Engineering Nanodegree

Machine Learning Capstone Project

Pablo R. Lopez

September 15th, 2021

Medical Diagnosis: Pneumonia Classification using Computer Vision and Transfer Learning

Fig 1: An example of a patient infected with bacterial pneumonia. source

1.0 Definition

1.1 Project Overview

Pneumonia is an inflammatory condition of the lung primarily affecting the small air sacs known as alveoli. Symptoms typically include some combination of productive or dry cough, chest pain, fever and difficulty breathing. The severity of the condition is variable. Pneumonia is a form of acute respiratory infection that affects the lungs. It is caused by viruses, bacteria and fungi and it can be treated with antibiotics.

Therefore, in order to alleviate this problem doctors are relying more on clinical decision support algorithms as a diagnostic tool. Image classification is an area of Deep Learning where the field of Health Care applies Deep Learning architecture models in disease detection into practical applications. Convolutional Neural Networks (CNNs) alongside Computer Vision could aid in diagnosing Pneumonia using chest X-ray images.

1.2 Problem Statement

The project objective is to use X-ray images to create an algorithm to identify patients with pneumonia vs normal. Therefore, this is a Computer Vision and classification task.

The dataset is about 1.2 Gb in size and this raises the question if it is big enough for our algorithm to perform well. Also, the dataset seems to be unbalanced and for the most part it is always best to have an equal number of images for each predicted class when doing image classification. In addition, the images seem to be in a grayscale format. This raises the question if it is necessary to convert grayscale images to RGB for better image object detection.

Steps to consider are:

1. Download, unzip , explore and preprocess the X-ray images.
2. Create a base model using the VGG-16 pretrained architecture.
3. Train and fine-tuned a Convolutional Neural Network to classify patients.

The expectation is to facilitate medical doctors or technicians to upload patient images into a web-app and thus to be able to decide if the patient has pneumonia or if the patient image seems normal.

The immediate solution is to build Deep Learning models such as CNN to find patterns and to predict which images have pneumonia or not. After finalizing the model a front-end serving web-app will be created to load images and determine a clinical diagnosis. This will facilitate tests to correctly identify all patients with the disease, and similarly correctly identify all patients who are disease free. However, this will serve only as a starting point, a professional needs to give the final verdict.

1.3 Metrics

When our model is trained an evaluation metric quantifies the performance of the classification model. There are standard metrics that are used in for evaluating predictive models such as:

1.3.1 Overall accuracy

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}$$

This will work well on most models, yet in the medical field, it is important to evaluate the problem and what is considered important. As a result, metrics that are useful in a clinical test do not necessarily rely only on accuracy. Can we do better? When it comes to clinical tests Sensitivity, Specificity, F-1 Score, and the AUC ROC Score can be used.

Definitions:

- ***Patient or Pneumonia*****:** positive for disease
- ***Healthy or Normal*****:** negative for disease
- ***True positive (TP)*** = the number of cases correctly identified as patient
- ***False positive (FP)*** = the number of cases incorrectly identified as patient
- ***True negative (TN)*** = the number of cases correctly identified as healthy
- ***False negative (FN)*** = the number of cases incorrectly identified as healthy

1.3.2 Accuracy

The accuracy of a test is its ability to differentiate the patient and healthy cases correctly. To estimate the accuracy of a test, we should calculate the proportion of true positive and true negative in all evaluated cases. Mathematically, this can be stated as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

1.3.3 Sensitivity - True Positive Rate - Recall

The sensitivity of a test is its ability to determine the patient cases correctly. To estimate it, we should calculate the proportion of true positives in patient cases. It refers to the true positive rate and it depicts how well the positive class was predicted. Mathematically, this can be stated as:

$$Sensitivity = \frac{TP}{TP + FN}$$

1.3.4 Specificity - True Negative Rate - Precision

The specificity of a test is its ability to determine the healthy cases correctly or the true negative rate. It helps us depict how well the negative class was predicted. To estimate it, we should calculate the proportion of true negatives in healthy cases. Mathematically, this can be stated as:

$$Specificity = \frac{TN}{FP + FN}$$

1.3.5 F1-Score

Precision and recall can be combined into a single score that seeks to balance both precision and recall. F-score might be a better measure to use if we need to seek balance between precision and recall and if we have an uneven class distribution. In addition, the F-Score is a popular metric for imbalanced classification. Their mathematically representation can be seen below:

$$Precision = \frac{TP}{TP + FP}, \qquad Recall = \frac{TP}{TP + FN}, \text{ \& } \quad Fscore = 2x \frac{Precision\ x\ Recall}{Precision + Recall}$$

1.3.6 AUC ROC Curve

A ROC curve is a diagnostic plot for summarizing the behavior of a model by calculating the false positive rate and true positive rate for a set of predictions by the model at all classification thresholds. AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0 and a model whose predictions are 100% correct has an AUC of 1.0.

The true positive rate (TPR) is the recall or sensitivity.

$$TPR = \frac{TP}{TP + FN}$$

The false positive rate (FPR) is calculated as:

$$FPR = \frac{FP}{FP + TN}$$

We can also note that prevalence matters because a machine learning algorithm with very high sensitivity and specificity may not be useful in practice when prevalence is close to either 0 or 1.

2.0 Analysis

2.1 Data Exploration and Visualization

The original dataset was provided by Daniel Kermany, Kang Zhang, and Michael Goldbaum. This is a part of a larger dataset named **Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images**. They collected and labeled a total of 5,232 chest X-ray images from children, including 3,883 characterized as depicting pneumonia (2,538 bacterial and 1,345 viral) and 1,349 as normal, from a total of 5,856 patients. This dataset is organized into 2 folders (train, test) and contains subfolders for each image category such as (Pneumonia/Normal).

The exploration was conducted on the training set only. The directory contains two

sub-directories:

- **NORMAL**: These are the samples that describe the normal cases.
- **PNEUMONIA:** These samples contain the pneumonia cases.

Below is a raw comparison of randomly samples of images in the training dataset folder.

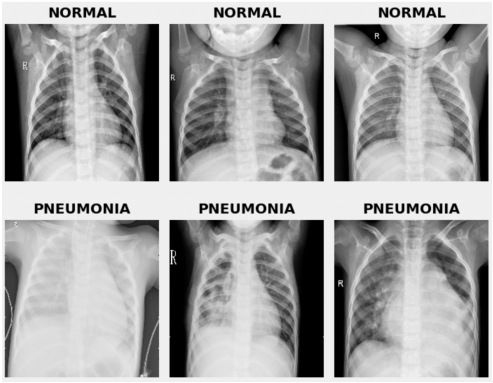


Fig 2: Sample images that contain both classes.

Moreover, figure 3 shows that our data is imbalanced. There seems to be a larger number of Pneumonia cases. This can indicate that all Pneumonia cases can be detected but there could also be false positives among them. Thus, a ROC could be a better metric to implement.

Fig 3: Number of classes in training images.

Also, figure 4 shows the components that describe each class the best. By using the eigenfaces we can visualize the principal component that describes 70% of variability for both classes. The eigenimages for the Normal images show much more edge definition around rib cages compared to the Pneumonia class. The image classifier will be able to

capture this difference.

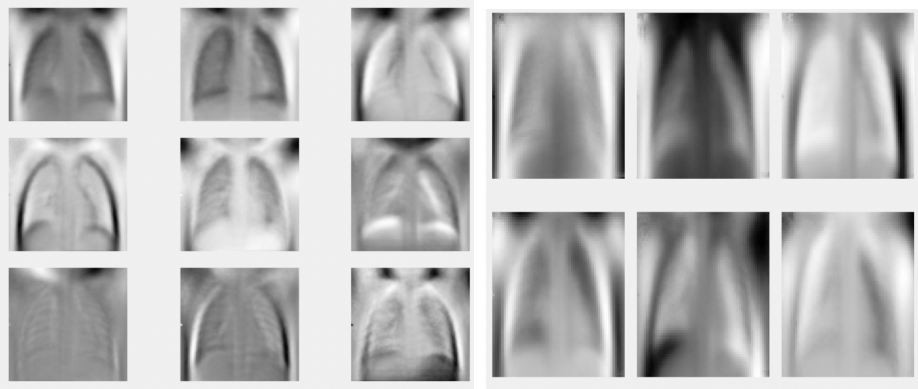


Fig 4: Eigenimages of both Normal (left)) and Pneumonia (right).

2.3 Algorithms and Techniques

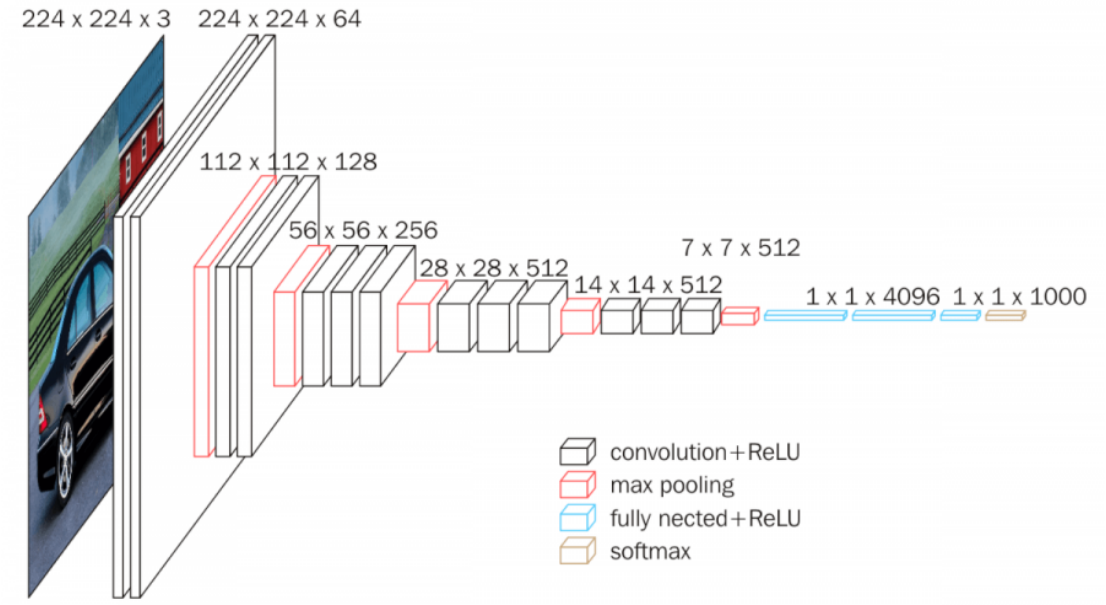


Fig 5: VGG-16 Architecture. source

The algorithm chosen is a VGG-16 Convolutional Neural Network for classification and detection. A CNN was chosen due to the fact that we are dealing with images and computer vision is ideal for this. The VGG stands for Visual Geometry Group and it is composed of 13 convolutional and 3 fully-connected layers as well as the ReLU activation function. This network stacks more layers onto AlexNet and uses smaller size filters (2x2 and 3x3). It consists of 138M parameters and takes up about 500MB of storage space. For

this reason, l took advantage of using transfer learning by leveraging the pretrained VGG-16 model.

Furthermore, an EDA will be conducted to get a quick look of the data structure (2D vs 3D images), random spot checks at a pixel level by looking at intensity profiles of individual images. Also, a check for data imbalance will be analyzed. In the image preprocessing phase will be removing potential noise from images (e.g. background extraction), performing normalization, conducting image augmentation and resizing images for the CNN architecture.

After preprocessing the images, the convolutional base will be frozen and the CNN algorithm will be executed. An analysis of its performance will be done. Then, after running the base model the first time a fine-tuning will be conducted. This will include data augmentation techniques to prevent overfitting along with some parameter tuning. For the learning rate optimization Adam will be applied. Also, fine-tuning will be performed by extracting features and adding layers to our pre-trained weights of the first model.

Finally, after we have our top performance algorithm a front-end using streamlit (a python front-end library) will be created to start making diagnostics on the images.

2.4 Benchmark

The goal is to obtain a AUC ROC score higher than 90%. As well as, a sensitivity and specificity score higher than 80%. The plan is to compare the base model (without data augmentation) and the fine-tuned model applying data augmentation as well as using parameter tuning.

3.0 Methodology

The project methodology will be using some aspect of the Machine Learning Life Cycle.

1. Load data
2. Split data into train/test (if necessary)
3. Exploratory Data Analysis (EDA)
4. Image pre-processing and data augmentation

5. Modeling (CNN) and Fine-tuning
6. Model Evaluation
7. Testing Deployment (Serving with a front-end)

3.1 Data preprocessing

The VGG-16 CNN pretrained model only accepts images as RGB color format. As a result, the original dataset comes in a grayscale color format, thus, it is needed to fit the images as an RGB format first. As well as, our images all have to be the same dimensions or pixel size. Also, the images have to be normalized within a scale of (0, 1). As a summary , the data processing techniques used for the base model and the fine-tuned model were:

1. Feed data to the model in a RGB color format.
2. Convert images to the same heights and widths, i.e. (224 x 224) before being fed to the model.
3. Standardized or resize the images to a unified dimension of a scale of [0,1].
4. Generate data batches for both train and test images.

Furthermore, for the fine-tuned model, we needed to expand our dataset artificially by using data augmentation. The idea is to alter the training data with small transformations to reproduce some variations. Data augmentation technique will occur on the training data in ways that change the array representation while keeping the label the same.

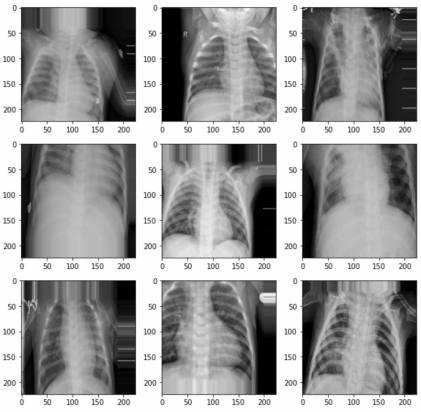


Fig 6: Data augmentation samples.

By applying just a couple of these transformations to our training data, we can easily double or triple the number of training examples and create a very robust model. We will

use “on-the-fly” data augmentation because this augmentation is done at training time.

For the fine-tuned model, data augmentation is included but at minimum because Medical X-ray scans are only taken in a specific orientation, and variations such as flips and rotations will not exist in real X-ray images. As a result, the following data augmentation techniques were created:

1. Randomly **zoom** train images by 20%
2. Randomly **width shift range** train images by 15%
3. As well as, randomly **height shift range** images by 15%
4. And finally, randomly **shear range images** by 15%

3.2 Implementation

Transfer Learning

Transfer learning was leveraged to reuse the pre-trained model on a new problem. The VGG-16 CNN pre-trained model was implemented to classify X-ray as a pneumonia case or as a normal case. The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world or our new model. We can then take advantage of these learned feature maps or model architecture without having to start from scratch by training a large model on a large dataset. Figure 7 depicts an example of using transfer learning and then adding a head model on top of it.

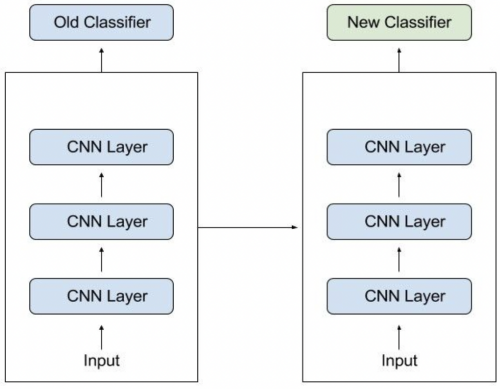


Fig 7: Example of an old and new classifier. source

Model Building

First, we'll pick a layer of VGG-16 to use for feature extraction. The very last classification layer (on "top", as most diagrams of machine learning models go from bottom to top) is not very useful. Instead, we will follow the common practice to depend on the very last layer. This layer is called the "bottleneck layer". The bottleneck layer features retain more generality as compared to the top layer.

Second, we'll instantiate a VGG-16 model pre-loaded with weights trained on ImageNet. By specifying the "`include_top=False`" argument, we'll load a network that doesn't include the classification layers at the top, which is ideal for feature extraction.

We can see the summary of our pre-trained VGG-16 model below:

```
[INFO] Summary of the pretrained base model architecture.

Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Fig 8: VGG-16 Model architecture.

From figure 8 we can see that we have 5 blocks available to train on. For the moment, we have 0 non-trainable parameters but will add some when we do some feature extraction.

Feature Extraction

The convolutional base (VGG-16 pre-trained model) from the previous step will be frozen and used as a feature extractor. Then, we will add a classification head and compile the model. When using Transfer learning via feature extraction it is important to freeze the convolutional base before we compile and train the model.

As a result, the model will be frozen by setting `”layer.trainable = False”` which prevents the weights in a given layer from being updated during training.

Additionally, we'll add a classifier on top of it and train the top-level classifier. The steps taken for feature extraction and to add a new classifier on top are:

1. We utilize an existing pre-trained classifier as a starting point for a new classification.
2. First, we will treat networks as arbitrary feature extractors.
3. Then, we'll use the representations learned by a previous network to extract meaningful features from new samples.
4. We'll add a new classifier, which will be trained from scratch, on top of the pre-trained model so that we can repurpose the feature maps learned previously for the dataset.

Therefore, we do not need to retrain the entire model. The base convolutional network already contains features that are generically useful for classifying images. However, the final classification part of the pretrained model is specific to the original classification task, and subsequently specific to the set of classes on which the model was trained.

When performing feature extraction, we treat the pre-trained network as an arbitrary feature extractor, allowing the input image to propagate forward, stopping at a pre-specified layer, and taking the outputs of that layer as our features.

Thus, For the base model, a top model was constructed and compiled. We can check the

new architecture on top of the VGG-16 model below on figure 9.
From figure 9 we can see that we now have 513 trainable parameters. This is a result of adding 2 new trainable variables on top of the VGG-16 model. A Global average pooling 2D and a Dense layer for predictions.

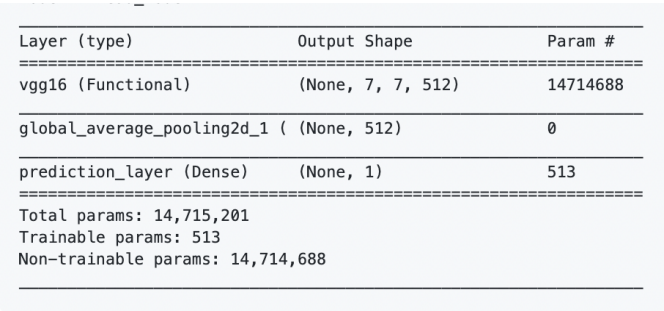


Fig 9: Top or head model over VGG-16 pre-trained model.

Model Training

The model was trained after it was chained together with the resized images, the VGG-16 pre-trained model and a feature extractor while freezing the VGG-16 CNN base. In addition, a classification head was added on top of the feature extractor and trained the top-level classifier. The top level classifier included a GlobalAveragePooling2D and a Dense layer with a sigmoid activation which was created using the Tensorflow and Keras Sequential API.

The feature extractor was used to convert an input shape of (224x224x3) into an input shape of (7x7x512) block of features. Thus, in order to make predictions from the block of features from the VGG-16 pre-trained model, it was necessary to average over the spatial (7x7) spatial locations, using the "**tf.keras.layers.GlobalAveragePooling2D**" layer to convert the features to a single 512-element vector per image such as in the shape of (32, 512).

Moreover, I applied a "**tf.keras.layers.Dense**" layer to convert these features into a single prediction per image. Here, an activation function was not needed because this prediction will be treated as a logit, or a raw prediction value. For instance, positive numbers will predict class 1, while negative numbers predict class 0. Thus, the single predictor per image will have the input shape format of (32, 1) as depicted in figure 9.

The number of samples per batch computation used was the default of 32. Since training a

transfer learning model takes a lot of computational power, the number of epochs chosen to train the entire images was 10.

The model optimizer chosen was the **Adam** algorithm which is a combination of **RMSprop** and **Stochastic Gradient Descent** method. Adam algorithm consists of adaptive estimation of first-order and second-order moments. This algorithm was chosen because it leverages the power of adaptive learning rates methods to find individual learning rates for each parameter. As well as, the Adam algorithm can adapt the learning rate for each weight of the Convolutional Neural Network.

Since our model has two classes, a **cross-entropy** loss for binary (0 or 1) classification applications was added because it will return a float tensor that will be minimized by the model. Also, we assume that **y_pred** (predictor label) encodes a probability distribution and the model provides a linear output. As well as, the **label smoothing** will produce a float between [0, 1]. If the float is greater than 0, then the label smoothing will squeeze them towards 0.5. That is, by using (1.0 - 0.5 * **label_smoothing**) for the target class and (0.5 * **label_smoothing)** for the non-target class.

The initial **learning rate** used was the default value of 0.001. The **metric** for our base model chosen was **accuracy** to be evaluated by the model during training and testing phases.

3.3 Refinement

Fine-tuning

Our base model leveraged the pre-trained model VGG-16 as a featured extractor for transfer learning. In other words, I did not train the entire model but I froze the convolutional base and added a head classifier to make predictions upon.

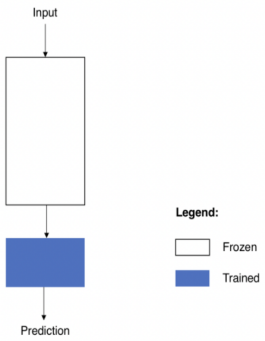


Fig 10: Freezing the CNN base.

Thus, the convolutional base was kept in its original form and then used its outputs to feed the classifier. You can see a depiction of how freezing the CNN base model can be done on figure 10. However, our results were not promising and thus the model needed fine tuning.

The steps taking to perform fine tuning:

1. Apply Data Augmentation
2. Estimate class weights for unbalanced datasets
3. Calculate training steps and validation steps per epoch
4. Define Callbacks and Tuning Parameters
5. Adding a new Head Classifier

3.3.1 Data Augmentation

The first step into tuning our model was to perform in-place data augmentation to the training set. This helped us create new training samples from the original ones by applying random modification. By seeing new data, it helps increase the performance of the model by generalizing better and thereby reducing overfitting. As mentioned in the data preprocessing section, data augmentation is included but at minimum because Medical X-ray scans are only taken in a specific orientation, and variations such as flips and rotations will not exist in real X-ray images. Therefore, only zooming by 20%, width and height shift range by 15% and shear range by 15% was also added.

3.3.2 Estimate class weights for unbalanced datasets

Considering that our data is highly unbalanced, a quick solution was to use the **`compute_class_weight`** from the sklearn library. The weight for class 0 (Normal) is a lot higher than the weight for class 1 (Pneumonia). The result obtained was:

{0: 1.939214232765011, 1: 0.6737058975019315}

Since there are less normal images, each normal image will be weighted more than Pneumonia to balance the data as the CNN works best when the training data is balanced.

An important fact is that while using **`compute_class_weights`** changes the range of the loss. This may affect the stability of the training depending on the optimizer. However, the

optimizer used is **`tf.keras.optimizers.Adam`**, and therefore is unaffected by the scaling changes.

3.3.3 Calculate training steps and validation steps per epoch

Now, since we have more data from generating random data augmentations on the fly, as well as, time was a constraint, there was a need to speed up the training process. For this, training steps and validation steps per each epoch were added to our model.

3.3.5 Define Callbacks and Tuning Parameters

Callbacks are designed to be able to monitor the model performance in metrics at certain points in the training run. Callbacks perform some actions that might depend on those performances in the metric values.

The **Early Stopping** callback was set up to avoid overfitting and stop it if a validation accuracy is not maximized for 5 epochs. The **Model Checkpoint** was used to save weights of the model at frequent intervals only during training if the **val_loss** score has improved.

Also, the **Learning Rate Scheduler** was also used to define a function for decaying the learning rate. This function keeps the initial learning rate for the first specified epochs and decreases it exponentially after that.

3.3.5 Adding a new Head Classifier

Moreover, a new classification top model was added, which will be trained from scratch, on top of the VGG-16 pre-trained model so that we can repurpose the feature maps learned previously for the X-ray images.

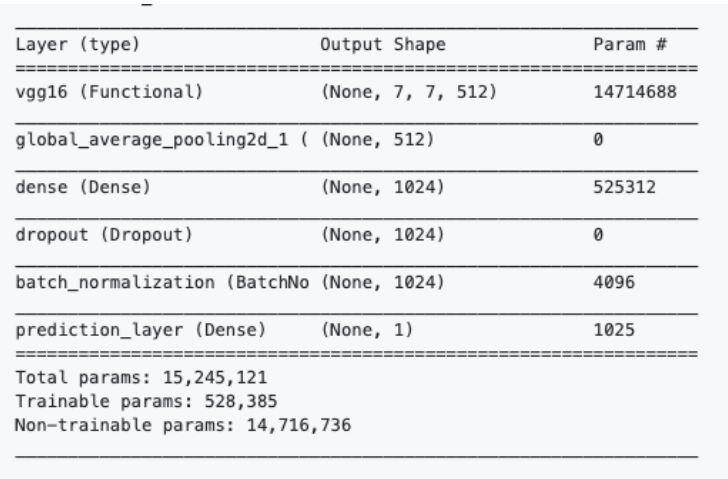
The new classifier will take **GlobalAveragePooling2D** layers exactly the same from the first model. This is set exactly to the size of the layer input so that the model ends with a convolutional layer that generates as many feature maps as the number of target classes and applies global average pooling to each in order to convert each feature map into one value.

Also, a **Dense** connected Neural Network layer of 1024 units with a rectifier linear unit or **relu** activation function was added. The model complexity was increased because we are

now using data augmentation and this will give us more training data.

Followed by a **Dropout** layer with a rate of 0.3 to help prevent overfitting since our model is more complex now. Then, a **BatchNormalization** layer was added to standardize the inputs of the network and also to accelerate the training, to try to reduce the number of epochs, to provide regularization and reduce generalization error.

Here is a depiction of the fine-tuned architecture that sits on top of the VGG-16 model.



Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
global_average_pooling2d_1 (	(None, 512)	0
dense (Dense)	(None, 1024)	525312
dropout (Dropout)	(None, 1024)	0
batch_normalization (BatchNo	(None, 1024)	4096
prediction_layer (Dense)	(None, 1)	1025

Total params: 15,245,121
Trainable params: 528,385
Non-trainable params: 14,716,736

Fig 11: Fine-tuned architecture.

Now, the model will be trained for 20 epochs and will have a new learning rate of 0.0001. The batch size 32 and the Adam optimizer will be the same as in the first model.

4.0 Results

4.1 Model Evaluation and Validation

The final model was supposed to run for 20 epochs, however, an early stopping callback was set up to avoid overfitting and thus the model ran for only 12 epochs. When the model was evaluated on the test data, the model **AUC score** was 97.83, and the **validation loss** of 0.197. The evaluation was performed on the test data and using validation steps to ensure that the same validation samples are used every time.

Let's take a look at the learning curves of the training and validation of both the **area under curve** and **precision recall** (figure 12).

Accuracy alone doesn't tell the full story when you're working with a class-imbalanced data set, like this one, where there is a significant disparity between the number of normal and pneumonia labels.

Therefore, we'll look at a better metric for evaluating class-imbalanced problems. Since, the **Area Under the Curve** (AUC) is the measure of the ability of a classifier to distinguish between classes. And, the higher the AUC the better the performance at distinguishing between the two classes will be. Our value for the AUC score is 97.83 and we can be confident that we have a high chance that our classifier will be able to detect more number of True positives and True negatives than False positives and False negatives.

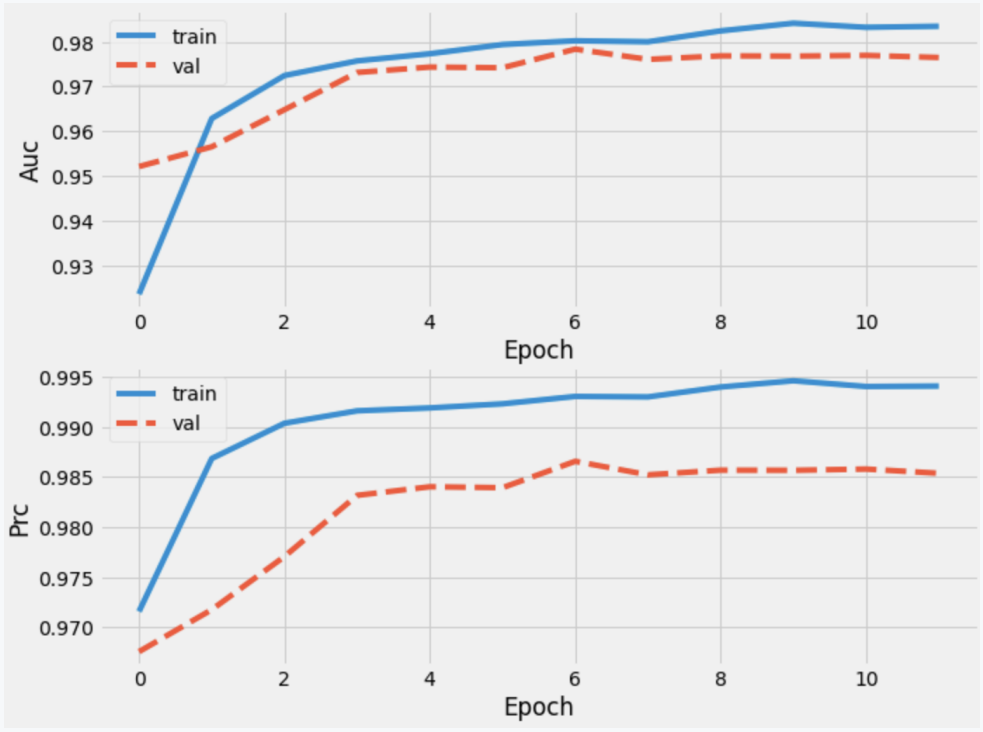


Figure 12: Learning curve for both AUC and Precision.

Moreover, a cut-off probability threshold of 0.5 was made to minimize the cost of **False positives** and **False negatives**. As mentioned earlier, a **true positive** is an outcome where the model correctly predicts the positive class (pneumonia). Similarly, a **true negative** is an outcome where the model correctly predicts the negative class (normal). Also, A **false positive** is an outcome where the model incorrectly predicts the **positive class**. As well as, a **false negative** is an outcome where the model incorrectly predicts the **negative class**.

The final model gave us a precision of 95.24, a recall of 92.31, a Sensitivity and Specificity of 92.31 and a F1-score of 0.90 for Normal and a F1-score of 0.94 for Pneumonia class.

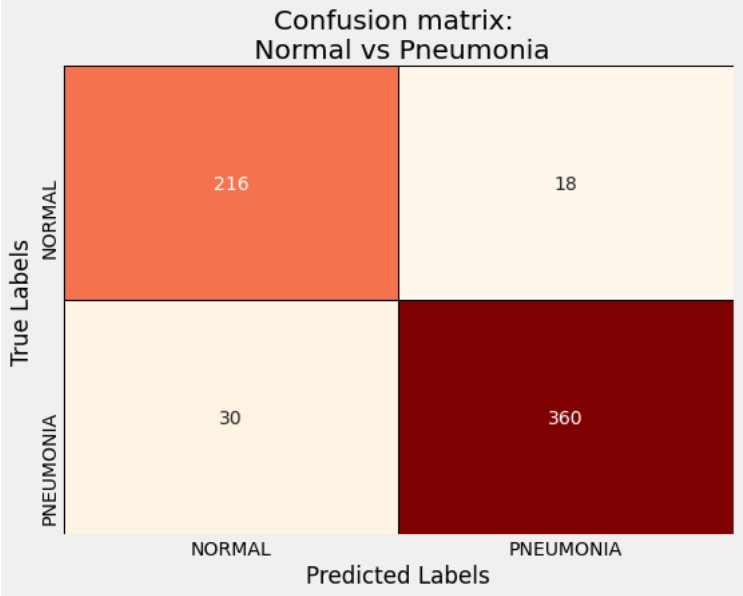


Figure 13: Confusion matrix for our final model.

The confusion matrix shows that the normal cases correctly detected (True Negatives) were 216. Also, the pneumonia cases correctly detected (True Positives) were 360. As well as, the normal cases incorrectly detected (False Positives) were 18 and the pneumonia cases incorrectly detected (False Negatives) were 30.

Moreover, the **AUC ROC** plot in figure 14 shows, at a glance, the range of performance the model can reach just by tuning the output threshold. As mentioned earlier, a scalar value of 0.5 was applied to the model predicted score in order to separate the positive class from the negative class. For example, our scalar value of 0.5 will show that If the classification threshold is 0.5, then predictor values above 0.5 are classified as positive and those below 0.5 are classified as negative. The model gave a **AUC ROC** score of 97.83 and we can be highly confident that a randomly chosen positive example is actually positive than that a randomly chosen negative example is positive.

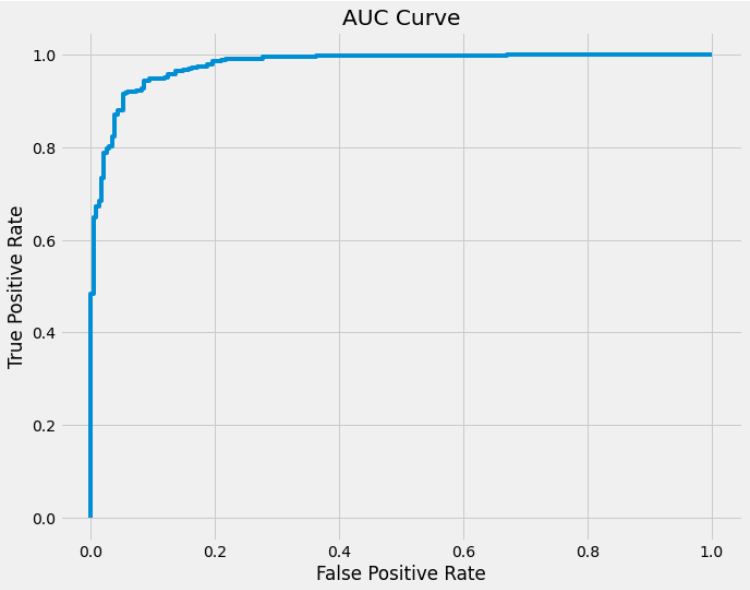


Figure 14: AUC ROC Curve.

4.2 Justification

Since we know that our data is imbalanced. Accuracy might not be the best model evaluation metric every time. It can convey the health of a model well only when all the classes have similar prevalence in the data. Therefore, when the class distribution is imbalanced, accuracy is not a good model evaluation metric. For this reason, I relied on the AUC ROC Score, Sensitivity, Specificity and looked at F1-score to seek a balance between Precision and Recall.

Therefore, a good F1-score means that you have **low false positives** and **low false negatives**, this way so we can correctly identify real threats and we will not be disturbed by false alarms. Also, a F1-score is considered perfect when it is 1, while the model is a total failure when it is close or at 0.

Furthermore, by comparing our final model's confusion matrix (figure 13) and our first model's confusion matrix (figure 15) we can come to the conclusion that our final model is a better predictor.

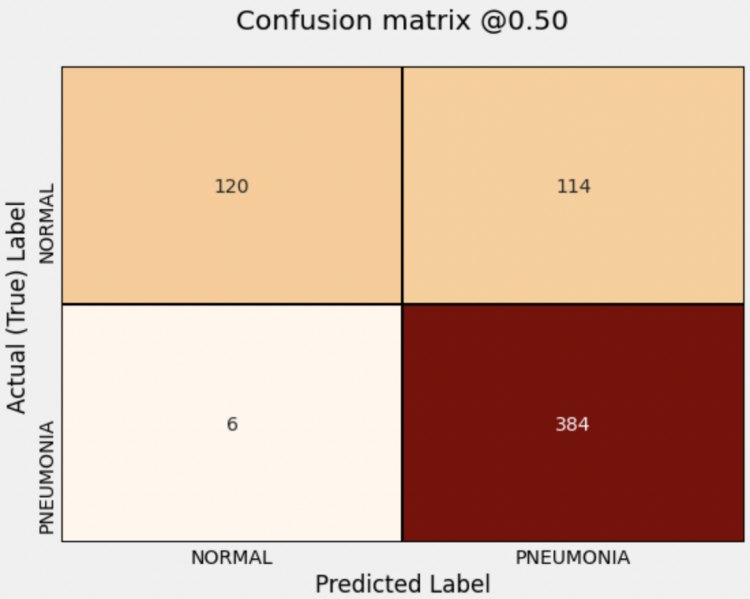


Figure 15: Base model or First model Confusion Matrix.

The base model got a F1-score of 0.67 for the normal class and a F1-score of 0.86 for the pneumonia class. In figure 15, we can see that it is predicting 6 false negatives which is very low but it's predicting a very high false positives rate of 114. There seems to be an imbalance here.

In comparison, our final model (see figure 13) reduced our false positives to 18 but increased our false negatives to 30. However, it seems that our final model reached a better balance. The question to answer here is why is the final model still better than our base model? We can further look at our AUC ROC score, Sensitivity and Specificity scores.

For our first model, our AUC ROC score was 74.87, our Sensitivity value was 98.46 and our Specificity value was 51.28. As mentioned earlier, this dataset is highly unbalanced and it reflects in the values given for both Sensitivity and Specificity for pneumonia and normal predictions. This is because the prevalence of pneumonia class in this dataset is higher than the normal class.

Moreover, the addition of data augmentation and estimating the class weights gave an overall better metric performance. The final AUC ROC score value was 97.83, the Sensitivity and Specificity values were the same value of 92.31 accordingly. Already we can see an improvement. We can determine that in 92.31% of the cases this model is predicting right

for both cases. The Sensitivity value indicates that out of 100 detections of pneumonia cases our model will predict class 1 correctly 92.31% of the time. That is, out of 100 pneumonia cases approximately 92 cases will be predicted right and 8 pneumonia cases will be wrongly predicted as normal cases. Also, sSpecificity indicates that out of 100 normal cases detected approximately 92 cases will be predicted correctly but it will miss 8 cases and will end up being predicted as pneumonia cases.

5.0 Web App Deployment

After training our final model and saving the entire model on a **.h5** file format. The web-app consisting of one page was ready to be constructed. For this, I leveraged the easy deployment method that Streamlit front-end framework offers. The final version of the web app can be depicted below:

Figure 16: An example of the front-end or the web app.

The aim of the web app is to facilitate X-ray technicians in the classification of pneumonia and normal cases. Below, some images were tested on the new deployed web-app to check how our classifier performed. The examples can be seen in figure 17 for an X-ray image that seems to be from a normal person on the top and from a pneumonia person on the bottom.

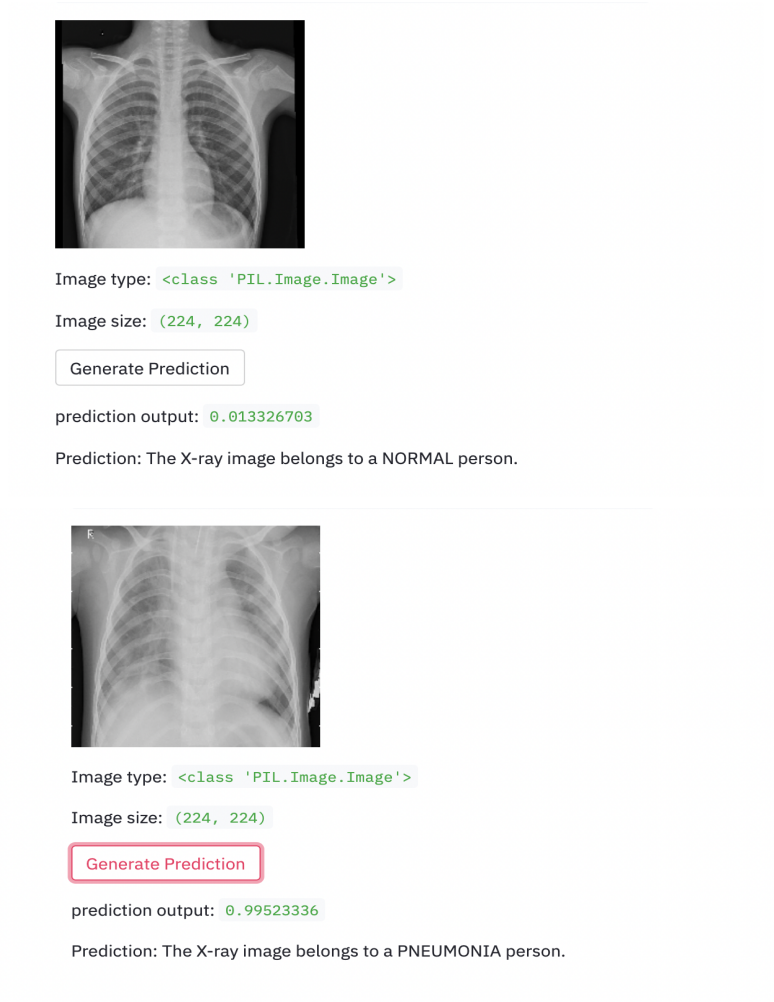


Figure 17: A normal (top) and a Pneumonia (bottom) sample.

An example of the web app deployed can be found via Streamlit Sharing and the github repository.

6.0 Conclusion

6.1 Reflection

It is important to note that I was only trying to predict if an image was prone to pneumonia or not. Yet, these images could also be further divided into four directories: CNV, DME, DRUSEN and NORMAL. The Pneumonia directories divided into CNV, DME, and DRUSEN have X-ray images that depict forms of bacteria. Thus, instead of making a binary classification model I could have created a multi-classification model. However, this could have made the exploratory data analysis, the model evaluation and results explanation much harder.

Moreover, by leveraging the power of transfer learning and computer vision a base model was created and trained on the images. Initially, the base model gave a Sensitivity score of 98.46 but the Specificity score was 51.28. As well as the AUC ROC score of 74.87. Thus, it was performing better in one class over the other. However, when fine-tuning the base model by adding hyperparameters, callbacks, class weights for unbalanced datasets and data augmentation, our model saw a major improvement. For instance, the final model gave a AUC ROC score of 97.83 and a Sensitivity and Specificity score of 92.31 respectively. The final performance of the model gave more confidence in the predicted power of our model and endup saving our final model into an .h5 file format to use it to create a web-app.

6.2 Improvement

As mentioned earlier, the original images had a disproportionate ratio of observations in each class. This is why accuracy was not used as the main metric. Also, the data was already split for us into train and test folders. A possible solution would have been to put all the images together back in one folder and make my own train/test split. Perhaps, by doing this, I could have a much better ratio for each class.

Also, in the data exploratory analysis process, I could have checked the images for further irregular compositions, looked for black or light background and for any repeated images

or values. By doing this, I could have eliminated images that were not actually helpful in the model predicting power and were only using space.

Furthermore, when fine-tuning on the VGG-16 base model was conducted, I only put a head classifier on top of the base model. Meaning, I trained the model using feature extraction and added a Dense layer of 1024 units. This was good enough to help me beat the initial base model metric. But, if I had unfreezed a few of the top CNN base layers and had joined both the new classifier layers and the last layers of the base model, I probably could have a better overall metric performance. This could have helped me further fine-tuned the higher-order feature representations in the final model in order to make them more relevant to predict pneumonia from the images.

References

1. https://neurohive.io/en/popular-networks/vgg16/
2. https://towardsdatascience.com/exploratory-data-analysis-ideas-for-image-classification-d3fc6bbfb2d2
3. https://data.mendeley.com/datasets/rscbjbr9sj/3
4. https://www.who.int/news-room/fact-sheets/detail/pneumonia
5. https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#evaluation-metrics
6. https://academic.oup.com/bjaed/article/8/6/221/406440
7. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
8. https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c
9. https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751