Abstract

Pneumonia is a common pulmonary disease that affects a large number of individuals worldwide, with a higher prevalence in less developed low-income regions. Chest X-ray images are a widely used diagnostic tool for the detection of pneumonia, although there is inherent variability in interpretation due to the subjective nature of visual analysis. The purpose of this research endeavor is to address this limitation by exploring automated approaches to pneumonia detection from chest X-ray images, while maintaining a high level of accuracy. I conducted a comprehensive investigation of various machine learning methods, including Support Vector Machines, Random Forest, Logistic Regression, several modified Convolutional Neural Networks, and the Vision Transformer. Moreover, I examined the utility of transfer learning techniques and assessed the performance of each of the different models. My results revealed that the proposed DenseNet model achieved an accuracy of 91.08% and an F1 score of 89.42% on the Guangzhou Women and Children's Medical Center dataset, indicating significant promise in this area of research.

1 Table of Contents

2 Introduction

Pneumonia is an infection, beginning with initial contact with a pathogenic microorganism and finally invading the lower respiratory tracts (Torres & Cillóniz, 2015). Dry coughs, chest pain, fever, and difficulty in breathing are major symptoms of this infection (Moine, 1994). Patients can acquire pneumonia from either the community or within the hospital setting. More than 15% deaths in children under the age of five are caused by it every year (WHO, 2019). Commonly seen in the underdeveloped region, early diagnosis and management can significantly reduce the fatality of this disease.

For the diagnosis, Doctors usually rely on multiple methods, such as, clinical examination, medical history, and chest x-rays. However, chest X-rays have recently become cheaper and more common due to the advancements, availability and cost reductions of medical equipment. From the comparison shown in figure 1, I can see that pneumonic X-ray (as shown in B) contains white spots in them. These spots are called infiltrates (Kundu, R. et al., 2021), which distinguishes it from healthy lungs. However, the subjective variability of these examinations can introduce inaccuracies for early detections (Neuman, M. I. et al., 2012)(Williams, G. J. et al., 2013). This introduced the need for an automated system for the detection of pneumonia.

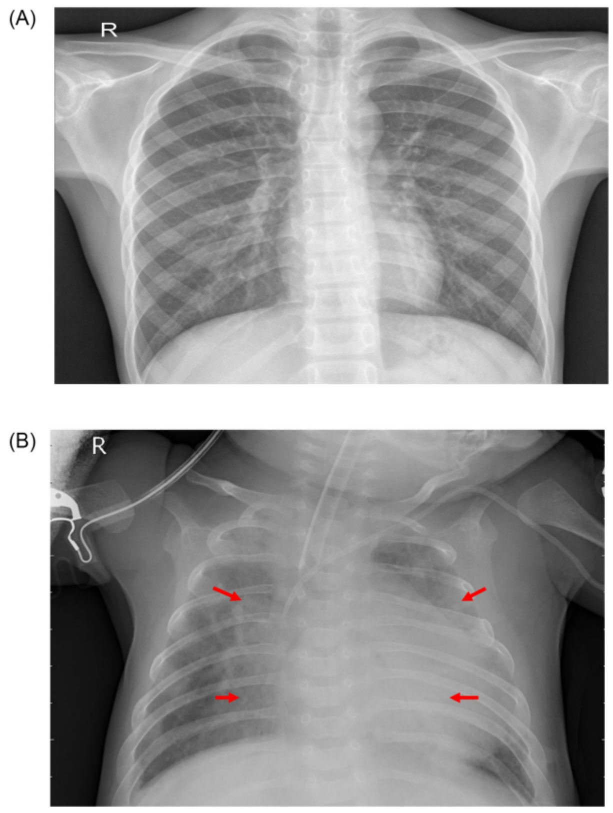

Figure 1: (A) Shows a radiograph of Healthy Lung
(B) Shows a radiograph of Pneumonia affected Lung

In recent years, there has been a significant increase in the usage of Computer Aided Design (CAD) and Deep Learning in medical cases. CAD systems have been developed to assist clinicians in the design of implants and prostheses, while Deep

Learning algorithms have been developed to aid in the diagnosis of medical conditions. CAD systems provide clinicians with the ability to design implants and prostheses that are tailored to the specific needs of the patient, which can lead to improved outcomes and reduced recovery times. In addition, CAD systems can help to reduce the risk of complications and errors associated with manual design processes. Deep Learning algorithms have shown promise in the diagnosis of medical conditions, particularly in the areas of radiology and pathology. These algorithms are capable of processing large amounts of medical data and identifying patterns that may not be apparent to human clinicians. This can lead to earlier and more accurate diagnoses, which can improve patient outcomes and reduce the burden on healthcare systems. Despite the potential benefits of CAD and Deep Learning in medical cases, there are also challenges to their adoption. These include the need for high-quality data and the requirement for extensive training of clinicians and technicians. However, with continued research and development, CAD and Deep Learning are likely to play an increasingly important role in the diagnosis and treatment of medical conditions.

2.1 Objectives

The primary aim of this study is to assess the efficacy of deep learning and machine learning techniques in classifying X-ray images as either indicative of pneumonia-affected lungs or healthy lungs. In order to achieve this goal, I examined several machine learning architectures and subjected them to a standardized evaluation process. I also employed various data preprocessing techniques in order to enhance the robustness of the models trained.

3 Background

With the progress of deep learning, numerous researchers have put forward various machine learning-based approaches to address diverse medical issues. Convolutional Neural Network (CNN) has already been extensively used with prosperous outcomes in the detection of medical conditions such as breast cancer, brain tumor, and the classification of different diseases from X-ray images (Kallianos, K., 2019). In order to propose solutions with X-ray images for detecting different diseases, the ChestX-ray8 dataset was presented(Wang, X., 2019), which contained radiograph images of 32717 patients with 108948 frontal X-ray images. It has been demonstrated that this dataset can be exploited to learn different features from images and can thus be successfully utilized to detect diseases. In this regard, CheXNeXt, a new CNN architecture with 121 layers (Rajpurkar, P., 2018), was proposed that can detect 14 different diseases, including pneumonia from frontal-view chest X-rays. In this study, multiple neural networks were initially trained to detect 14 different abnormalities from chest X-rays. Subsequently, an ensemble of these models was employed to enhance the overall architecture's accuracy by calculating the mean predictions of individual networks.

In another study (Chouhan, V., 2020), a novel deep learning framework for detecting pneumonia in chest X-ray images was introduced using transfer learning. Transfer learning is a method that utilizes pretrained neural network models on large-scale image datasets to extract features from images and feed them into a classifier for prediction. The paper compares five different pretrained models and then merges them into an ensemble model that achieves state-of-the-art performance with 96.4% accuracy on a dataset of unseen pneumonia X-ray images.

Furthermore, in a study conducted by Rohit et al. (Kundu, R., 2021), transfer learning was utilized to leverage pretrained models on large-scale image datasets and combine three convolutional neural network models: GoogLeNet, ResNet-18, and DenseNet-121. The paper assigns weights to the base models based on four evaluation metrics and utilizes a weighted average ensemble technique. The paper assesses the proposed method on two pneumonia X-ray datasets and achieves high accuracy and sensitivity rates. Additionally, it compares the results with state-of-the-art methods and conducts statistical tests to demonstrate the robustness of the approach with over 85% accuracy on different datasets.

4 Dataset

4.1 Dataset Collection

The present study was designed to explore the feasibility and effectiveness of employing deep learning techniques for detecting pneumonia in X-ray images of pediatric patients.
Specifically, 5863 Child X-ray images were collected from a cohort of pediatric patients at the Guangzhou Women and Children’s Medical Center (Karmany D.S., 2018), and were subsequently categorized into two classes based on their diagnostic status: normal and pneumonia affected splits as shown in Figure 2.

It should be noted that the X-ray images under consideration were collected as part of routine clinical care, and thus represent a large and diverse sample of pediatric patients. The study obtained approval from the Institutional Review Board (IRB) and Ethics Committee, and was conducted in compliance with the United States Health Insurance Portability and Accountability Act (HIPAA). The data curators also adhered to the principles outlined in the Declaration of Helsinki.
In order to ensure the reliability and validity of the subsequent classification analyses, only high quality and highly readable radiograph scans were retained for further analysis.

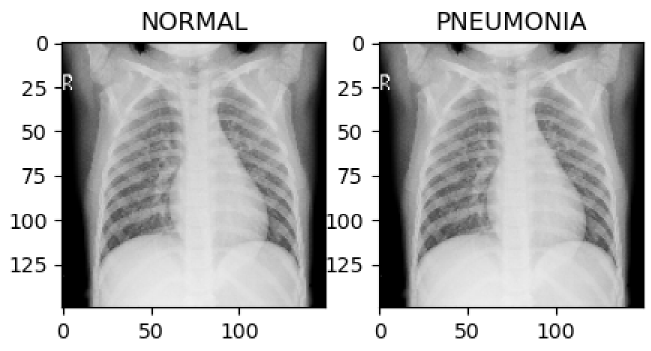

Figure 2: Representation of Normal vs Pneumonia affected lungs in my collected dataset.

In addition, two expert physicians independently evaluated each image and assigned a diagnostic label based on their professional judgment. To account for any potential grading errors, a third expert physician reviewed the classifications of the evaluation set and provided additional oversight.

4.2 Dataset Splits

The collected dataset was then divided into three splits: Train (containing 5216 samples), Test (containing 624 samples) and Validation (containing 16 samples). As shown in the figure 3, the train and test split contains a skew towards positive samples.

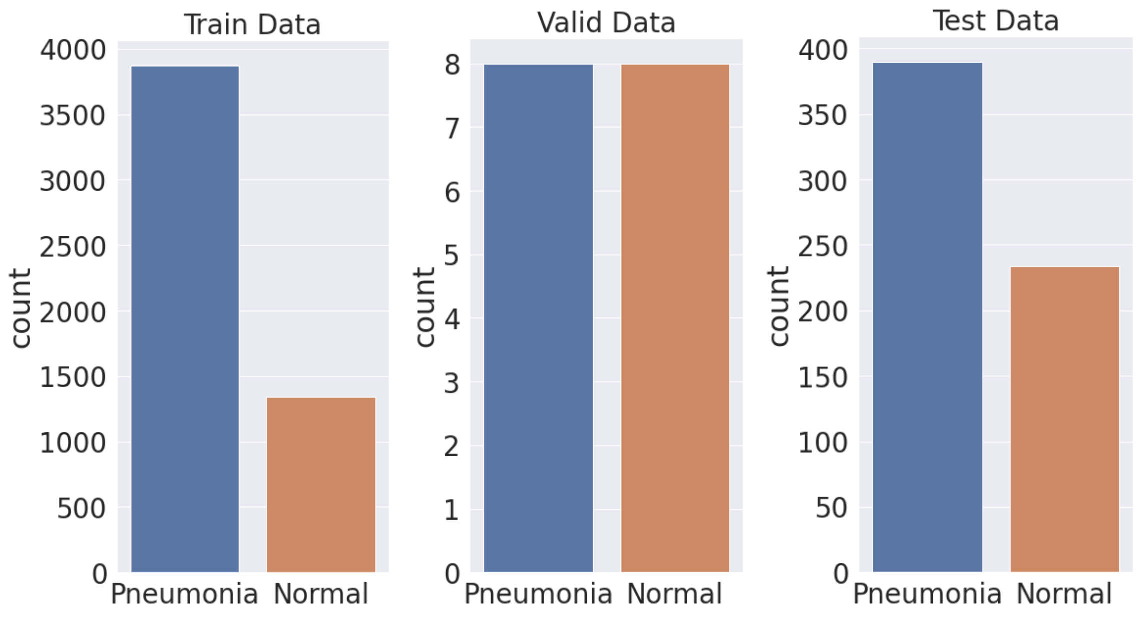

Figure 3: Dataset split distribution

4.3 Augmentation

To train my models, I also decided to experiment and visualize my models with different sets of augmentations. Data augmentation techniques can be employed to artificially expand the size of the training set and improve the accuracy of the model. These techniques involve creating new training examples by applying various transformations to the original images, such as rotations, translations, and scaling. By doing so, the model is exposed to a larger and more diverse range of data, which can help it to learn more robust features and reduce overfitting. Moreover, data augmentation can also help to reduce the risk of bias in the model by ensuring that it is not only trained on a narrow subset of the data. I represent a small sample subset of the augmentations step I have experimented with in the figure 4.

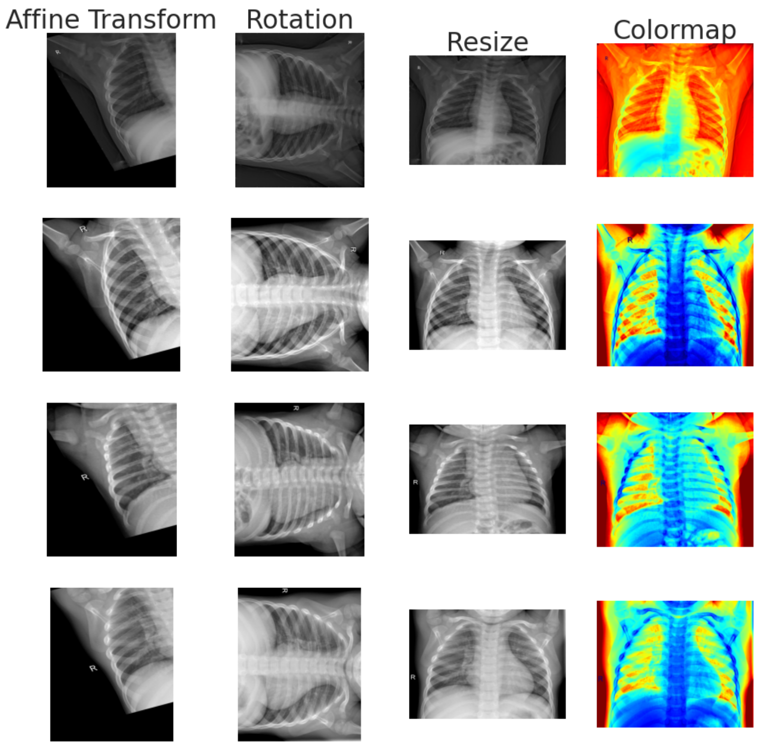

Figure 4: Example images after augmentation

4.4 Ethical concerns

Despite containing highly sensitive information about individual patients, the dataset used in this study was obtained with the explicit consent of the guardians of each patient. Furthermore, appropriate measures were taken to maintain the anonymity of the dataset, and domain experts were consulted to ensure the accuracy and validity of the labeling process.

5 Methodology

For my task I followed the pipeline as shown in figure 5:

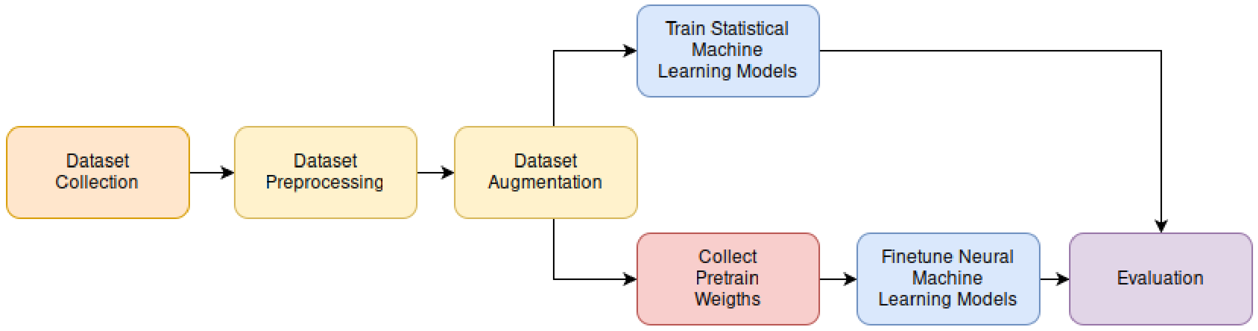

Figure 5: Project Pipeline

5.1 Dataset Preprocessing and Augmentation

Due to the requirement of computing a substantial number of variables for model training and inference using feature representation, the reduction of the original image dimensions has been implemented. This decision was made to facilitate accelerated training and to lessen the computational complexity. Concretely, the original 1024 x 1024 image dimensions were altered to 100 x 100 pixels. Additionally, various augmentation techniques and further model training procedures were subsequently applied to these downsampled images.

To prepare my dataset for training, I introduced augmentation parameters during the data loading stage as shown in Table 1.

Step	Value
Rotation	20
Width Shift	0.1
Shear	0.1
Zoom	0.1
Samplewise Center	True
STD Normalization	True

Table 1: Augmentation Parameter

The **rotation** parameter controls the range of random rotations applied to the image, in degrees. The **width shift** parameter controls the range of random horizontal shifts applied to the image, as a fraction of the total width. The **Shear** parameter controls the range of random shearing transformations applied to the image. Finally, the **Zoom** parameter controls the range of random zooms applied to the image.

Additionally, the **Samplewise Center** parameter centers each image around its mean pixel value, and the **STD Normalization** parameter standardizes each image by dividing each pixel value by the standard deviation of all pixel values in the dataset.

5.2 Model Training

After carefully preprocessing my dataset and addressing any imbalances present, my research focuses on building machine learning models to evaluate their effectiveness. To obtain a comprehensive understanding of the machine learning paradigm, I explore both traditional or statistical models and modern neural network-based deep and shallow models. This enables me to compare and contrast the strengths and weaknesses of each approach, and to determine which models are most appropriate for specific applications.

5.2.1 Traditional Machine Learning Methods

Traditional machine learning is a field of study that involves building models using statistical techniques and algorithms. This involves analyzing data and identifying patterns and relationships between variables using algorithms like regression, decision trees, and clustering. These models have been effective in many applications like image classification, speech recognition, and fraud detection where the data is well-structured and patterns can be easily identified. However, they may struggle with more complex or unstructured data, such as natural language processing or image recognition tasks. Also, developing these models may require significant human expertise to choose relevant features and tune the algorithms, which can be costly and time-consuming.

In this study I use the traditional Machine Learning models as a baseline. For this, I explored the following models. I also described the major parameters used for building these models.

5.2.1.1 SVM Classifier:

SVM (Support Vector Machine) Classifier is a machine learning algorithm used for classification tasks. It is a discriminative and supervised learning algorithm that aims to find the hyperplane that best separates the classes in the feature space.

The algorithm works by mapping the input data into a higher-dimensional space using a non-linear transformation. In this higher-dimensional space, the algorithm finds the hyperplane that best separates the data into the different classes. The hyperplane is chosen such that it maximizes the margin, which is the distance between the hyperplane and the closest data points from each class as shown in figure 6.

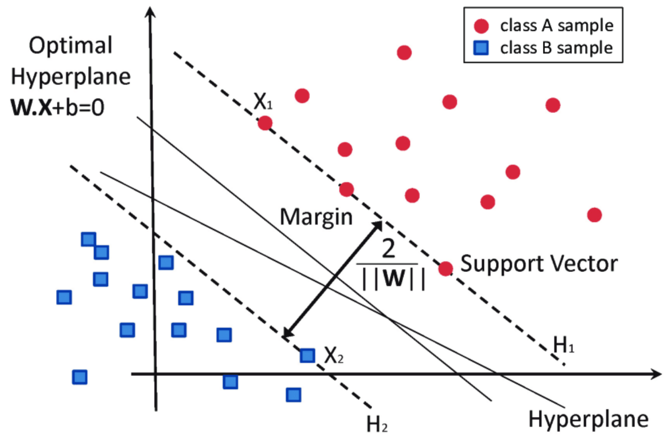

Figure 6: Example demonstration of SVM algorithm

Once the hyperplane is found, the algorithm uses it to make predictions for new data points. The algorithm assigns a class label to the new data point based on which side of the hyperplane it falls on.
For my training, I used the following training parameter set described in table 2.

Parameter	Value
Regularization	0.1
Kernel	Radial basis function

Table 2: Training Parameter set for SV

5.2.1.2 Logistic Regression

Logistic Regression as shown in figure 7 is a statistical technique used for classification problems, where the goal is to predict the probability of an event occurring based on one or more input variables. The technique is particularly useful when the outcome variable is binary, meaning it can take only two possible values, such as 0 or 1.
In Logistic Regression, the input variables are combined linearly using coefficients, and the result is passed through a logistic function, which outputs a value between 0 and 1. This value represents the predicted probability of the event occurring.

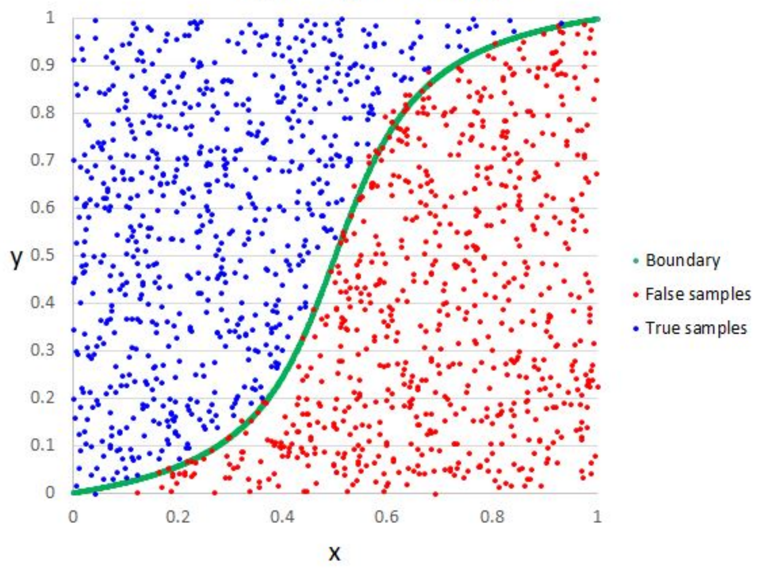

Figure 7: Example demonstration of Logistic algorithm

The Logistic Regression model estimates coefficients by maximizing the likelihood function through iterative algorithms like gradient descent. Using these coefficients, the model predicts the probability of an event for new data points. A threshold, often 0.5, determines whether the event is predicted to occur (output is 1) or not occur (output is 0).
For my training, I used the following training parameter set described in table 3.

Parameter	Value
Regularization	1
Solver	LibLinear

Table 3: Training Parameter set for Logistic Algorithm

The solver parameter specifies the numerical optimization algorithm to use in order to optimize the logistic regression model's cost function during training. I use a linear iterative solver suitable for large datasets to minimize the cost function.

5.2.1.3 KNN (K Nearest Neighbor Classifier)

The K-Nearest Neighbor (KNN) Classifier is a non-parametric and lazy learning algorithm used for classification tasks. It compares an input data point with the k closest data points in the training set using a distance metric like Euclidean or Manhattan distance. Based on the majority class of the k neighbors, the algorithm assigns a class label to the input data point. In other words, if most of the k nearest neighbors belong to a certain class, then the input data point is assigned to that class as shown in figure 8.

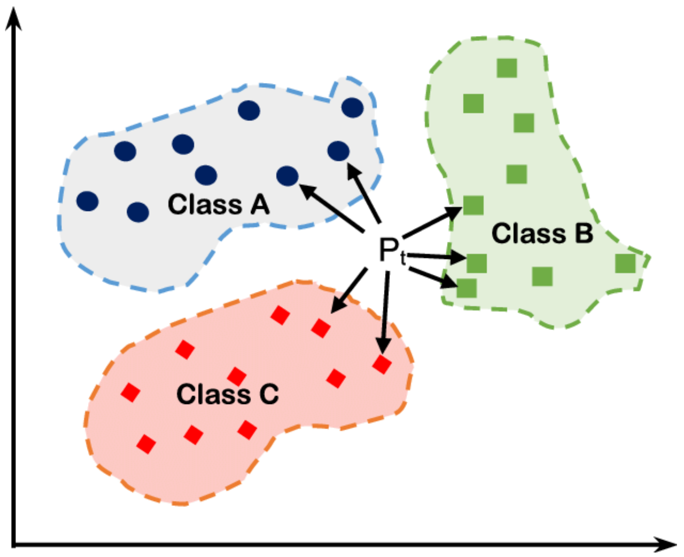

Figure 8: Example demonstration of KNN algorithm

One of the main advantages of the KNN Classifier algorithm is that it is simple and easy to implement. It can also handle both binary and multi-class classification problems and can be used with any distance metric. However, the algorithm can be computationally expensive and may require a large amount of memory to store the training data.
For my training, I used the following training parameter set described in table 4.

Parameter	Value
Number of Neighbors	5

Table 4: Training Parameter set for KNN

5.2.1.4 Random Forest

Random Forest is an ensemble machine learning algorithm that improves model performance for classification and regression tasks. It works by creating multiple decision trees from subsets of features and data samples randomly selected from the original dataset, using the "bootstrap aggregating" or "bagging" method, as shown in Figure 9. This reduces overfitting and improves the model's generalization ability. The decision trees are trained on different subsets of data and features to ensure diversity and independence. The algorithm evaluates the importance of each feature by measuring its contribution to the model's accuracy.

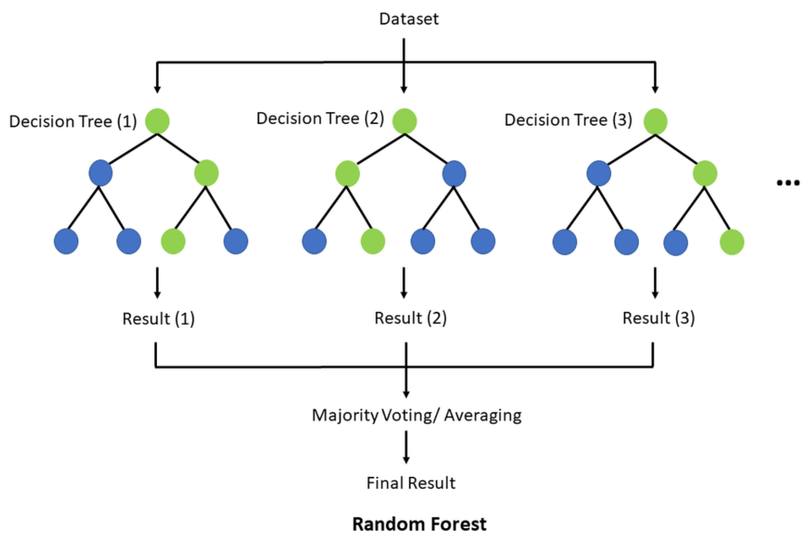

Figure 9: Example demonstration of Random Forest algorithm

One of the main advantages of the Random Forest algorithm is that it is highly accurate and robust, even when dealing with noisy or missing data. It also provides a measure of feature importance, which can be useful in understanding the underlying patterns and relationships in the data.
For my training, I used the following training parameter set described in table 5.

Parameter	Value
Max Depth	10
Number of Estimators	150

Table 5: Training Parameter set for Random Forest

The “Number of Estimators” parameter determines the number of decision trees that will be created in the forest. The max_depth parameter of Random Forest controls the maximum depth of the decision trees in the ensemble, and can be used to balance between overfitting and underfitting.

5.2.2 Neural Machine Learning Methods

Neural machine learning uses artificial neural networks to learn from data and make predictions or decisions. These networks mimic the human brain's structure and function, with interconnected layers of neurons processing information. Neural machine learning is popular because it can handle complex and unstructured data, and deep neural networks can learn multiple levels of abstraction.

There are two ways to build neural models: from scratch or using pre-trained weights. Building a model from scratch requires designing a neural network architecture, initializing weights and biases, and training the network on a specific dataset. This approach provides flexibility in customizing the architecture and tuning hyperparameters for optimal performance. In contrast, using pre-trained weights involves starting with an existing neural network architecture that has been trained on a similar task and fine-tuning it on a smaller, specific dataset. This approach, called transfer learning, is useful when the dataset is small or a related task needs to be learned.

For both types of models, I have used a fixed set of hyper parameters, which I describe below:

- **Optimizer:** In neural networks, optimizers are algorithms that help adjust the weights and biases of the network during training in order to minimize the error . There are several popular optimizers used in neural networks, including stochastic gradient descent (SGD), Adam, Adagrad, RMSProp, and more. For my classification task, I have used the Adam optimizer that can adjust the learning rate for each weight in the network based on the magnitude of its gradients and a running estimate of the second moment of the gradients.
- **Loss Function:** A loss function (also known as a cost function or objective function) is a mathematical function that measures the difference between the predicted output of the neural network and the true output (or target) for a specific input.The goal of a deep learning model is to minimize the value of the loss function, which means that the predicted output is as close as possible to the true output. In my task, I have used the Binary Cross-Entropy loss function.
- **Learning Rate:** Learning rate determines how much the weights and biases are updated in response to the gradient of the loss function with respect to those parameters. I have used a learning rate of 0.001.
- **Batch Size:** Batch size is an important hyperparameter in neural model training that controls the number of samples or data points that are processed by the model in each iteration during training. The choice of batch size depends on various factors, including the size of the dataset, the complexity of the model, the available computational resources, and the desired tradeoff between computational efficiency, memory usage, and generalization performance. Given my computational resources, I have used a batch size of 32 - which may be considered optimal given the size of the dataset.
- **Class Weight:** Class weights are used in neural model training to address class imbalance, which is a common problem in many machine learning tasks - such as ours. When one or more classes in the dataset are significantly underrepresented compared to the other classes, it gets difficult for the model to learn to classify the minority class(es) accurately. To address this issue, class weights can be used to give more importance to the minority class(es) during training, by assigning a higher weight to the minority class(es) in the loss function. The class weights are calculated based on the relative frequency of each class in the training set, where the weight of each class is inversely proportional to its frequency.
- **Activation Function:** Activation functions are a crucial component of neural models that introduce non-linearity to the output of a neuron or a layer. An

activation function takes the weighted sum of the inputs to a neuron, adds a bias term, and applies a non-linear transformation to produce the output of the neuron. There are various types of activation functions used in neural models, including sigmoid, tanh, ReLU (Rectified Linear Unit), LeakyReLU, softmax, and more. In my models, I have combined different combinations of these functions across the different layers, as appropriate.

5.2.2.1 From Scratch: CNN

Convolutional Neural Networks (CNNs) are a type of deep neural network that excel at tasks such as image classification, object recognition, and image segmentation. They mimic the visual processing system of animals, specifically the visual cortex in the brain. As shown in Figure 10, a CNN is composed of several layers: convolutional, pooling, and fully connected. The convolutional layer applies a set of filters to the input image to extract pertinent features, generating a feature map. The pooling layer downsamples the feature maps to reduce their size and retain important information. The final fully connected layer classifies the image by taking the output of the previous layers and passing it through fully connected nodes. During training, the CNN adjusts the filter values and weights to minimize the error between predicted and actual output. This is accomplished using a loss function, such as cross-entropy, and an optimization algorithm, such as stochastic gradient descent.

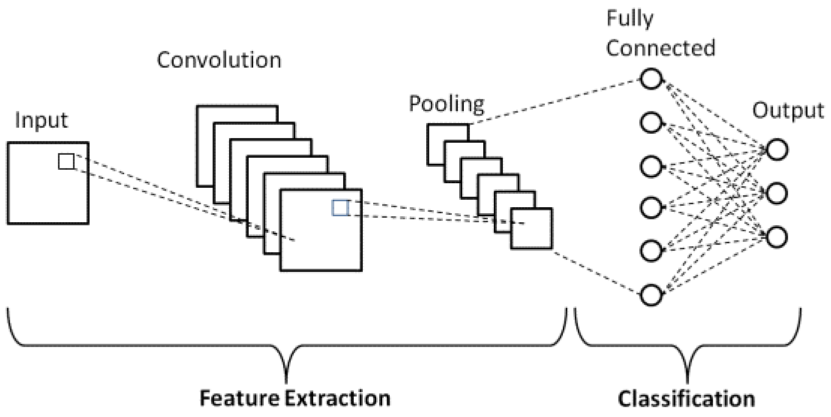

Figure 10: Example of a CNN Model

For my training, I used the following training parameter set described in table 6.

Parameter	Value
Optimizer	AdamW
Learning Rate	0.001
Loss Function	Binary Crossentropy

Table 6: Training Parameter set for CNN

5.2.2.2 Usage of Pretrained Weights

Pretrained weights in machine learning refer to previously learned weights by a model on a related task that are saved for later use. This method, known as transfer learning, uses pretrained weights as a starting point for training new models on related tasks, especially when data and time are limited. Utilizing pretrained weights for image classification is advantageous as it incorporates knowledge acquired by other models and improves the model's performance. Additionally, it helps prevent overfitting by improving the model's generalization performance. For this reason, I initialized all of my following models with pretrained weights.

5.2.2.2.1 DenseNet

DenseNet is a convolutional neural network architecture designed in 2017 for image classification and other computer vision tasks. It addresses the issue of vanishing gradients by allowing for better gradient flow throughout the network. It consists of several dense blocks, which have multiple layers (Figure 11). Each dense block concatenates the output feature maps from previous layers and feeds them into the current layer, creating a dense connectivity pattern between the layers. Additionally, DenseNet includes transition layers to reduce the size of feature maps between dense blocks. During training, DenseNet adjusts the filters and weights in the fully connected layers to minimize error between the predicted and actual output using a loss function and optimization algorithm, such as cross-entropy and stochastic gradient descent, respectively.

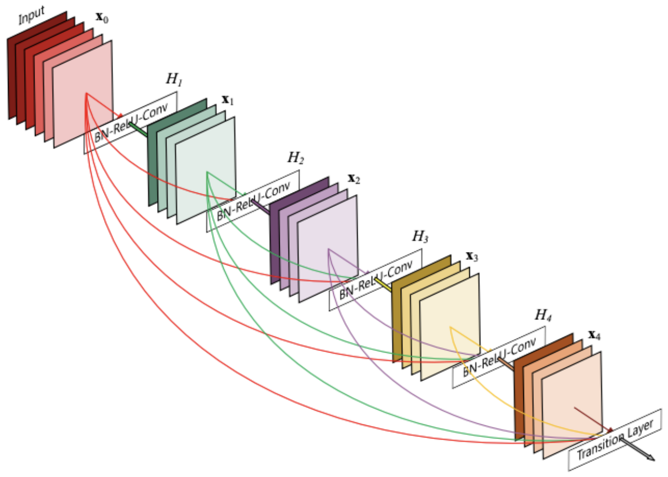

Figure 11: Example of a DenseNet Model

DenseNet has several advantages over other CNN architectures, such as ResNet and Inception. It requires fewer parameters, which reduces the risk of overfitting and makes the model more memory-efficient. It also achieves state-of-the-art performance on a variety of computer vision tasks, including image classification, object detection, and semantic segmentation.

For my training, I used the following training parameter set described in table 7.

Parameter	Value
Weights	ImageNet
Learning Rate	0.001
Optimizer	AdamW
Loss Function	Binary Crossentropy

Table 7: Training Parameter set for DenseNet

5.2.2.2.2 VGG-16

VGG-16 is a CNN architecture with 16 layers, consisting of 13 convolutional layers and 3 fully connected layers. Each convolutional layer in VGG-16 has a fixed kernel size of 3x3 and a stride of 1. This enables it to capture small features in the input image, with the output of each layer passed through a ReLU activation function for non-linearity (as shown in Figure 12). Max pooling layers are used to downsample the feature maps, making the model more efficient and reducing the risk of overfitting. Finally, the fully connected layers take the output from the convolutional and pooling layers, flatten it into a one-dimensional vector, and use fully connected nodes to output the final classification probabilities.

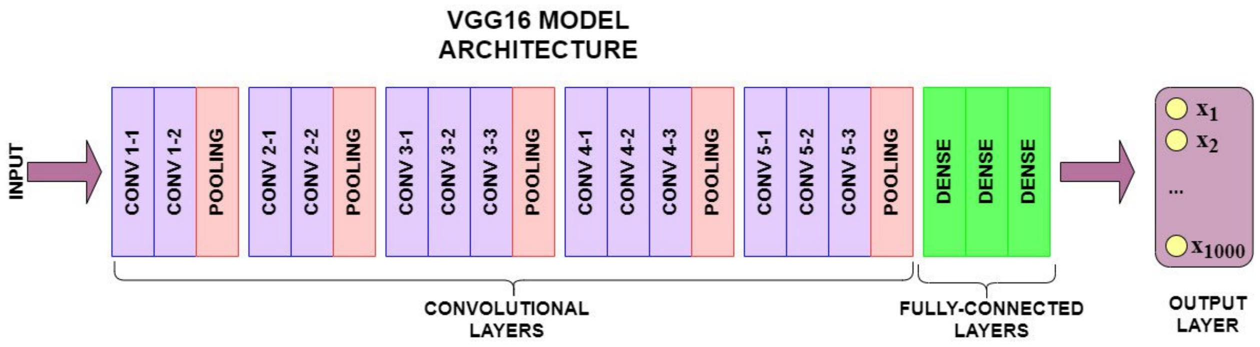

Figure 12: Example of a VGG16 Model

During training, the VGG-16 network learns to adjust the values of the filters in the convolutional layers and the weights in the fully connected layers in order to minimize the error between the predicted output and the actual output. This is done using a loss function, such as cross-entropy, and an optimization algorithm, such as stochastic gradient descent.
For my training, I used the following training parameter set described in table 8.

Parameter	Value
Weights	ImageNet
Learning Rate	0.001
Optimizer	AdamW
Loss Function	Binary Crossentropy

Table 8: Training Parameter set for VGG-16

5.2.2.2.3 InceptionNet

InceptionNet, also known as GoogLeNet, is a CNN architecture created in 2014 for image classification tasks. Its goal is to achieve high accuracy while being computationally efficient. The architecture contains multiple inception modules that combine various convolutional layers of different sizes and types to capture features of varying complexities (Figure 13). This enables the network to classify images better. InceptionNet also includes auxiliary classifiers that are used to improve the gradient flow and prevent overfitting. During training, InceptionNet learns to adjust the values of filters and weights in fully connected layers using optimization algorithms such as stochastic gradient descent and loss functions like cross-entropy.

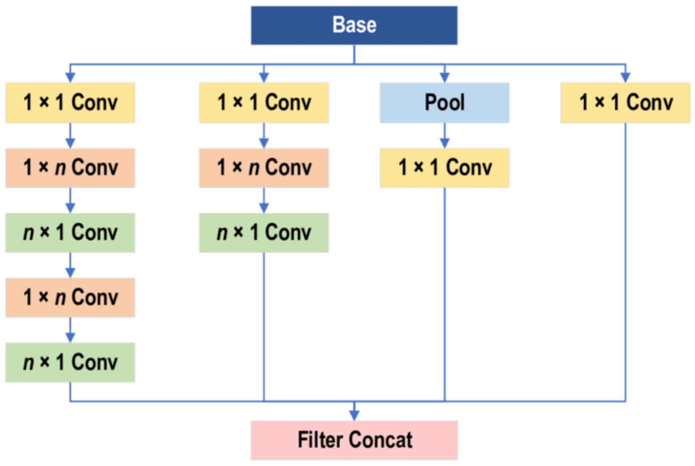

Figure 13: Example of a InceptionNet Model

InceptionNet has several advantages over other CNN architectures. It is computationally efficient due to its use of 1x1 convolutional layers, which reduce the number of parameters and computation required. It also achieves state-of-the-art performance on a variety of visual recognition tasks, including image classification, object detection, and semantic segmentation.

For my training, I used the following training parameter set described in table 9.

Parameter	Value
Weights	ImageNet
Learning Rate	0.001
Optimizer	AdamW
Loss Function	Binary Crossentropy
InceptionNet Version	3

Table 9: Training Parameter set for InceptionNet

5.2.2.2.4 ResNet

ResNet was created to tackle the issue of vanishing gradients that can occur when training deep neural networks. It uses residual blocks that contain several convolutional layers and a shortcut connection that bypasses the convolutional layers - as shown in Figure 14, enabling the gradient to flow directly from the input to the output. This helps prevent the vanishing gradient problem. ResNet also employs global average pooling and a fully connected layer for classification. The global average pooling layer reduces the spatial dimensions of the feature maps to one value while retaining the most important features. The fully connected layer maps the output of the global average pooling layer to the final class probabilities. To minimize the error between the predicted output and the actual output, ResNet adjusts the filter values in the convolutional layers and the weights in the fully connected layer during training, using a loss function like cross-entropy and an optimization algorithm like stochastic gradient descent.

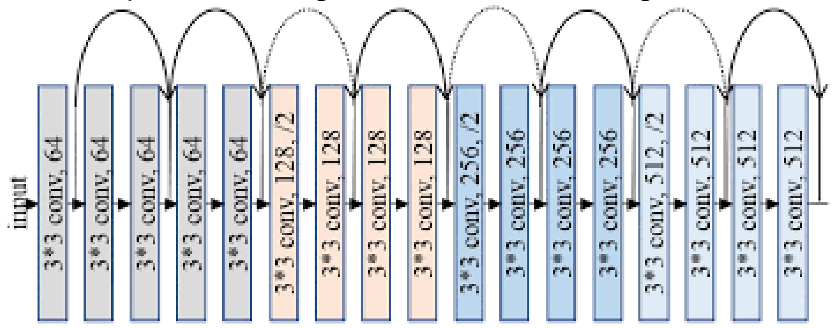

Figure 14: Example of a ResNet Model

ResNet has several advantages over other CNN architectures. It can be trained to very deep depths, achieving state-of-the-art performance on a variety of visual recognition tasks, including image classification, object detection, and semantic segmentation. The use of residual blocks allows for faster convergence and better accuracy compared to traditional CNN architectures. It also has a relatively simple architecture, which makes it easy to understand and modify.
For my training, I used the following training parameter set described in table 10.

Parameter	Value
Weights	ImageNet
Learning Rate	0.001
Optimizer	AdamW
Loss Function	Binary Crossentropy
ResNet Implementation	ResNet50

Table 10: Training Parameter set for ResNet Model

5.2.2.2.5 ViT

VIT (Vision Transformer) is a deep learning architecture that utilizes the Transformer architecture from natural language processing to model long-range dependencies in sequential image data. Instead of processing the input image as a grid of pixels, VIT processes it as a sequence of patches that are flattened and embedded using a learned linear projection, resulting in a lower-dimensional vector representation. The sequence of embedded patches is then processed by a stack of transformer blocks, each performing multi-head self-attention and feed-forward operations. During training, VIT learns to adjust the linear projection and transformer block weights using a loss function, such as cross-entropy, and an optimization algorithm, such as stochastic gradient descent.

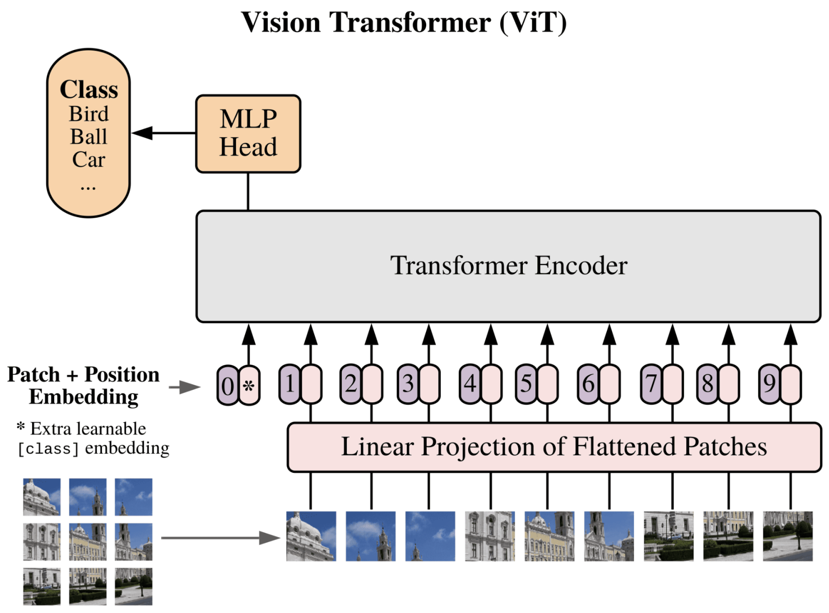
Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches

Figure 15: Example of a Vision Transformer Model

VIT has several advantages over traditional convolutional neural networks (CNNs) for image classification tasks. It can handle images of arbitrary sizes and aspect ratios, and can learn to recognize objects based on their global structure rather than their local features. It is also more computationally efficient than CNNs for very large images, as it can process each patch in parallel.

For my training, I used the following training parameter set described in table 11.

Parameter	Value
Weights	ImageNet
Pretrained	True
Include Top	True
Loss Function	Binary Crossentropy
Optimizer	AdamW
Learning rate	0.001

Table 11: Training Parameter set for ViT

5.2.3 Model Training Modification

To ensure the best learning gain and training for the best weights, I took two significant measures while training my model.

5.2.3.1 EarlyStopping Callback

Early stopping is a technique in machine learning that prevents overfitting by monitoring a model's performance on a validation dataset during training. Overfitting occurs when a model becomes too complex and starts memorizing the training data instead of learning the underlying patterns. The validation dataset is a separate subset of the training data used to evaluate the model's performance on unseen data. If the performance on the validation dataset stops improving or starts to degrade, the training is stopped early, and the best performing model on the validation dataset is selected as the final model.

5.2.3.2 Reduce Learning Rate on Plateau

"Reduce Learning Rate on Plateau" is used in deep neural network training to enhance performance and convergence. During training, the model is optimized using gradient descent, which involves adjusting the weights of the model. The learning rate determines the step size of these weight updates. Setting the learning rate too high or too low can cause issues such as unstable training or a slow convergence. "Reduce Learning Rate on Plateau" helps solve these issues by dynamically adjusting the learning rate based on the validation loss. When the validation loss doesn't improve for a set number of epochs, the learning rate is reduced by a factor. This ensures that the optimization process takes smaller steps towards optimal weight values and avoids overshooting.

6 Results

6.1 Evaluation Mtrics

Evaluating the performance of trained machine learning models is a very important way to get good performance from a machine learning model. While there are many effective evaluating function out there, I have opted for 3 different evaluation function as explained below:
Recall is the proportion of true positives (correctly predicted positive instances) out of all actual positive instances. It measures the ability of the model to identify all positive instances, indicating the completeness of the model's predictions.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Precision is the proportion of true positives out of all predicted positive instances. It measures the accuracy of the model's positive predictions, indicating the correctness of the model's predictions.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

F1 score is a harmonic mean of precision and recall, giving equal weight to both metrics. It is a measure of the model's overall ability to correctly classify positive instances while minimizing false positives and false negatives.

$$\text{F1 Score} = \frac{2. Precision * Recall}{Precision + Recall}$$

In general, a high accuracy or F1-Score value is desirable, but it is also important to balance this with other factors such as model interpretability, generalizability to different patient populations and imaging settings, and potential risks of false positives and false negatives. The acceptable values of these metrics depend on the specific application and context of the X-ray classification task. Generally, a diagnostic accuracy of at least 80-85% is considered acceptable for a machine learning model to be used in practice for X-ray classification tasks. In the medical domain, it is also important to consider the balance between sensitivity and specificity. A high sensitivity is desirable to minimize false negatives, as missing a true positive can have serious consequences in a medical context. On the other hand, a high specificity is necessary to minimize false positives, which can lead to unnecessary follow-up tests or treatments.

6.2 Result Analysis

In the context of classification modeling, evaluating the performance of the model is of utmost importance. Two key metrics used to evaluate the performance of such models are precision and recall. Precision measures the proportion of true positives in the

predicted positive samples, while recall measures the proportion of true positives in the actual positive samples.
In addition, the F-1 Score is used to assess the overall performance of the classification model by combining both precision and recall into a single metric, which is calculated as the harmonic mean of precision and recall.
By utilizing these metrics, it is possible to determine the types of errors the model is making and to make necessary adjustments to improve its performance. Consequently, each of these metrics plays a crucial role in assessing and refining classification models.
In the table 12, I present the results of my experimentations with different models:

Model Name	Precision	Recall	F1-Score
Support Vector Machine (SVM)	0.839502	0.690171	0.697793
Logistic Regression	0.815534	0.512821	0.411905
K-Nearest Neighborhood (KNN)	0.839599	0.667094	0.668048
Random Forest	0.848161	0.674786	0.677841
CNN	0.807108	0.596581	0.566945
DenseNet	**0.910844**	**0.883761**	**0.894247**
VGG16	0.715742	0.634615	0.633454
InceptionNet	0.804491	0.810256	0.807031
ResNet	0.312500	0.500000	0.384615
Visual Transformer (ViT)	0.187500	0.500000	0.272727

Table 12: Model Score Comparison

The experimental results indicate that the DenseNet architecture, a modified version of Convolutional Neural Networks (CNN), outperforms other models in all the evaluation metrics for our classification task with precision of 91%, recall of 88.38% and F-1 Score of 89.42%. In terms of statistical models, the Support Vector Machine (SVM) achieved the highest precision of 83.95% and F-1 score of 69.78%. It is noteworthy that despite their high density, complexity, and contextual awareness, ResNet and ViT exhibited poor performance compared to other models. This observation suggests that the number of samples in the dataset may have contributed to this outcome.

7 Analysis and Discussion

The detection of pneumonia from chest x-ray is a classification problem that involves a significant amount of data points to work with. Intuitively, it is expected that larger and denser models will perform better. In line with this expectation, neural networks have been found to outperform classical models. However, it should be noted that the performance gain of neural networks seems to plateau after reaching its highest score in DenseNet.
In the context of pneumonia detection in chest X-ray images, DenseNet's unique architecture may have contributed to its superior performance. Dense connections between layers allow for the efficient propagation of information through the network, enabling the model to effectively capture complex patterns and features in the images. Its ability to capture both global and local image features effectively is important for this task because pneumonia can appear in different areas of the lung and can have different sizes and shapes.
In addition, DenseNet has a smaller number of parameters compared to other popular deep neural network architectures such as ResNet and Inception, which can be advantageous in terms of computational efficiency and reduced risk of overfitting. This can be especially important for medical imaging tasks where large datasets may be limited.

Given the intended use of the models in various settings, it is recommended to employ both the best-performing classical and neural models based on the specific needs of the system. Moreover, the explainability of the models can be better explored for the neural models, which can be of great assistance to medical professionals in understanding the model's decision-making process.

It is important to note, however, that the models were trained solely on chest x-ray images of children in a specific region and with a limited dataset. Therefore, their robustness for adult x-ray images from different regions is questionable.

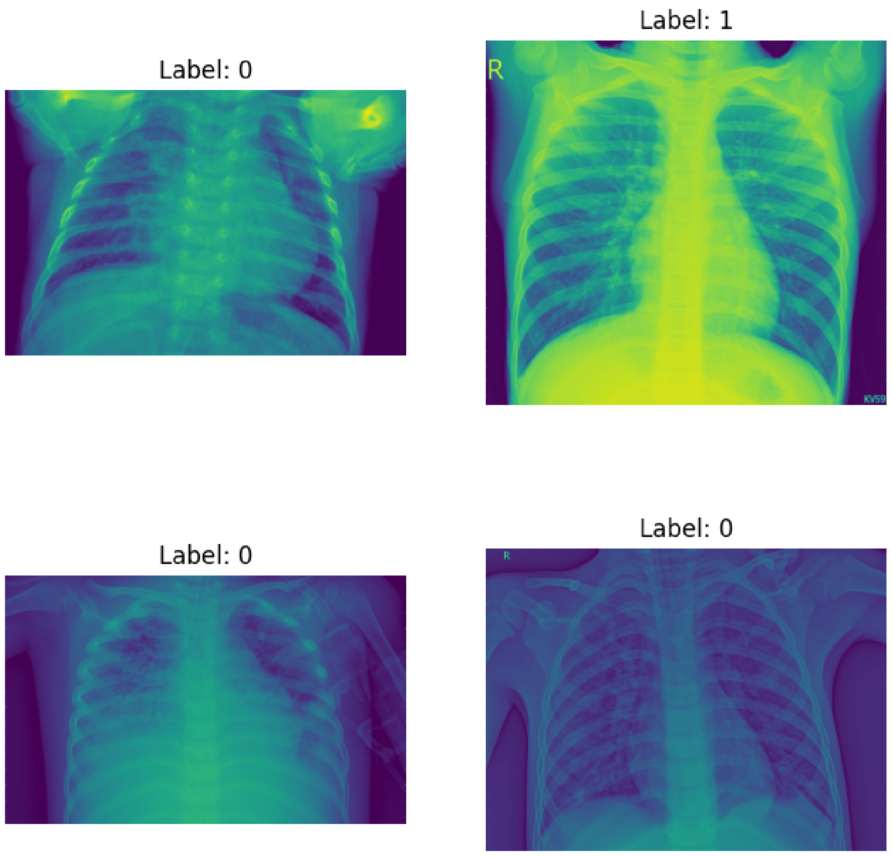

Figure 16: Examples of mislabelled instances. Label 0 indicates a Normal XRay and 1 indicates Pulmonary Infection/s

Additionally, while the best-performing models achieved an accuracy of 89.42%, this is still far from a perfect diagnosis. Among the 624 instances of test images, our best performing model mislabeled 80 samples. The model predicted more NORMAL images as infected (48), than the opposite (32). I have demonstrated a few example mislabeled images in Figure 16. While it is tough to pin-point why the model performed in this manner, it can be said without hesitation that relying solely on the model without seeking a third expert opinion may prove to be detrimental to patients.

8 Conclusion

The aim of this study was to investigate the effectiveness of deep learning models for the detection of pneumonia in chest X-ray images. My results demonstrate that the use of deep learning models can successfully identify pneumonia with a high degree of accuracy, without imposing significant training and inference costs. Among the models I have experimented with, DenseNet performed the most accurately with 89.42% F1-Score. Although, all of the transfer learning models that I have experimented with in the task, uses ImageNet for pretrained weights - ultimately, the underlying architecture and other hyperparameters impacted the model performances. Thus, despite the presence of more recent and intricate models, for our particular task, DenseNet was the most appropriate architecture. Overall, I was able to leverage the power of transfer learning to optimize my approach for all the models. In conclusion, this study demonstrates the promise of deep learning in medical imaging and highlights the importance of carefully selecting and fine-tuning models for specific tasks

8.1 Future Works

In future research, there is potential to expand the dataset used for training my machine learning models to accommodate a wider range of adult X-ray images, including those depicting various pulmonary diseases beyond the scope of my current study. By doing so, I can increase the robustness and generalizability of my models, as well as enhance their accuracy and efficacy in detecting a broader range of conditions. Future research should also explore the use of larger datasets from diverse populations and regions to improve the generalizability and robustness of the models. Furthermore, it is crucial to focus on the explainability of these models to ensure that they are effectively identifying only the salient features required to accurately detect the target condition. This can help greatly in establishing trust and confidence in the models, as well as ensuring that they are not relying on irrelevant or potentially harmful features. Ultimately, these efforts can lead to more effective and reliable diagnostic tools for healthcare professionals, and improved outcomes for patients.

9 References

Chouhan, V., Singh, S. K., Khamparia, A., Gupta, D., Tiwari, P., Moreira, C., ... & De Albuquerque, V. H. C. (2020). A novel transfer learning based approach for pneumonia detection in chest X-ray images. Applied Sciences, 10(2), 559.

Kallianos, K., Mongan, J., Antani, S., Henry, T., Taylor, A., Abuya, J., & Kohli, M. (2019). How far have we come? Artificial intelligence for chest radiograph interpretation. Clinical radiology, 74(5), 338-345.

Kermany, D. S., Goldbaum, M., Cai, W., Valentim, C. C. S., Liang, H., Baxter, S. L., McKeown, A., Yang, G., Wu, X., Yan, F., Dong, J., Prasadha, M. K., Pei, J., Ting, M. Y. L., Zhu, J., Li, C., Hewett, S., Dong, J., Ziyar, I., Shi, A., … Zhang, K. (2018). Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning. Cell, 172(5), 1122–1131.e9. https://doi.org/10.1016/j.cell.2018.02.010

Kundu, R., Das, R., Geem, Z. W., Han, G. T., & Sarkar, R. (2021). Pneumonia detection in chest X-ray images using an ensemble of deep learning models. PloS one, 16(9), e0256630.

Kundu, R., Das, R., Geem, Z. W., Han, G. T., & Sarkar, R. (2021). Pneumonia detection in chest X-ray images using an ensemble of deep learning models. PloS one, 16(9), e0256630.

Moine, P. (1994). Severe community-acquired pneumonia: etiology, epidemiology, and prognosis factors. Chest 105.5, 1487-1495.

Neuman, M. I., Lee, E. Y., Bixby, S., Diperna, S., Hellinger, J., Markowitz, R., ... & Shah, S. S. (2012). Variability in the interpretation of chest radiographs for the diagnosis of pneumonia in children. Journal of hospital medicine, 7(4), 294-298.

Rajpurkar, P., Irvin, J., Ball, R. L., Zhu, K., Yang, B., Mehta, H., ... & Lungren, M. P. (2018). Deep learning for chest radiograph diagnosis: A retrospective comparison of the CheXNeXt algorithm to practicing radiologists. PLoS medicine, 15(11), e1002686.

Torres, A., & Cillóniz, C. (2015). Clinical Management of Bacterial Pneumonia. Springer International Publishing.

WHO. (2019). WHO Pneumonia. WHO Report. Retrieved April 22, 2023, from https://www.who.int/news-room/fact-sheets/%20detail/pneumonia

Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2017). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2097-2106).

Williams, G. J., Macaskill, P., Kerr, M., Fitzgerald, D. A., Isaacs, D., Codarini, M., ... & Craig, J. C. (2013). Variability and accuracy in interpretation of consolidation on chest radiography for diagnosing pneumonia in children under 5 years of age. Pediatric Pulmonology, 48(12), 1195-1200.

10 Appendix

10.1 Training Code for Modeling:

```
# -*- coding: utf-8 -*-
"""modeling.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1hKCwK8RsXYZncaM4kJNaSskPAZgGOaxj
"""

# Commented out IPython magic to ensure Python compatibility.
import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content/drive/MyDrive/XRAY"
# %cd /content/drive/MyDrive/XRAY
os.getcwd()

!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia
!unzip *.zip
!rm -rf *.zip

!pip install visualkeras
!pip install vit_keras -q
! pip install tensorflow-addons

"""## Import necessary models

"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly
import plotly.express as px
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, plot, iplot
import cv2
from skimage.transform import resize
from skimage.io import imread
import random
import os
import glob
from tqdm.notebook import tqdm
import albumentations as A
import visualkeras

import tensorflow as tf
import keras
from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D, Dense,
Dropout , BatchNormalization, MaxPool2D, GlobalAveragePooling2D
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import Model
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from keras.callbacks import ReduceLROnPlateau
from tensorflow.random import set_seed
from vit_keras import  vit, utils
set_seed(42)
import warnings
warnings.filterwarnings('ignore')
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'

"""## Load Datasets for Traditional ML Models"""

BASE_PATH = "/content/drive/MyDrive/XRAY/chest_xray"
TRAIN_PATH = os.path.join(BASE_PATH, "train")
VAL_PATH = os.path.join(BASE_PATH, "val")
TEST_PATH = os.path.join(BASE_PATH, "test")
def load_data(dir):
    Categories=['NORMAL','PNEUMONIA']
    X_arr=[]#input array
    Y_arr=[] #output array
    for i in Categories:
        print(f'loading... category : {i}')
        path=os.path.join(dir,i)
        for img in os.listdir(path):
            img_array=imread(os.path.join(path,img))
            img_resized=resize(img_array,(100,100,3))
            X_arr.append(img_resized.flatten()/255)
            Y_arr.append(Categories.index(i))
        print(f'loaded category:{i} successfully')
    print("Completed!")
    return X_arr,Y_arr

X_train, y_train = load_data(TRAIN_PATH)
X_val, y_val = load_data(VAL_PATH)
X_test, y_test = load_data(TEST_PATH)

"""## Model Building : Traditional

### SVC Model
"""

from sklearn.svm import SVC
model_svm=SVC(C=0.1,kernel='rbf' )
model_svm.fit(X_train,y_train)
y_predict = model_svm.predict(X_test)
display(pd.DataFrame(classification_report(y_test, y_predict,
output_dict=True)))

"""### Logistic Regression Model"""
```

```
from sklearn.linear_model import LogisticRegression
model_reg = LogisticRegression(solver='liblinear',C=1)
model_reg.fit(X_train,y_train)
y_predict = model_reg.predict(X_test)
display(pd.DataFrame(classification_report(y_test, y_predict,
output_dict=True)))

"""### K-nearest Neighborhood """

from sklearn.neighbors import KNeighborsClassifier
model_knn = KNeighborsClassifier(n_neighbors=5)
model_knn.fit(X_train, y_train)
y_predict = model_knn.predict(X_test)
display(pd.DataFrame(classification_report(y_test, y_predict,
output_dict=True)))

"""### Random Forest """

from sklearn.ensemble import RandomForestClassifier
model_rfc = RandomForestClassifier(max_depth=10,n_estimators=150)
model_rfc.fit(X_train, y_train)
y_predict = model_rfc.predict(X_test)
display(pd.DataFrame(classification_report(y_test, y_predict,
output_dict=True)))

"""## Load datasets for Neural Models"""

BASE_PATH = "/content/drive/MyDrive/XRAY/chest_xray"
TRAIN_PATH = os.path.join(BASE_PATH, "train")
VAL_PATH = os.path.join(BASE_PATH, "val")
TEST_PATH = os.path.join(BASE_PATH, "test")

TRAIN_FILES = glob.glob(TRAIN_PATH+"/**/*.jpeg")
VAL_FILES = glob.glob(VAL_PATH+"/**/*.jpeg")
TEST_FILES = glob.glob(TEST_PATH+"/**/*.jpeg")

print("Number of training files: {}".format(len(TRAIN_FILES)))
print("Number of validation files: {}".format(len(VAL_FILES)))
print("Number of test files: {}".format(len(TEST_FILES)))

splits = ["train", "val", "test"]
all_pneumonia = []
all_normal = []

for split in splits:
    path = os.path.join(BASE_PATH, split)
    norm = glob.glob(os.path.join(path, "NORMAL/*.jpeg"))
    pneu = glob.glob(os.path.join(path, "PNEUMONIA/*.jpeg"))
    all_normal.extend(norm)
    all_pneumonia.extend(pneu)

print("Total Pneumonia Images: {}".format(len(all_pneumonia)))
print("Total Normal Images: {}".format(len(all_normal)))
```

```
normal_img = os.listdir(os.path.join(TRAIN_PATH, "NORMAL"))[0]
sample_img = plt.imread(os.path.join(TRAIN_PATH,"NORMAL", normal_img))
plt.imshow(sample_img, cmap='gray')
plt.colorbar()
plt.title('Raw Chest X Ray Image')

print(f"The dimensions of the image are {sample_img.shape[0]} pixels width and
{sample_img.shape[1]} pixels height, one single color channel.")
print(f"The maximum pixel value is {sample_img.max():.4f} and the minimum is
{sample_img.min():.4f}")
print(f"The mean value of the pixels is {sample_img.mean():.4f} and the
standard deviation is {sample_img.std():.4f}")

"""### Weighting by Class"""

# Class weights

weight_for_0 = len(all_pneumonia) / (len(all_pneumonia) + len(all_normal))
weight_for_1 = len(all_normal) / (len(all_normal) + len(all_pneumonia))

class_weight = {0: weight_for_0, 1: weight_for_1}

print(f"Weight for class 0: {weight_for_0:.2f}")
print(f"Weight for class 1: {weight_for_1:.2f}")

"""### Create Data Generators"""

IMG_SIZE = 224
BATCH = 32
SEED = 42
EPOCH = 20
image_generator = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    samplewise_center=True,
    samplewise_std_normalization=True
)

train = image_generator.flow_from_directory(TRAIN_PATH,
                                            batch_size=8,
                                            shuffle=True,
                                            class_mode='binary',
                                            seed = SEED,
                                            target_size=(IMG_SIZE, IMG_SIZE))

validation = image_generator.flow_from_directory(VAL_PATH,
                                                 batch_size=1,
                                                 shuffle=False,
                                                 class_mode='binary',
                                                 seed = SEED,
                                                        target_size=(IMG_SIZE,
IMG_SIZE))

test = image_generator.flow_from_directory(TEST_PATH,
                                            batch_size=1,
                                            shuffle=False,
                                            class_mode='binary',
                                            seed = SEED,
                                            target_size=(IMG_SIZE, IMG_SIZE))

"""## Model Building : Neural

### Shallow Models : CNN
"""

keras.backend.clear_session()


model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(IMG_SIZE,
IMG_SIZE, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(IMG_SIZE,
IMG_SIZE, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
METRICS = [
    tf.keras.losses.BinaryCrossentropy(name="loss"),
    tf.keras.metrics.Accuracy(name="accuracy"),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
```

```
model.compile(
              optimizer=opt,
              loss='binary_crossentropy',
              run_eagerly=True,
              metrics=METRICS)

"""#### Visualize CNN Model"""

visualkeras.layered_view(model,
to_file='/content/drive/MyDrive/XRAY/assets/layered_model_cnn.png.png').show()
# display using your system viewer
model.summary()

tf.keras.utils.plot_model(
    model,
    to_file='/content/drive/MyDrive/XRAY/assets/model_cnn.png',
    show_shapes=True,
    show_dtype=False,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=False,
    dpi=96,
    layer_range=None,
    show_layer_activations=False,
    show_trainable=False
)

"""#### Start Training"""

ES = tf.keras.callbacks.EarlyStopping(
    monitor = "val_loss",
    patience = 10,
    verbose = 1,
    mode = "max",
)
PT = tf.keras.callbacks.ReduceLROnPlateau(
    monitor = "val_loss",
    patience = 2,
    verbose = 1,
    factor = 0.3,
    min_lr = 0.000000001,
    cooldown = 1
)

trainer_log = model.fit(
    train,
    epochs = EPOCH,
    batch_size = BATCH,
    validation_data = validation,
    callbacks = [PT, ES],
    class_weight = class_weight,
    steps_per_epoch=len(train) // BATCH,
    validation_steps=len(validation),
]

"""#### Plot Training Progress"""

plt.figure(figsize=(8,6))
plt.title('Accuracy scores')
plt.plot(trainer_log.history['accuracy'],'go-')
plt.plot(trainer_log.history['val_accuracy'],'ro-')
plt.legend(['accuracy', 'val_accuracy'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/cnn_accuracy.png")

plt.show()
plt.figure(figsize=(8,6))
plt.title('Loss value')
plt.plot(trainer_log.history['loss'],'go-')
plt.plot(trainer_log.history['val_loss'],'ro-')
plt.legend(['loss', 'val_loss'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/cnn_loss.png")

plt.show()

"""#### Model Evaluation"""

pred = model.predict(test)
print(confusion_matrix(test.classes, pred > 0.5))
display(pd.DataFrame(classification_report(test.classes, pred > 0.5,
output_dict=True)))
# Save the model
model.save("/content/drive/MyDrive/XRAY/assets/best_model_CNN.hdf5")

"""### Transfer Learning : DenseNet"""

from keras.applications.densenet import DenseNet121


keras.backend.clear_session()

densenet_base_model = DenseNet121(include_top=False, weights='imagenet')
x = densenet_base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(1, activation="sigmoid")(x)

model_dense = Model(inputs=densenet_base_model.input, outputs=predictions)

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
METRICS = [
    tf.keras.losses.BinaryCrossentropy(name="loss"),
    tf.keras.metrics.Accuracy(name="accuracy"),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
model_dense.compile(
              optimizer=opt,
              loss='binary_crossentropy',
              run_eagerly=True,
              metrics=METRICS)

# model_dense.summary()

"""#### Visualize DenseNet"""

# print(model_dense.summary())
tf.keras.utils.plot_model(
    model_dense,
    to_file='/content/drive/MyDrive/XRAY/assets/model_dnet.png',
    show_shapes=True,
    show_dtype=False,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=False,
    dpi=96,
    layer_range=None,
    show_layer_activations=False,
    show_trainable=False
)

"""#### Start Training"""

ES = tf.keras.callbacks.EarlyStopping(
    monitor = "val_accuracy",
    patience = 10,
    verbose = 1,
    mode = "max",
)
PT = tf.keras.callbacks.ReduceLROnPlateau(
    monitor = "val_accuracy",
    patience = 2,
    verbose = 1,
    factor = 0.3,
    min_lr = 0.000000001,
    cooldown = 1
)

trainer_dense = model_dense.fit(
    train,
    epochs = EPOCH,
    batch_size = BATCH,
    validation_data = validation,
    callbacks = [PT, ES],
    class_weight = class_weight,
    steps_per_epoch=len(train)//BATCH,
    validation_steps=len(validation),
)

"""#### Plot Training Progress"""
```

```
plt.figure(figsize=(8,6))
plt.title('Accuracy scores')
plt.plot(trainer_dense.history['accuracy'],'go-')
plt.plot(trainer_dense.history['val_accuracy'],'ro-')
plt.legend(['accuracy', 'val_accuracy'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/dnet_accuracy.png")

plt.show()
plt.figure(figsize=(8,6))
plt.title('Loss value')
plt.plot(trainer_dense.history['loss'],'go-')
plt.plot(trainer_dense.history['val_loss'],'ro-')
plt.legend(['loss', 'val_loss'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/dnet_loss.png")

plt.show()

"""#### Model Evaluation

"""

pred = model_dense.predict(test)
print(confusion_matrix(test.classes, pred > 0.5))
display(pd.DataFrame(classification_report(test.classes, pred > 0.5,
output_dict=True)))
# Save the model
model_dense.save("/content/drive/MyDrive/XRAY/assets/best_model_DenseNet.hdf5")

"""### Transfer Learning : VGG16"""

from keras.applications.vgg16 import VGG16
vgg16_base_model =
VGG16(input_shape=(IMG_SIZE,IMG_SIZE,3),include_top=False,weights='imagenet')

model_vgg = tf.keras.Sequential([
        vgg16_base_model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")
    ])

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
METRICS = [
    tf.keras.losses.BinaryCrossentropy(name="loss"),
    tf.keras.metrics.Accuracy(name="accuracy"),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
model_vgg.compile(
              optimizer=opt,
              loss='binary_crossentropy',
              run_eagerly=True,
              metrics=METRICS)

"""#### Visualize VGG Model"""

visualkeras.layered_view(model_vgg,
to_file="/content/drive/MyDrive/XRAY/assets/layered_model_vgg.png").show() #
display using your system viewer
print(model.summary())
tf.keras.utils.plot_model(
    model_vgg,
    to_file='/content/drive/MyDrive/XRAY/assets/model_vgg.png',
    show_shapes=True,
    show_dtype=False,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=False,
    dpi=96,
    layer_range=None,
    show_layer_activations=False,
    show_trainable=False
)

"""#### Start Training"""

ES = tf.keras.callbacks.EarlyStopping(
    monitor = "val_loss",
    patience = 10,
    verbose = 1,
    mode = "max",
)
PT = tf.keras.callbacks.ReduceLROnPlateau(
    monitor = "val_loss",
    patience = 2,
    verbose = 1,
    factor = 0.3,
    min_lr = 0.000000001,
    cooldown = 1
)

trainer_vgg = model_vgg.fit(
    train,
    epochs = EPOCH,
    batch_size = BATCH,
    validation_data = validation,
    callbacks = [PT, ES],
    class_weight = class_weight,
    steps_per_epoch=len(train) // BATCH,
    validation_steps=len(validation),
    validation_freq=1
)

"""#### Plot Training Progress"""

plt.figure(figsize=(8,6))
plt.title('Accuracy scores')
plt.plot(trainer_vgg.history['accuracy'],'go-')
plt.plot(trainer_vgg.history['val_accuracy'],'ro-')
plt.legend(['accuracy', 'val_accuracy'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/vgg_accuracy.png")

plt.show()
plt.figure(figsize=(8,6))
plt.title('Loss value')
plt.plot(trainer_vgg.history['loss'],'go-')
plt.plot(trainer_vgg.history['val_loss'],'ro-')
plt.legend(['loss', 'val_loss'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/vgg_loss.png")

plt.show()

"""#### Model Evaluation"""

pred = model_vgg.predict(test)
print(confusion_matrix(test.classes, pred > 0.5))
display(pd.DataFrame(classification_report(test.classes, pred > 0.5,
output_dict=True)))
# Save the model
model_vgg.save("/content/drive/MyDrive/XRAY/assets/best_model_VGG.hdf5")

"""### Transfer Learning : InceptionNet"""

from keras.applications import InceptionV3
keras.backend.clear_session()

inception_base_model = InceptionV3(
weights="imagenet",
input_shape=(IMG_SIZE,IMG_SIZE,3),
include_top=False)
model_icp =  tf.keras.Sequential([
        inception_base_model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")
    ])


opt = tf.keras.optimizers.Adam(learning_rate=0.001)
METRICS = [
    tf.keras.losses.BinaryCrossentropy(name="loss"),
    tf.keras.metrics.Accuracy(name="accuracy"),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
model_icp.compile(
              optimizer=opt,
              loss='binary_crossentropy',
              run_eagerly=True,
              metrics=METRICS)

"""#### Visualize InceptionNet"""

visualkeras.layered_view(model_icp,
to_file="/content/drive/MyDrive/XRAY/assets/layered_model_icp.png").show() #
display using your system viewer
# print(model_xcp.summary())
tf.keras.utils.plot_model(
    model_icp,
    to_file='/content/drive/MyDrive/XRAY/assets/model_icp.png',
    show_shapes=True,
    show_dtype=False,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=False,
    dpi=96,
    layer_range=None,
    show_layer_activations=True,
    show_trainable=True
)

"""#### Start Training"""

ES = tf.keras.callbacks.EarlyStopping(
    monitor = "val_loss",
    patience = 10,
    verbose = 1,
    mode = "max",
)
PT = tf.keras.callbacks.ReduceLROnPlateau(
    monitor = "val_loss",
    patience = 2,
    verbose = 1,
    factor = 0.3,
    min_lr = 0.000000001,
    cooldown = 1
```

```
)

trainer_icp = model_icp.fit(
    train,
    epochs = EPOCH,
    batch_size = BATCH,
    validation_data = validation,
    callbacks = [PT, ES],
    class_weight = class_weight,
    steps_per_epoch=len(train) // BATCH,
    validation_steps=len(validation),
    validation_freq=1
)

"""#### Plot Training Progress"""

plt.figure(figsize=(8,6))
plt.title('Accuracy scores')
plt.plot(trainer_icp.history['accuracy'],'go-')
plt.plot(trainer_icp.history['val_accuracy'],'ro-')
plt.legend(['accuracy', 'val_accuracy'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/inet_accuracy.png")

plt.show()
plt.figure(figsize=(8,6))
plt.title('Loss value')
plt.plot(trainer_icp.history['loss'],'go-')
plt.plot(trainer_icp.history['val_loss'],'ro-')
plt.legend(['loss', 'val_loss'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/inet_loss.png")

plt.show()

"""#### Model Evaluation"""

pred = model_icp.predict(test)
print(confusion_matrix(test.classes, pred > 0.5))
display(pd.DataFrame(classification_report(test.classes, pred > 0.5,
output_dict=True)))
# Save the model
model_icp.save("/content/drive/MyDrive/XRAY/assets/best_model_inception.hdf5")

"""### Transfer Learning: ResNet"""

from keras.applications import ResNet50

keras.backend.clear_session()
resnet_base_model = ResNet50(input_shape=(IMG_SIZE,IMG_SIZE,3),
include_top=False, weights='imagenet')

model_rnet = tf.keras.Sequential([
        resnet_base_model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")
    ])


opt = tf.keras.optimizers.Adam(learning_rate=0.001)
METRICS = [
    tf.keras.losses.BinaryCrossentropy(name="loss"),
    tf.keras.metrics.Accuracy(name="accuracy"),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
model_rnet.compile(
              optimizer=opt,
              loss='binary_crossentropy',
              run_eagerly=True,
              metrics=METRICS)

"""#### Visualize ResNet Model"""

visualkeras.layered_view(model_rnet,
to_file="/content/drive/MyDrive/XRAY/assets/layered_model_rnet.png").show() #
display using your system viewer
# print(model_xcp.summary())
tf.keras.utils.plot_model(
    model_rnet,
    to_file='/content/drive/MyDrive/XRAY/assets/model_resnet.png',
    show_shapes=True,
    show_dtype=False,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=False,
    dpi=96,
    layer_range=None,
    show_layer_activations=True,
    show_trainable=True
)

"""#### Start Training"""

ES = tf.keras.callbacks.EarlyStopping(
    monitor = "val_loss",
    patience = 10,
    verbose = 1,
    mode = "max",
)
PT = tf.keras.callbacks.ReduceLROnPlateau(
    monitor = "val_loss",
    patience = 2,
    verbose = 1,
    factor = 0.3,
    min_lr = 0.000000001,
    cooldown = 1
)

trainer_rnet = model_rnet.fit(
    train,
    epochs = EPOCH,
    batch_size = BATCH,
    validation_data = validation,
    callbacks = [PT, ES],
    class_weight = class_weight,
    steps_per_epoch=len(train) // BATCH,
    validation_steps=len(validation),
    validation_freq=1
)

"""#### Plot Training Progress"""

plt.figure(figsize=(8,6))
plt.title('Accuracy scores')
plt.plot(trainer_rnet.history['accuracy'],'go-')
plt.plot(trainer_rnet.history['val_accuracy'],'ro-')
plt.legend(['accuracy', 'val_accuracy'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/rnet_accuracy.png")

plt.show()
plt.figure(figsize=(8,6))
plt.title('Loss value')
plt.plot(trainer_rnet.history['loss'],'go-')
plt.plot(trainer_rnet.history['val_loss'],'ro-')
plt.legend(['loss', 'val_loss'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/rnet_loss.png")

plt.show()

"""#### Model Evaluation"""

pred = model_rnet.predict(test)
print(confusion_matrix(test.classes, pred > 0.5))
display(pd.DataFrame(classification_report(test.classes, pred > 0.5 ,
output_dict=True)))
# Save the model
model_rnet.save("/content/drive/MyDrive/XRAY/assets/best_model_resnet.hdf5")

"""### Transfer Learning: ViT"""

keras.backend.clear_session()
```

```
model_vit = vit.vit_b16(
    image_size = IMG_SIZE,
    pretrained = True,
    include_top = True,
    pretrained_top = False,
    classes = 1
)

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
METRICS = [
    tf.keras.losses.BinaryCrossentropy(name="loss"),
    tf.keras.metrics.Accuracy(name="accuracy"),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
model_vit.compile(
              optimizer=opt,
              loss='binary_crossentropy',
              run_eagerly=True,
              metrics=METRICS)

"""#### Visualize ViT Model"""

# print(model_xcp.summary())
tf.keras.utils.plot_model(
    model_vit,
    to_file='/content/drive/MyDrive/XRAY/assets/model_vit.png',
    show_shapes=True,
    show_dtype=False,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=False,
    dpi=96,
    layer_range=None,
    show_layer_activations=True,
    show_trainable=True
)

"""#### Start Training"""

ES = tf.keras.callbacks.EarlyStopping(
    monitor = "val_loss",
    patience = 10,
    verbose = 1,
    mode = "max",
)
PT = tf.keras.callbacks.ReduceLROnPlateau(
    monitor = "val_loss",
    patience = 2,
    verbose = 1,
    factor = 0.3,
    min_lr = 0.000000001,
    cooldown = 1
)

trainer_vit = model_vit.fit(
    train,
    epochs = EPOCH,
    batch_size = BATCH,
    validation_data = validation,
    callbacks = [PT, ES],
    class_weight = class_weight,
    steps_per_epoch=len(train) // BATCH,
    validation_steps=len(validation),
    validation_freq=1
)

"""#### Plot Training Progress"""

plt.figure(figsize=(8,6))
plt.title('Accuracy scores')
plt.plot(trainer_vit.history['accuracy'],'go-')
plt.plot(trainer_vit.history['val_accuracy'],'ro-')
plt.legend(['accuracy', 'val_accuracy'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/vit_accuracy.png")

plt.show()
plt.figure(figsize=(8,6))
plt.title('Loss value')
plt.plot(trainer_vit.history['loss'],'go-')
plt.plot(trainer_vit.history['val_loss'],'ro-')
plt.legend(['loss', 'val_loss'])
plt.savefig("/content/drive/MyDrive/XRAY/assets/vit_loss.png")

plt.show()

"""#### Model Evaluation"""

pred = model_vit.predict(test)
print(confusion_matrix(test.classes, pred > 0.5))
display(pd.DataFrame(classification_report(test.classes, pred > 0.5 ,
output_dict=True)))
# Save the model
model_vit.save("/content/drive/MyDrive/XRAY/assets/best_model_vit.hdf5")
```