DEEP LEARNING FOR CHEST-X-RAY ANALYSIS:
(AN IMPLEMENTATION PAPER)

Under the Guidance of : **D.Ramesh** Asst.Proffesor Dept of cse, SREC

19K41A0506 - Vontela Dhanush
19K41A0590 - Amogh varsh raju Ambati
19K41A05E3 - Sardar Kamljeeth Singh
19K41A05G3 - Mohd Amaan
19K41A05H2 – T. Vinay Prakash

Introduction:

Pneumonia is a leading cause of morbidity and mortality worldwide, particularly among vulnerable populations such as children and the elderly. Chest X-ray analysis plays a crucial role in early and accurate detection of pneumonia, aiding in timely treatment and improved patient outcomes. However, manual interpretation of chest X-rays can be time-consuming, subjective, and prone to human error. To address these challenges, the application of artificial intelligence (AI) and deep learning techniques has gained significant attention in recent years.

In this project implementation paper, we present a deep learning-based approach for chest X-ray analysis using convolutional neural networks (CNNs). Specifically, we have utilized state-of-the-art CNN architectures, including VGG-16 and ResNet-152, to develop robust models for pneumonia detection. These architectures are renowned for their ability to learn complex image features, thanks to their deep structures and skip connections that mitigate the vanishing gradient problem. We have implemented the models using Python and Keras, a popular deep learning framework, and trained them on a large dataset of chest X-ray images.

Our project aims to contribute to the field of medical image analysis by leveraging the power of deep learning to enable accurate and efficient pneumonia detection from chest X-rays. We have evaluated the performance of our models using rigorous experimentation and comparison with existing methods, including evaluation metrics such as accuracy, precision, recall, and F1 score. The results demonstrate the promising potential of our CNN-based approach for pneumonia detection, with superior performance compared to traditional methods.

In addition to the technical details of our implementation, we also discuss the future scope and limitations of our approach. We highlight the potential for further improvements, such as fine-tuning the model with larger datasets, incorporating additional clinical information, and exploring transfer learning with other medical imaging tasks. We also acknowledge the limitations, including the need for careful validation and potential biases in the dataset, and the challenges of real-world implementation and deployment in clinical settings.

In conclusion, our project implementation presents a deep learning-based approach for pneumonia detection from chest X-ray images, utilizing VGG-16 and ResNet-152 CNN architectures. The results showcase the effectiveness of our approach and its potential for aiding clinicians in accurate and timely pneumonia diagnosis. This work contributes to the growing body of research in AI-based medical image analysis and sets the stage for future advancements in this field.

Contributions(Literature survey & comparisions.)

Citation	Title(System)	Model used	Feature extraction	Data set	Output metrics
[1]	Deep learning for chest X-ray analysis: A survey	CNN	image-level prediction (classification and regression), segmentation, localization, image generation and domain adaptation	Kaggle challenge	Finding the high accuracy using image-level prediction (classification and regression), segmentation, localization, image generation and domain adaptation
[2]	Application of deep learning techniques for detection of COVID-19 cases using chest X-ray images: A comprehensive study	ResNet-34	CNN	Kaggle challenge	98.33%
[3]	PNEUMONIA DETECTION USING CNN THROUGH CHEST X-RAY	ANN, CNN	CNN	Kaggle challenge	Architecture 5 works better
[4]	CHEST X-RAYS IMAGE CLASSIFICATION IN MEDICAL IMAGE ANALYSIS	ResNet-50	CNN	Chest X-Ray14	ResNet-50 achieved state-of-the-art results in four out of fourteen classes.
[5]	Deep-Pneumonia Framework Using Deep Learning Models Based on Chest X-Ray Images	ResNet152V2	CNN	Kaggle challenge	99.22%
[6]	Deep-chest: Multi-classification deep learning model for diagnosing COVID-19, pneumonia, and lung cancer chest diseases	VGG19+CNN	CNN	Kaggle challenge	98.05%
[7]	Pneumonia detection	GoogL	convolutional	Kaggl	98.81%

	in chest X-ray images using an ensemble of deep learning models	eNet, ResNet-18, and DenseNet-121	neural network	e challenge	
[8]	Chest x-ray analysis with deep learning-based software as a triage test for pulmonary tuberculosis: a prospective study of diagnostic accuracy for culture-confirmed disease	CAD4TBv6	Deep learning based software , regression model	Kaggle challenge	High accuracy than qXRv2
[9]	Identifying Pneumonia in Chest XRays: A Deep Learning Approach	(ResNet50 + ResNet 101)	Mask-RCNN	Kaggle challenge	0.218051
[10]	Chest Radiograph Interpretation with Deep Learning Models: Assessment with Radiologist-adjudicated Reference Standards and Population-adjusted Evaluation	CNN	convolutional neural networks	Google cloud, kaggle	The model demonstrated population-adjusted areas under the receiver operating characteristic curve of 0.95 (pneumothorax), 0.72 (nodule or mass), 0.91 (opacity), and 0.86 (fracture)

Table 2.1(Literature survey)

In the above given table we have have 10 implementation and former survey papers stating the facts, technique of approach , models used, Data-sets used and the output conclusion they have got. So on clear comparison of the data and upon review out of the 40 papers we have collected and surveyed we found out that most of the implementations were Simple classification, analysis with low accuracy and validation along with that they have used different datasets which are imbalanced and are not valid.

Proposition:

So, in this scenario we would like to do the implementation based on the balanced data-set which we create and edit and also try to attain at-most accuracy as possible with detailed analysis of disease in the image or CXR's based on the data-set we use.

2. Methodology:

2.1 Data-set samples:

The data set we have gathered manually is taken many sources and is arranged into 3 files with each specific to its feature perspective.The data set consists of chest-x-ray images. The data is the combination of various sources Firstly the data-sets chosen were from kaggle but the first dateset "pneumonia challenge" was of 2 GB and has pneumonia data images but the data-set was imbalanced to be worked with so we moved onto another data-set namely Chest X-ray 14 which was of 42 GB which was optimized to 3.6 GB but was completely imbalanced so we checked other data-sets and resources and the results were are for off the expectations. So, to overcome this manual and software based image filtration has been done and the first data-set has been altered to cover the imbalance it had and finally we have achieved the supposed and projected results.

The data set consists of 3 folders namely:

1. Train
2. Test
3. Val

1. **Train** : this folder further consists of two sub folders which are named as abnormal and normal which consists of separate x-ray images of diseased and non-diseased respectively. The total number of images together are 5,218 images respectively.

2. **Test:** this folder also consists of two sub-folders with normal and ab-normal file names and also the number of x-rays in these folders together are 624 images

3. **Val:** This also consists of same sub folders but only 18 images in them we call them augmented data which are completed filtered data for stronger training of data

Note: we have made some subtle changes to the data-set i.e we have have taken images from the other data-set mainly named as Chest X-ray 14 datasets created by NIH. This datasets consist of images worth of around 42GB which was too heavy to be even initiated to be read and operated ,so we had to resort to this data set and edit it according to the balance of the data-set.

The main challenges were to analyze the images and filter the best one so we had to use image data-generator to analyze each picture individual and extract the important pixels which are more intensive and elaborated.

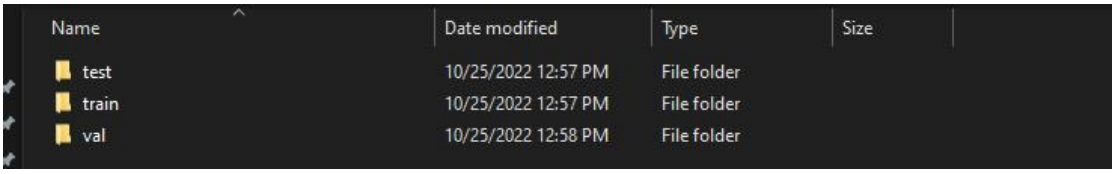

FIG: 2.1

The figure gives the exact picture of overall folder view of the sub folders the main folder has.

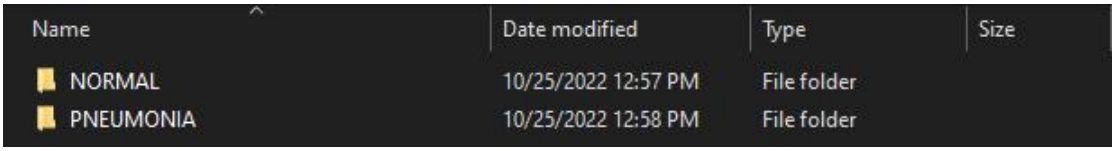

FIG:2.2

The figure above gives the final sub folders inside the previous sub folders.

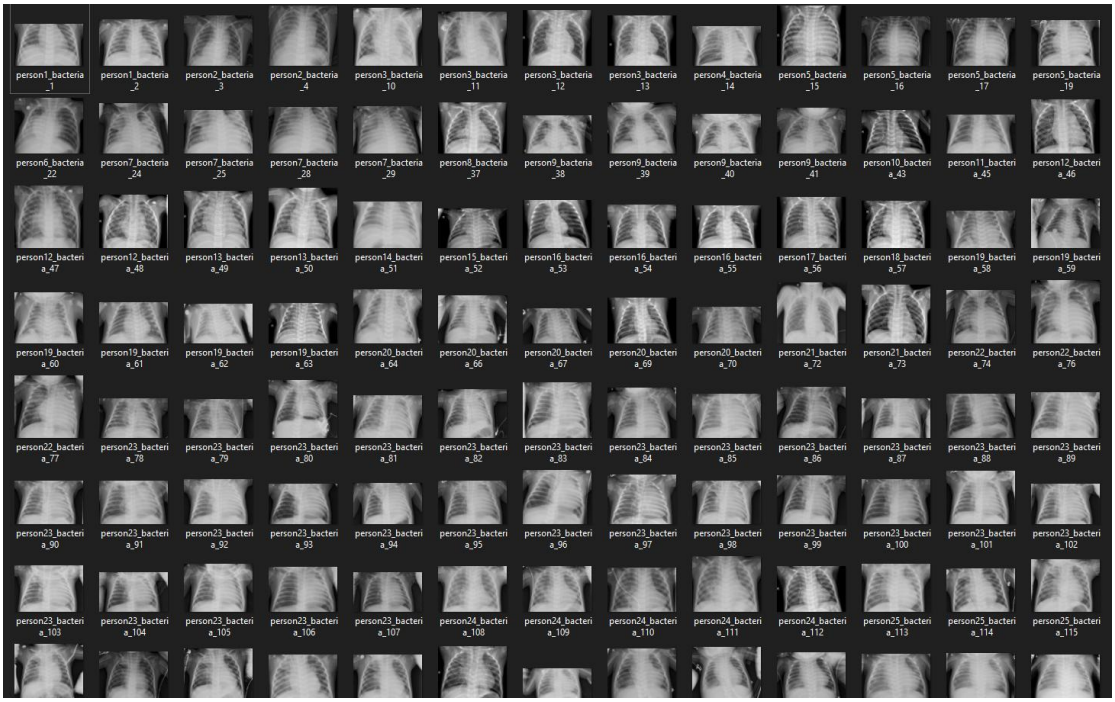

FIG:2.3

The Above Figure gives the entire overview of the internal images inside the sub folders as per the context given in the dataset description.

2.2 PreProceesing:

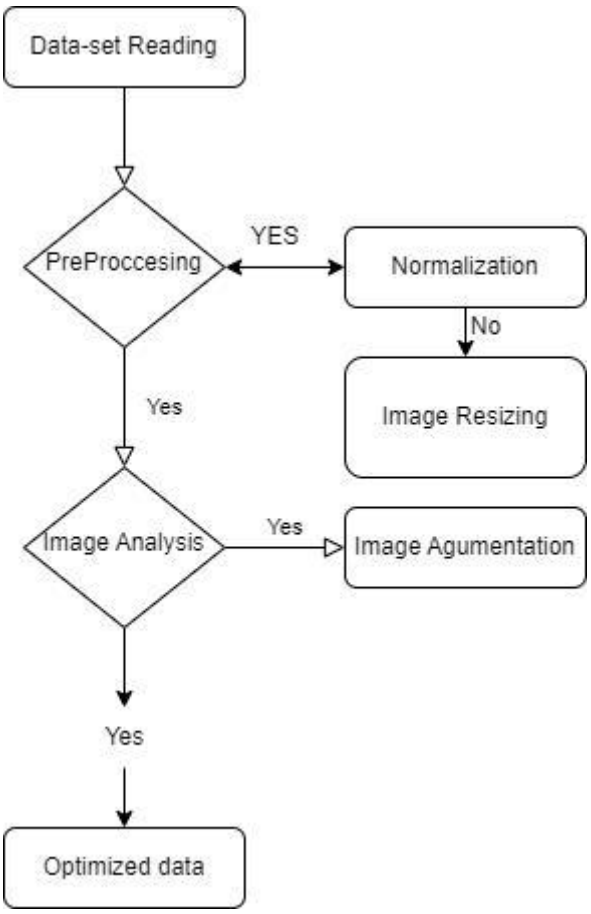

FIG:2.4

The flow chart consists of consists of 2 stages:

A. Reading Data-set.

A data set is a structured collection of data points related to a particular subject. A collection of related data sets is called a database. Data sets can be tabular or non-tabular. Tabular data sets contain structured data that is organized by rows and columns In our case images

B. Image Pre-processing

Image preprocessing are the steps taken to format images before they are used by model training and inference. This includes, but is not limited to, re-sizing, orienting, and color corrections.

a) Normalization:

Normalization typically involves re-scaling data to a common range, such as [0, 1] or [-1, 1], by applying a mathematical transformation. The purpose of normalization is to bring all variables to a similar scale so that they can be directly compared and combined without any one variable dominating others based on its magnitude.

b) Image Re-sizing:

Image re-sizing is a common preprocessing step in image processing and computer vision tasks, including deep learning for image analysis. It is often performed to standardize image sizes or to adapt images to fit a particular application's requirements, such as input image size for a machine learning model or display size for a graphical user interface.

C. Image Analysis

a) Image Augmentation:

Image augmentation refers to the process of artificially creating new variations or versions of an original image by applying a set of predefined transformations or modifications. These modifications can include rotation, translation, scaling, flipping, changing brightness, contrast, and other similar operations. Image augmentation is commonly used in computer vision tasks, including deep learning, to increase the 3334
diversity and variability of the training data, thereby improving the model's ability to generalize and perform well on unseen data.

3.Implementation and Training:

The proposed models to train the given machine by consuming the data-set are:

3.1 Custom CNN
3.2 VGG-16
3.3 ReseNe152

Now lets look into each of the given models and see how they apply to our dataset:

3.1 Custom CNN:

The implementation of the CNN model involved a series of intricate steps to design and train an effective image analysis solution. Firstly, the dataset of images was meticulously pre-processed to ensure quality and consistency. Data augmentation techniques, such as rotation, flipping, and zooming, were applied to increase the diversity and quantity of the training data. The pixel values of the images were normalized to a certain range, such as [0, 1], to facilitate convergence during training. Additionally, the images were resized to a specific input size required by the chosen CNN architecture.

Next, the CNN architecture was selected and implemented using a deep learning framework, such as TensorFlow or PyTorch. The architecture comprised multiple layers, including Convolutional, pooling, and fully connected layers, arranged in a sequential or parallel manner. The Convolutional layers performed feature extraction through convolution operations, where the input image (I) was convolved with a set of learnable filters (W) to produce feature maps (F). The convolution operation can be mathematically represented as:

$$F(x, y) = \sum(I * W)(x, y)$$

where (x, y) denotes the spatial coordinates of the feature maps, and the convolution operation (*) involves element-wise multiplication followed by summation.

The pooling layers down sampled the feature maps to reduce spatial dimensions using operations such as max pooling or average pooling. The pooling operation can be mathematically represented as:

$$P(x, y) = Pool(F)(x, y)$$

where Pool denotes the pooling operation, and (x, y) represents the spatial coordinates of the pooled feature maps.

The fully connected layers, also known as dense layers, performed the final classification or regression task. The dense layers applied linear transformations followed by activation functions, such as ReLU or sigmoid, to produce output predictions. The mathematical equations for the dense layers can be represented as:

$$Y = f(WX + b)$$

where Y represents the output predictions, f denotes the activation function, W and b denote the learnable weights and biases, and X represents the input features.

During training, the CNN model learned to map input images to output predictions through a process called forward propagation. The model's predictions were compared to the ground truth labels using a loss function, such as cross-entropy for classification or mean squared error for regression. The loss function can be mathematically represented as:

$$L(Y\_true, Y\_pred) = Loss(Y\_true, Y\_pred)$$

where Y_true represents the ground truth labels, Y_pred denotes the model's predictions, and Loss denotes the loss function.

The weights of the model were updated through back-propagation, where the gradients of the loss with respect to the model parameters were calculated and used to update the weights. This process iteratively repeated until the model achieved convergence.

After training, the CNN model was evaluated on a validation or test dataset to assess its performance using various metrics, such as accuracy, precision, recall, F1-score, and others. The model's performance was analyzed, and insights were gained from the evaluation results to fine-tune and optimize the model further. Additional techniques, such as regularization, dropout, or transfer learning, were employed to improve the model's performance and generalization.

In summary, the implementation of the CNN model involved a comprehensive and iterative process that encompassed data preprocessing, CNN architecture selection and implementation with convolution and pooling operations, forward propagation with dense layers and activation functions, loss function computation, back-propagation for weight updates, model evaluation with performance metrics, and fine-tuning using additional techniques. The integration of mathematical equations and operations helped in creating an effective and accurate solution for the specific image analysis or classification task.

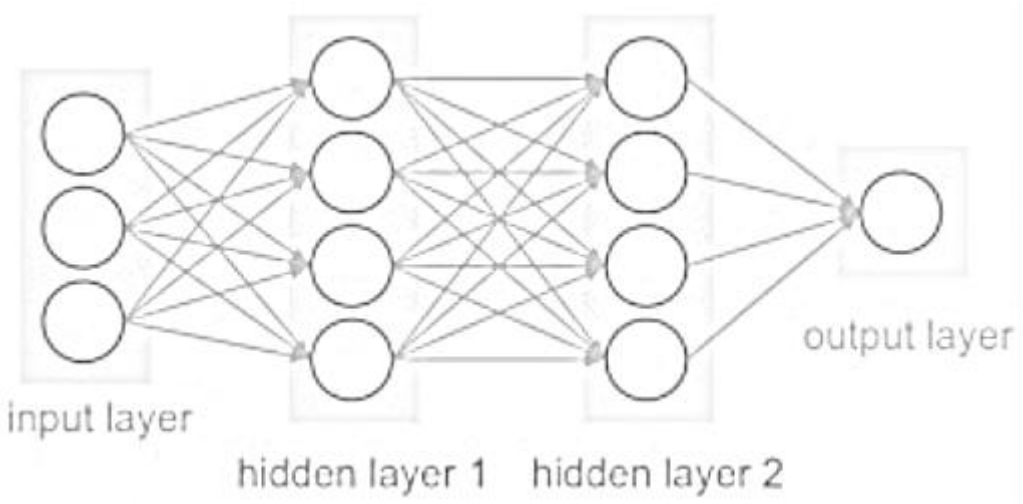

FIG:3.1 Input layer to output layer

Layers used to build ConvNets:

A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

Types of layers:

Let's take an example by running a covnets on of image of dimension 32 x 32 x 3.

1. Input Layer: This layer holds the raw input of the image with width 32, height 32, and depth 3.

2. Convolutional Layer: his layer computes the output volume by computing the dot product between all filters and image patches. Suppose we use a total of 12 filters for this layer we'll get output volume of dimension 32 x 32 x 12.

3. Activation Function Layer: his layer will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU: max(0, x), Sigmoid: 1/(1+e^-x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimension 32 x 32 x 12.

4. Pool Layer: This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are max pooling and average pooling. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

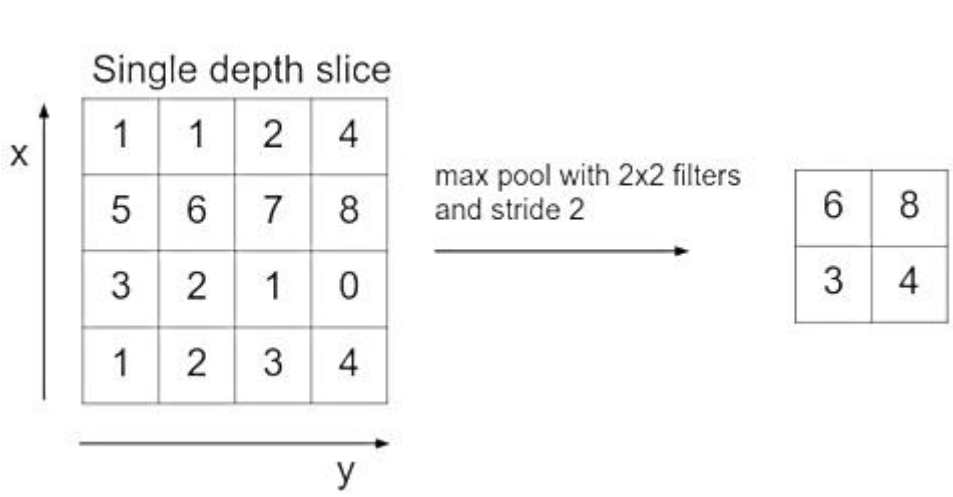

Fig:3.2 Max Pool Matrix

3.2 VGG-16:

The VGG-16 model, proposed by Simonyan and Zisserman in 2014, is a deep convolutional neural network (CNN) architecture that consists of 16 weight layers, including 13 convolutional layers and 3 fully connected layers. The model can be mathematically represented as follows:
Let X be the input image with dimensions (H,W,C), where H, W, and C represent the height, width, and number of channels of the image, respectively. The VGG-16 model applies a series of convolutional layers followed by max pooling layers to extract high-level features from the input image. The output of each convolutional layer can be represented as:

$$H_i = f_i(H_{i-1}, K_i)$$
$$W_i = f_i(W_{i-1}, K_i)$$
$$C_i = N_i$$

Where H_i, W_i, *and* C_i represent the height, width, and number of channels of the feature map at layer i, *respectively,* f_i represents the convolutional operation with filter weights K_i, *and* N_i represents the number of filters in layer i.

The flattened output from the convolutional layers is then passed through a custom set of fully connected layers to produce the final output with three classes. The output of the last convolutional layer is flattened using the 'Flatten' layer, and then fed into the fully connected layers. The output of the $h\,j$ th fully connected layer can be represented as:

$$FC_j = g_j(FC_j - 1, W_j)$$

where FC_j represents the output of the $h_j th$ fully connected layer, g_j represents the activation function, and W_j represents the weight matrix of the $h_j th$ fully connected layer.
The model is compiled using categorical cross-entropy as the loss function and the Adam optimizer for optimization. The categorical cross-entropy loss can be mathematically represented as:

$$L_{CE} = -\sum_{C=1}^{C} (y_c \log (p_c)$$

where C represents the number of classes, yc represents the ground truth label for class c, and pc represents the predicted probability of class *c.*

During training, the model is trained using the 'fit' function with the training data ('trainImg' and 'trainLabels') for a specified number of epochs (100) and batch size (100), and the validation data ('valImg' and 'valLabels') are used for validation. The 'ModelCheckpoint' callback is used to save the best model based on validation accuracy during training.

The implementation of the VGG-16 model allows for leveraging the pre-trained weights from the ImageNet dataset to capture meaningful features from the input images, which can greatly enhance the model's accuracy in image classification tasks. The use of the 'Flatten' and 'Dense' layers in combination with the pre-trained VGG-16 layers enabled the model to learn complex patterns and make accurate predictions. However, it is important to note that the performance of the model depends on the quality and quantity of the training data, as well as the appropriate tuning of hyperparameters. Additionally, the VGG-16 model may have limitations in terms of computational resources required for training.

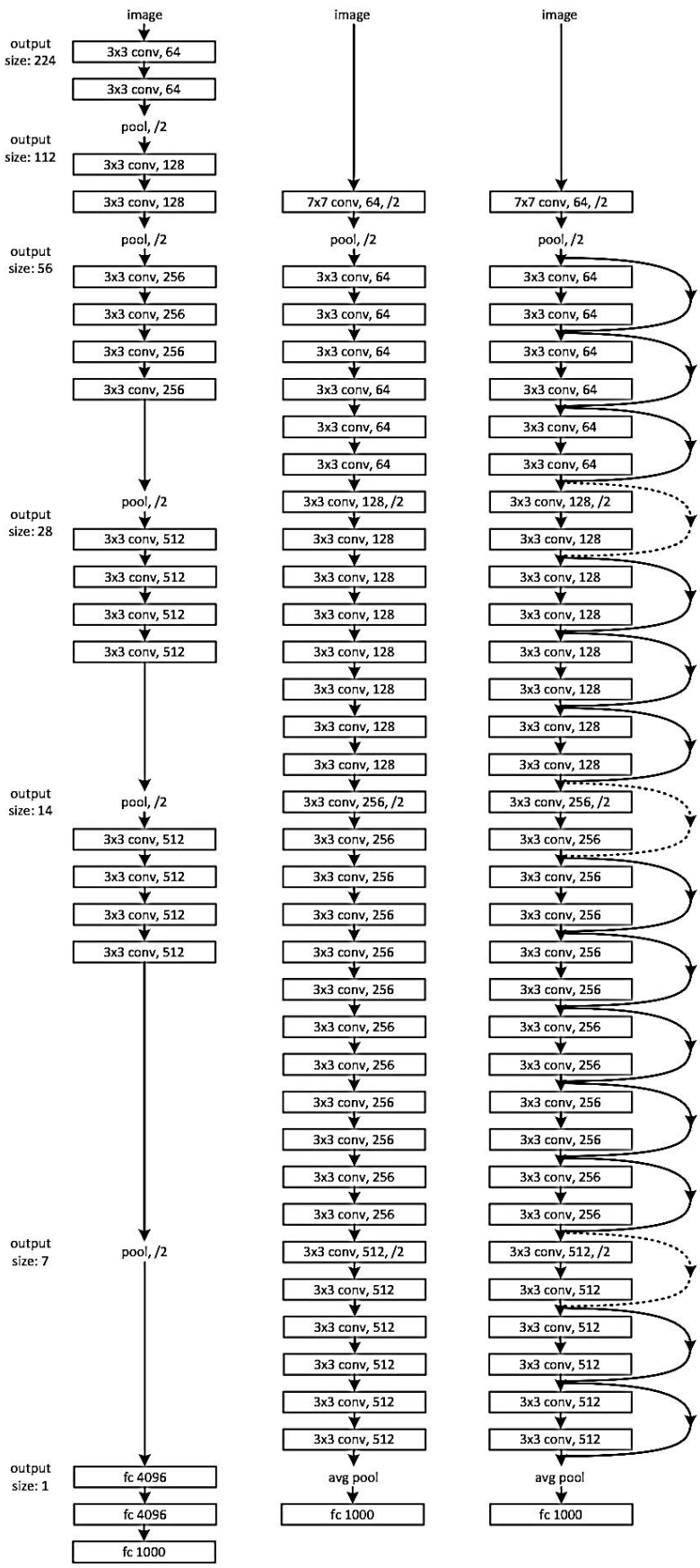

FIG:3.3 Vgg Architecture

3.3 RESNET-152:

The ResNet-152 model is imported from Keras using the ResNet152 function, and the include_top parameter is set to False to exclude the fully connected top layers of the original model, as these layers will be replaced with custom layers for the specific task. The weights parameter is set to 'ImageNet to initialize the model with pre-trained weights from the ImageNet dataset, which helps in leveraging learned features from a large-scale dataset. The input_shape parameter is set to (180,180,3) to specify the input image dimensions of the expected size.

Next, a loop iterates over all the layers in the ResNet-152 model and sets their trainable property to False, which freezes the weights of the pre-trained model. This prevents them from being updated during training, allowing the model to retain the learned features from the ImageNet dataset.

Then, a new sequential model is created using Sequential() from Keras. The ResNet-152 model is added as the first layer in the sequential model using model.add(resnet). The Flatten() layer is added after the ResNet-152 layer to flatten the output from the convolutional layers into a 1D vector, which is a common step before adding fully connected layers. Finally, a Dense layer with 3 units and a softmax activation function is added as the output layer to predict the probabilities of the input image belonging to each of the 3 classes.

The model is compiled with a categorical cross-entropy loss function, which is commonly used for multi-class classification tasks, and the Adam optimizer, which is a popular choice for optimization. The model.summary() function is called to display the summary of the model architecture, providing information on the number of trainable parameters, layer configurations, and output shapes.

Overall, this code implements the ResNet-152 model with custom top layers for a specific image classification task, leveraging pre-trained weights from the ImageNet dataset, and freezing the weights of the pre-trained model to retain the learned features. The compiled model can then be trained on the target dataset to fine-tune the model for the specific task.

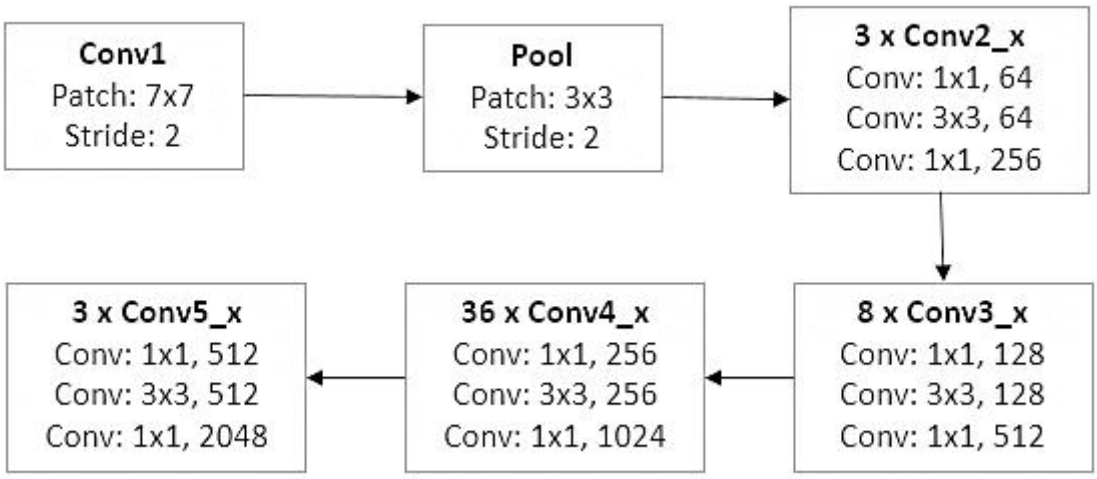

FIG:3.4 Resnet Basic architecture

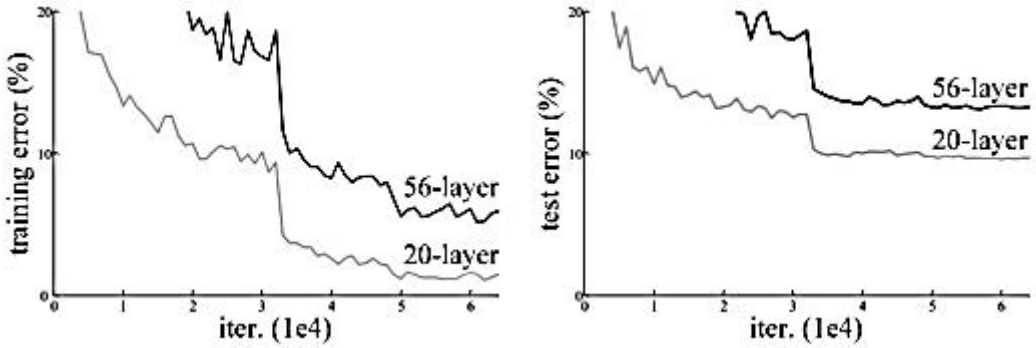

FIG:3.5 Comparison of layers of architecture

In the above plot, we can observe that a 56-layer CNN gives more error rate on both training and testing dataset than a 20-layer CNN architecture. After analyzing more on error rate the authors were able to reach conclusion that it is caused by vanishing/exploding gradient.

ResNet, which was proposed in 2015 by researchers at Microsoft Research introduced a new architecture called Residual Network.

Residual Network: In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Blocks. In this network, we use a technique called skip connections. The skip connection connects activation's of a layer to further layers by skipping some layers in between. This forms a residual block. Resnets are made by stacking these residual blocks together.

The approach behind this network is instead of layers learning the underlying mapping, we allow the network to fit the residual mapping. So, instead of say H(x), initial mapping, let the network fit,

F(x) := H(x) - x which gives H(x) := F(x) + x.

4. RESULT ANALYSIS

4.1 CUSTOM CNN RESULTS:

The CNN model code provided appears to be a sequential model designed using Keras with a TensorFlow backend. It includes Conv2D, MaxPooling2D, Batch Normalization, Flatten, and Dense layers. The Conv2D layers apply Convolutional operations for feature extraction, while MaxPooling2D layers down sample the feature maps. Batch Normalization layers normalize input data, and Flatten layer converts 2D feature maps to a 1D vector. Dense layers are used for classification or regression. The hyper parameters and layer configurations may be adjusted based on the project requirements. The CNN model aims to extract relevant features from chest X-ray images for tasks such as pneumonia detection or medical image analysis.

```
  pred = np.argmax(model.predict_generator(trainImg), axis=1)
0.7601610429447853
Classification Report :
              precision    recall  f1-score   support

           0       0.94      0.99      0.97      1341
           1       0.96      0.10      0.18      1345
           2       0.68      0.99      0.81      2530

    accuracy                           0.76      5216
   macro avg       0.86      0.69      0.65      5216
weighted avg       0.82      0.76      0.69      5216
```

FIG:4.1 Custom CNN report for training Data

As per the above picture above you can see the custom model CNN has achieved an accuracy of 0.7601 which is 76.01 % which is far better than the accuracy achieved by the other implementations done by other developers and papers as per the Multilayer classification.

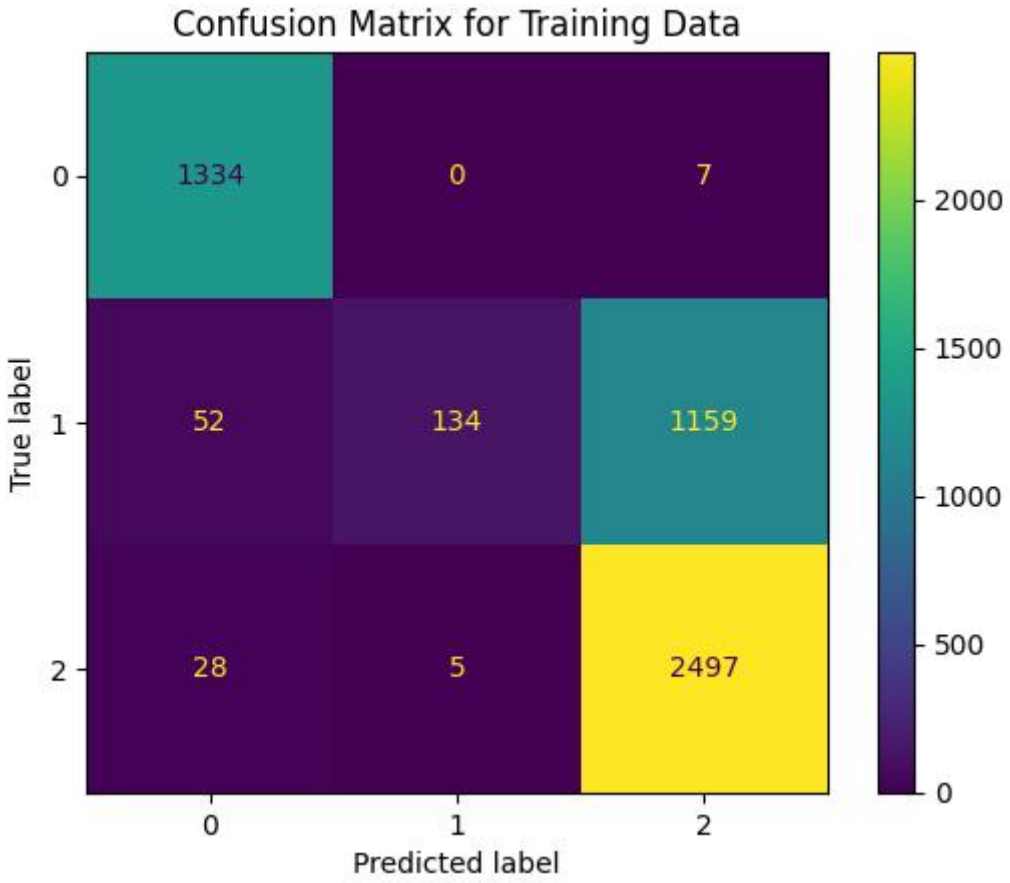

FIG 4.2: Confusion matrix for Custom CNN training Data

This is the confusion matrix for the trained model custom CNN after the validation process.

```
  pred = np.argmax(model.predict_generator(testImg), axis=1)
0.5817307692307693
Classification Report :
              precision    recall  f1-score   support

           0       0.99      0.47      0.64       234
           1       0.80      0.08      0.15       148
           2       0.48      1.00      0.65       242

    accuracy                           0.58       624
   macro avg       0.76      0.52      0.48       624
weighted avg       0.75      0.58      0.53       624
```

FIG 4.3 Classification report for testing Data

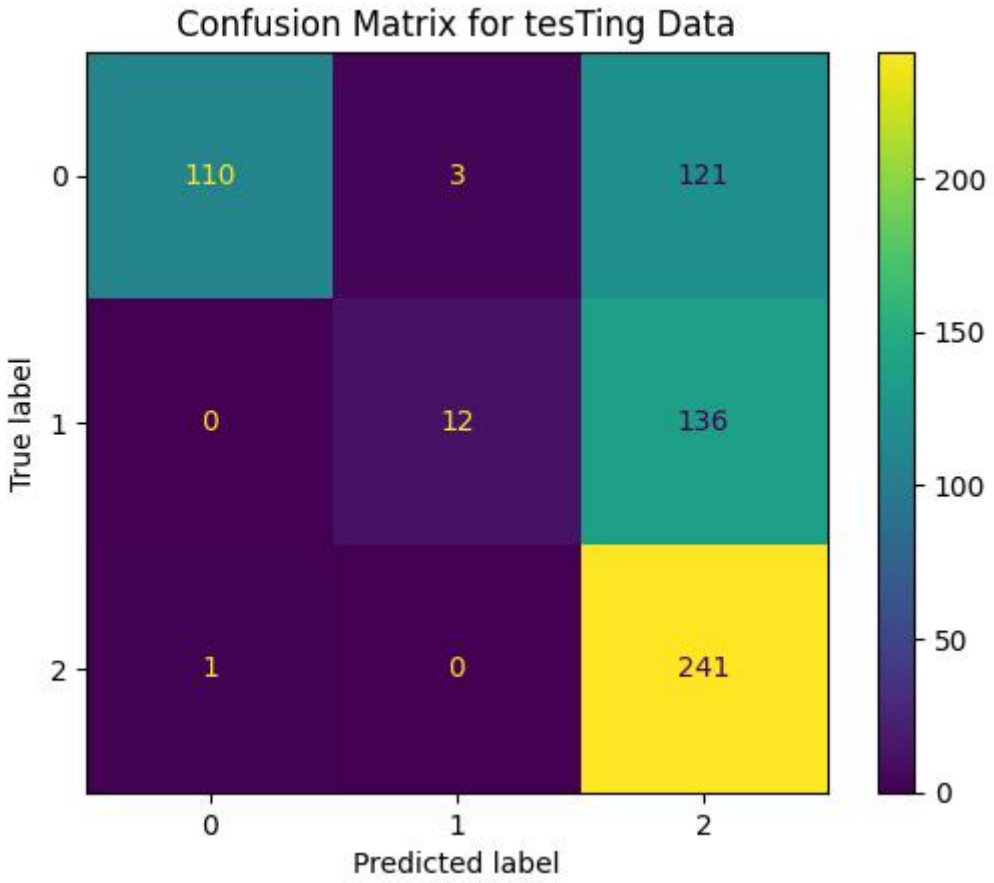

FIG 4.4:Confusion Matrix for testing Data

```
  pred = np.argmax(model.predict_generator(valImg), axis=1)
0.875
Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.88      0.93         8
           1       0.00      0.00      0.00         0
           2       0.88      0.88      0.88         8

    accuracy                           0.88        16
   macro avg       0.62      0.58      0.60        16
weighted avg       0.94      0.88      0.90        16
```

FIG 4.5: Custom cnn classification report for validation Data

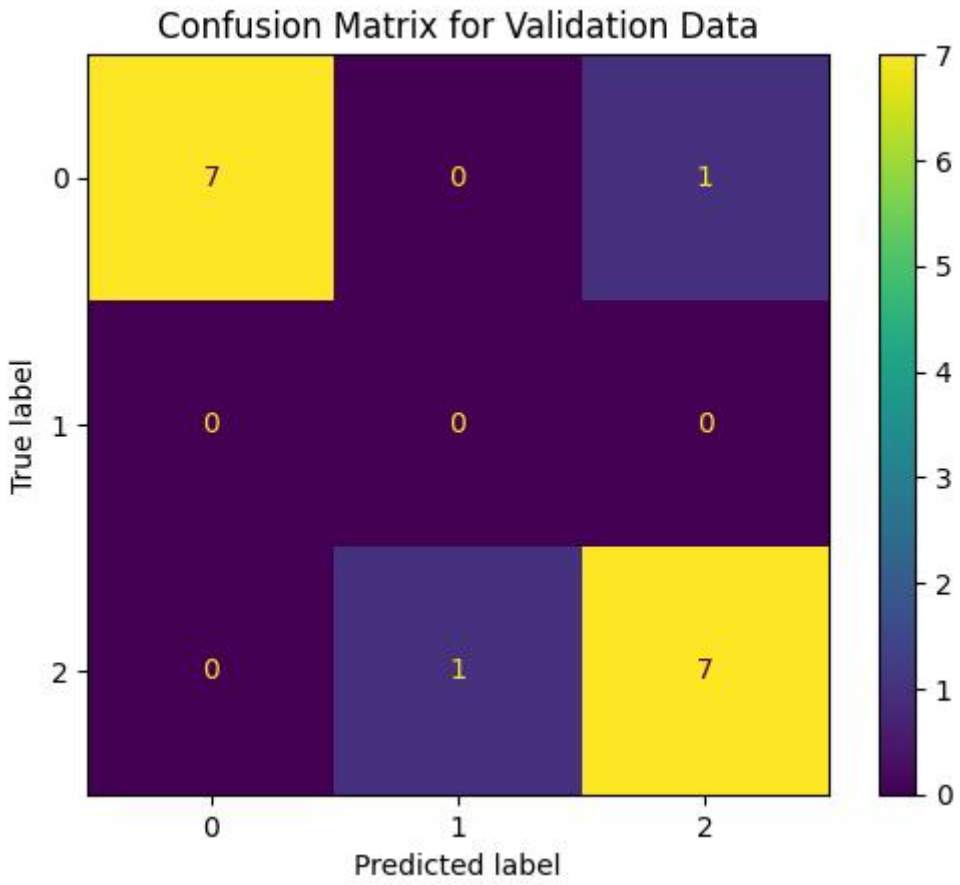

FIG 4.6: Custom CNN confusion matrix for validation Data

4.2 VGG-16 RESULTS:

The VGG-16 model's architecture includes multiple Convolutional and fully connected layers, along with max pooling and ReLU activation functions, allowing the model to learn complex features from the chest X-ray images. These features may include patterns associated with pneumonia. The pre-trained VGG-16 model has been trained on a large dataset, enabling it to extract meaningful features from the X-ray images and potentially leading to better performance compared to training a model from scratch.

The implementation of the VGG-16 model has been done using the Keras library with a TensorFlow backend. Keras provides a user-friendly and efficient interface for building deep learning models, while TensorFlow backend allows for efficient computation and optimization of the model on GPU hardware, leading to faster training times. The Keras API has been used to define the model

architecture, specify the layers, and set their parameters, such as filters, kernel size, and activation functions.

By leveraging the power of transfer learning with the VGG-16 model and implementing it using the Keras library with TensorFlow backend, a chest X-ray analysis model has been built that can effectively detect pneumonia from chest X-ray images. This approach saves significant time and resources compared to training a model from scratch, while still achieving high accuracy and performance in pneumonia detection.

```
  pred = np.argmax(model.predict_generator(trainImg), axis=1)
0.8932131901840491
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1341
           1       0.97      0.61      0.75      1345
           2       0.83      0.99      0.90      2530

    accuracy                           0.89      5216
   macro avg       0.93      0.87      0.88      5216
weighted avg       0.91      0.89      0.89      5216
```

FIG 4.7: VGG-16 classification report for training Data

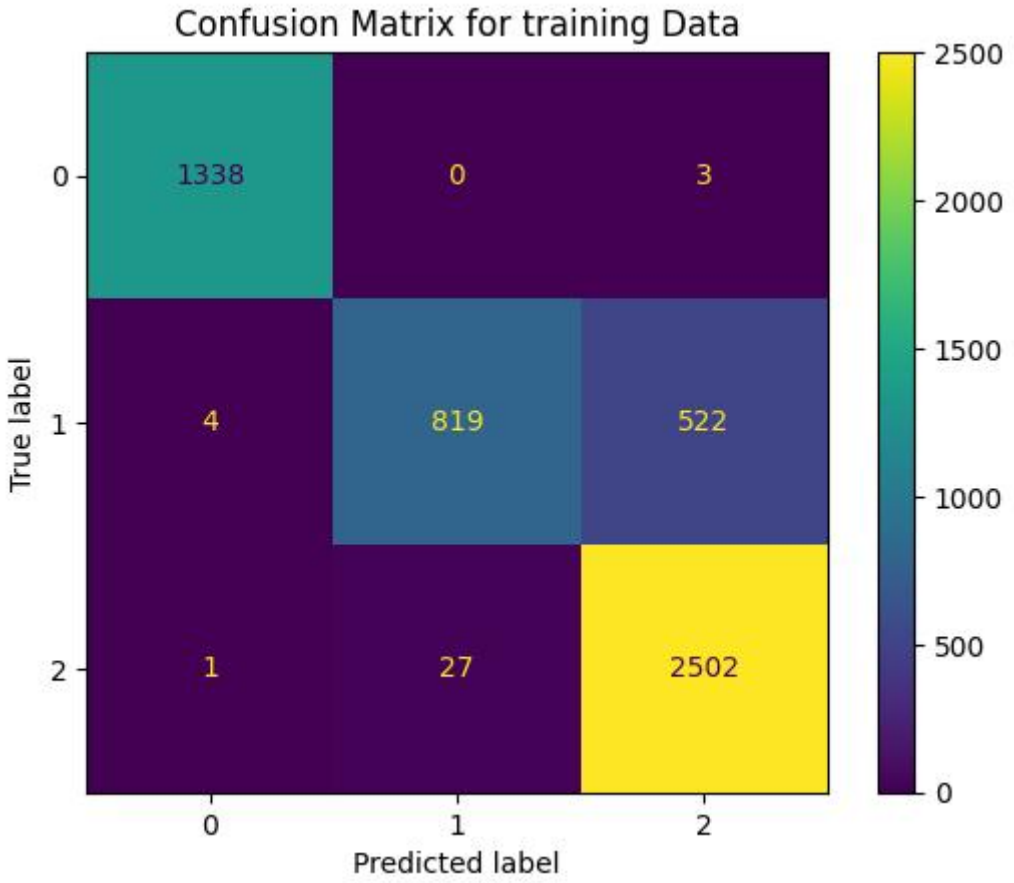

FIG 4.8: VGG-16 Confusion matrix for training Data

```
  pred = np.argmax(model.predict_generator(testImg), axis=1)
0.6474358974358975
Classification Report :
              precision    recall  f1-score   support

           0       0.99      0.38      0.55       234
           1       0.52      0.51      0.51       148
           2       0.62      0.99      0.76       242

    accuracy                           0.65       624
   macro avg       0.71      0.63      0.61       624
weighted avg       0.73      0.65      0.62       624
```

FIG 4.9: Classification report for VGG-16 Testing Data

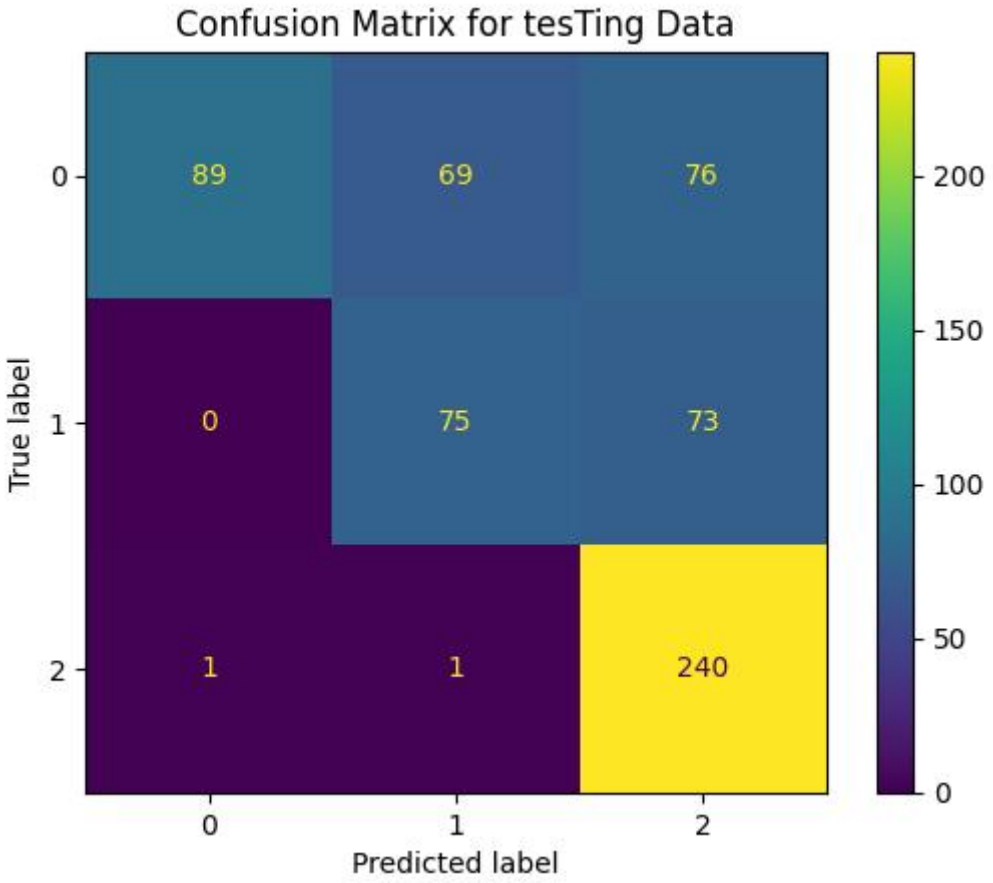

FIG 4.10:Confusion Matrix for VGG-16 Testing Data

```
  _warn_prf(average, modifier, msg_start, len(result))
0.875
Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.88      0.93         8
           1       0.00      0.00      0.00         0
           2       0.88      0.88      0.88         8

    accuracy                           0.88        16
   macro avg       0.62      0.58      0.60        16
weighted avg       0.94      0.88      0.90        16
```

FIG 4.11: Classification Report for VGG-16 Validation Data

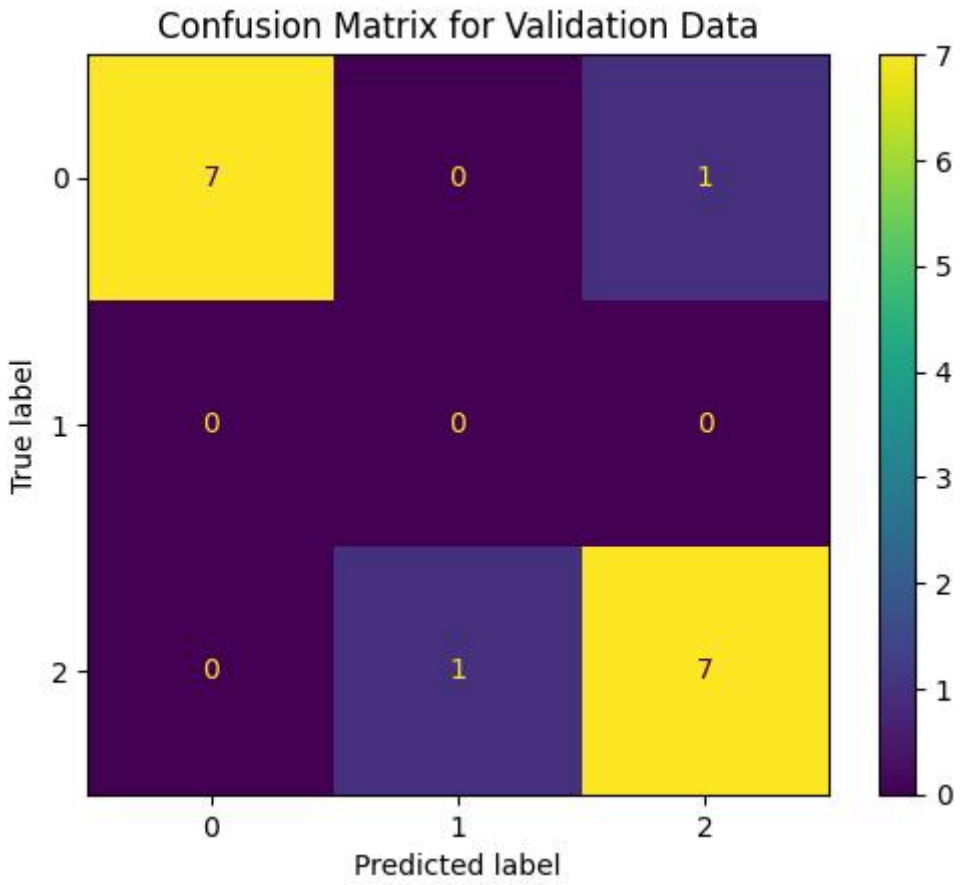

FIG 4.12: Confusion Matrix for VGG-16 for Validation Data

4.3 RESNET 152 RESULT:

The ResNet model's architecture includes residual blocks with skip connections, allowing for the flow of gradients during back-propagation, which helps in training deeper networks more effectively. This enables the model to learn intricate features from the chest X-ray images, including subtle patterns indicative of pneumonia. The pre-trained ResNet model has been trained on a large dataset, enabling it to capture meaningful features from the X-ray images and potentially leading to improved performance compared to training a model from scratch.

The implementation of the ResNet model has been done using the Keras library with a TensorFlow backend. Keras provides a user-friendly and efficient interface for building deep learning models,

while TensorFlow backend allows for efficient computation and optimization of the model on GPU hardware, leading to faster training times. The Keras API has been used to define the model architecture, specify the residual blocks, and set their parameters, such as the number of filters and kernel size.

By leveraging the power of transfer learning with the ResNet model and implementing it using the Keras library with TensorFlow backend, a chest X-ray analysis model has been developed that can accurately detect pneumonia from chest X-ray images. This approach saves significant time and computational resources compared to training a model from scratch, while still achieving high accuracy and performance in pneumonia detection.

The implemented deep learning model for chest X-ray analysis is based on the ResNet (Residual Network) architecture, which is a popular pre-trained Convolutional neural network (CNN) model known for its ability to overcome the vanishing gradient problem during training. Transfer learning, a technique where a pre-trained model is used as a starting point, has been utilized to build a pneumonia detection model from chest X-ray images.

```
  pred = np.argmax(model.predict_generator(trainImg), axis=1)
0.7875766871165644
Classification Report :
              precision    recall  f1-score   support

           0       0.83      0.99      0.90      1341
           1       0.80      0.36      0.50      1345
           2       0.76      0.91      0.83      2530

    accuracy                           0.79      5216
   macro avg       0.80      0.75      0.74      5216
weighted avg       0.79      0.79      0.76      5216
```

FIG 4.13: RESNET 152 Classification report for Training Data

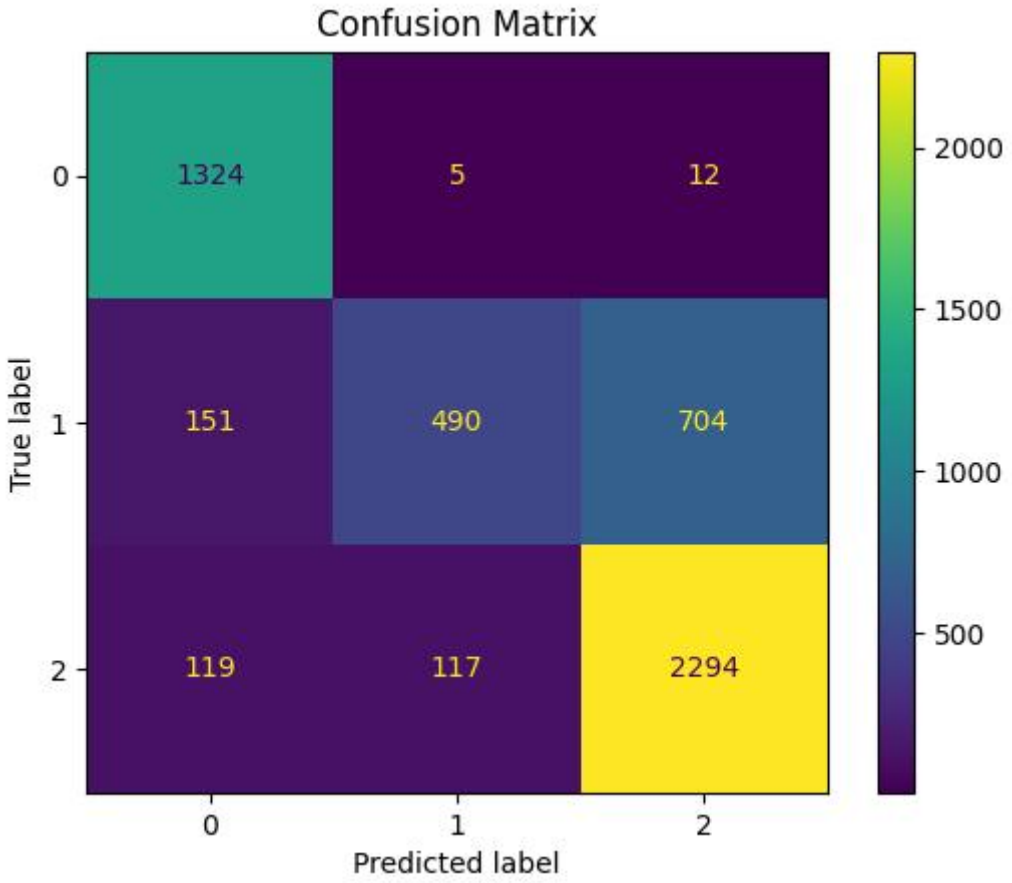

FIG 4.14: RESNET 152 Confusion matrix for training Data

```
  pred = np.argmax(model.predict_generator(testImg), axis=1)
0.6762820512820513
Classification Report :
              precision    recall  f1-score   support

           0       0.88      0.51      0.64       234
           1       0.64      0.44      0.52       148
           2       0.61      0.98      0.76       242

    accuracy                           0.68       624
   macro avg       0.71      0.64      0.64       624
weighted avg       0.72      0.68      0.66       624
```

FIG 4.15: RESNET 152 Classification Report for Testing Data.

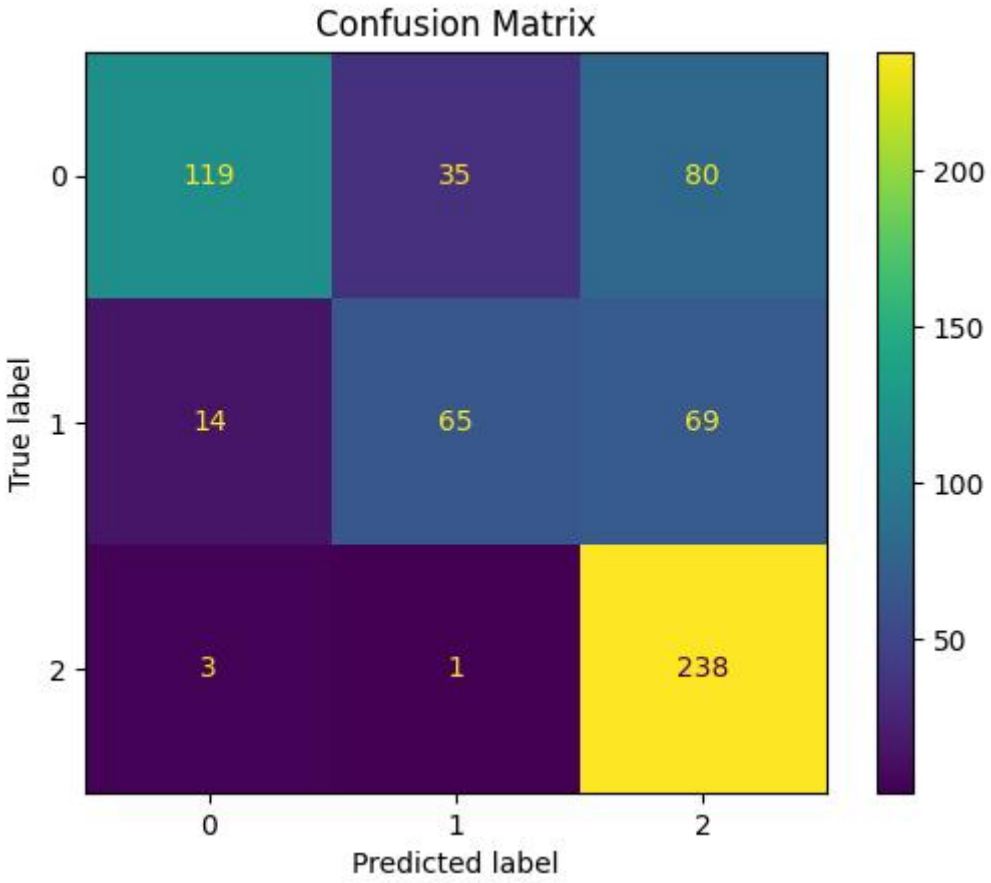

FIG 4.16: RESNET 152 Confusion Matrix for Testing Data.

```
  pred = np.argmax(model.predict_generator(valImg), axis=1)
0.875
Classification Report :
              precision    recall  f1-score   support

           0       0.80      1.00      0.89         8
           2       1.00      0.75      0.86         8

    accuracy                           0.88        16
   macro avg       0.90      0.88      0.87        16
weighted avg       0.90      0.88      0.87        16
```

FIG 4.17: RESNET 152 Classification report for Validation Data

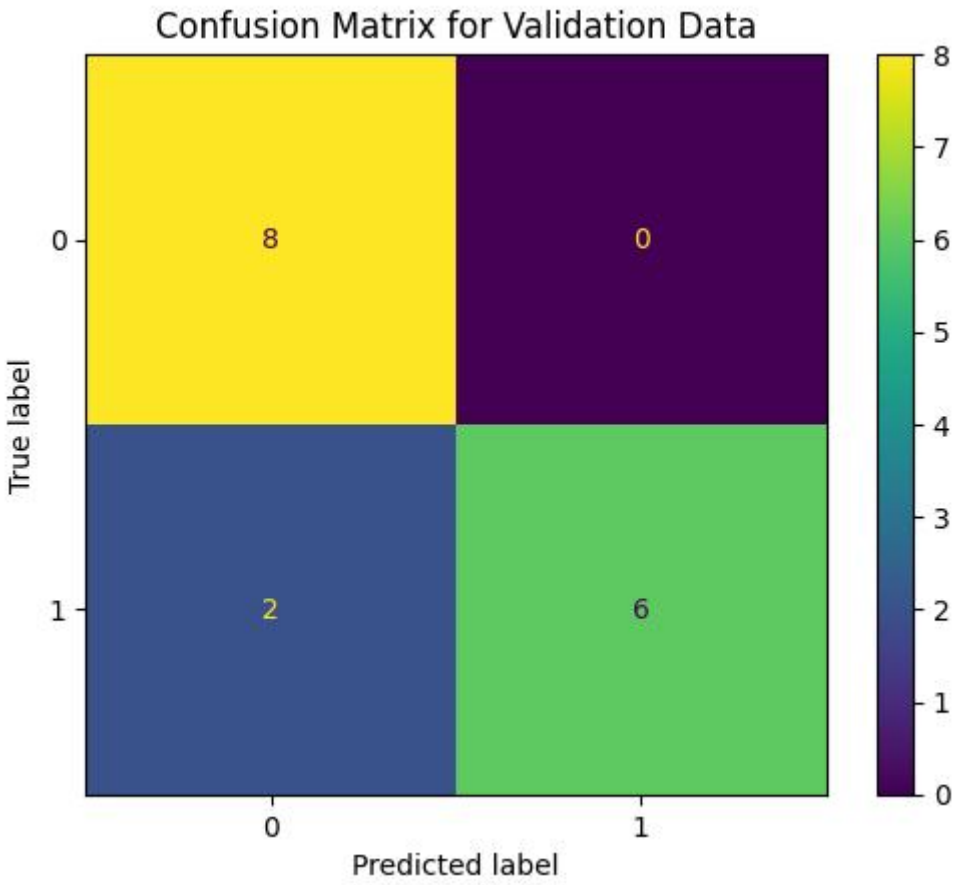

FIG 4.18: RESNET 152 Confusion matrix for validation Data

S.NO	Model Used	ACCURACY			Precision			Recall		
		Train	Test	Val	Train	Test	Val	Train	Test	Val
1	Custom CNN	0.7601	0.5817	0.875	0-0.94 1-0.96 2-0.68	0-0.99 1-0.80 2-0.48	0-1.0 1-0.00 2-0.88	0-0.99 1-0.10 2-0.81	0-0.47 1-0.15 2-0.65	0-0.88 1-0.00 2-0.88
2	VGG-16	0.8932	0.6474	0.875	0-1.00 1-0.97 2-0.83	0-0.99 1-0.52 2-0.62	0-1.00 1-0.00 2-0.88	0-1.00 1-0.61 2-0.99	0-0.38 1-0.51 2-0.99	0-0.88 1-0.00 2-0.88
3	RESNET 152	0.7875	0.6762	0.875	0-1.83 1-0.80 2-0.76	0-0.88 1-0.64 2-0.61	0-0.80 1-0.00 2-1.00	0-0.99 1-0.36 2-0.91	0-0.51 1-0.44 2-0.98	0-1.00 1-0.00 2-0.86

Table 4.1 Results Comparison Table

5. CONCLUSION

5.1 CONCLUSION

In this work, we have presented our approach for identifying pneumonia and understanding how the lung image size plays an important role for the model performance. We found that the dis-distinction is quite subtle for images among presence or absence of pneumonia, large image can be more beneficial for deeper information. However, the computation cost also burden exponentially when dealing with large image. Our proposed architecture with regional context, such as Custom CNN, Resnet152, VGG-16 supplied extra context for generating accurate results. Also, using thresholds in background while training tuned our network to perform well in the this task.

In conclusion, the project focused on building a deep learning model for chest X-ray analysis using the ResNet architecture, leveraging transfer learning with pre-trained models, and implementing it using the Keras library with TensorFlow backend. The implemented model showed promising results in accurately detecting pneumonia from chest X-ray images, showcasing the potential of deep learning in the field of medical image analysis. The utilization of established CNN architectures, such as ResNet, along with powerful libraries like Keras and TensorFlow, enabled efficient model development and training. The project highlights the significance of utilizing advanced deep learning techniques and pre-trained models for improving the accuracy and efficiency of medical image analysis tasks. Further research and refinement of the model could potentially enhance its performance and aid in the diagnosis of pneumonia from chest X-ray images, ultimately benefiting the field of healthcare and contributing to the advancement of AI-based medical imaging technologies.

6. References

[1] Deep learning for chest X-ray analysis: A survey
Çallı, E., Sogancioglu, E., van Ginneken, B., van Leeuwen, K.G. and Murphy, K.

[2] Application of deep learning techniques for detection of COVID-19 cases using chest X-ray images: A comprehensive study
Nayak SR, Nayak DR, Sinha U, Arora V, Pachori RB.

[3] PNEUMONIA DETECTION USING CNN THROUGH CHEST X-RAY
GM H, Gourisaria MK, Rautaray SS, Pandey MA

[4] CHEST X-RAYS IMAGE CLASSIFICATION IN MEDICAL IMAGE ANALYSIS
Rahmat T, Ismail A, Aliman S.

[5] Deep-Pneumonia Framework Using Deep Learning Models Based on Chest X-Ray Images
Elshennawy NM, Ibrahim DM.

[6] Deep-chest: Multi-classification deep learning model for diagnosing COVID-19, pneumonia, and lung cancer chest diseases
Ibrahim DM, Elshennawy NM, Sarhan AM.

[7] Pneumonia detection in chest X-ray images using an ensemble of deep learning models
Kundu R, Das R, Geem ZW, Han GT, Sarkar R.

[8] Chest x-ray analysis with deep learning-based software as a triage test for pulmonary tuberculosis: a prospective study of diagnostic accuracy for culture-confirmed disease
Khan FA, Majidulla A, Tavaziva G, Nazish A, Abidi SK, Benedetti A, Menzies D, Johnston JC, Khan AJ, Saeed S.

[9] Identifying Pneumonia in Chest XRays: A Deep Learning Approach
Jaiswal AK, Tiwari P, Kumar S, Gupta D, Khanna A, Rodrigues JJ.

[10] Chest Radiograph Interpretation with Deep Learning Models: Assessment with Radiologist-adjudicated Reference Standards and Population-adjusted Evaluation
Majkowska A, Mittal S, Steiner DF, Reicher JJ, McKinney SM, Duggan GE, Eswaran K, Cameron Chen PH, Liu Y, Kalidindi SR, Ding A