Early Screening of Lung Disease using Deep Learning

Stuti Tiwari[1], Sanskriti Vidushi[1], Archie Sachdeva[1], Gangupomu Hemanthi[1], Prachi Verma[1], Ritika Kumari[1,2], Poonam Bansal[1]

[1]Department of Artificial Intelligence and Data Sciences, IGDTUW, Delhi, India.
[2]USICT, GGSIPU, Delhi, India.
stuti137btcsai22@igdtuw.ac.in, sanskriti114btcsai22@igdtuw.ac.in, archie031btcsai22@igdtuw.ac.in, gangupomu045btcsai22@igdtuw.ac.in, prachi094btcsai22@igdtuw.ac.in, ritikakumari@igdtuw.ac.in, poonambansal@igdtuw.ac.in.

Abstract

Lung Diseases refer to diseases or disorders that affect the lungs and prevent them from functioning properly. The utilisation of Artificial Intelligence in predicting lung diseases have served as a major advantage in the medical field. The detection of presence of lung diseases in a human body through the development of Deep Residual UNET (ResUNET), a Deep Learning Semantic Segmentation technique, has been classified in this paper. UNET architecture along with residual neural network was use in creation of the model. Binary Cross-Entropy classification was used to facilitate efficient training, optimization and comparison in the model. Loss function and Performance Evaluation metrics like Jaccard Index, IoU, Dice Coefficient and Accuracy were used to enhance the model output. The dataset of chest X-Rays and their masks were taken as input variables for model. The predicted masks by the ResUNET model were the evaluated output. The results showed that our model evaluation is estimated at 98.10 percent accuracy.

Keywords: Lung Disease Diagnosis, Deep Residual UNET, Deep Learning, Image Segmentation , Semantic Segmentation, Binary Classification

1 Introduction

Lung diseases are some of the most common medical conditions in the world. Our lungs are part of a complex system, expanding and relaxing thousands of times each day to bring in oxygen and send out carbon dioxide. Lung diseases can occur when there are problems in any part of this system. Lung ailments are caused by bacterial, viral, fungal infections as well as environmental factors. the most common diseases include asthma, chronic obstructive pulmonary (COPD) disease, infections such as tuberculosis, influenza, lung cancer, pneumonia and other breathing problems [1]. Medical imaging is the technique and process used to create images of the human body for clinical purposes for diagnosis and analysis or medical science. A multitude of imaging tests are employed to create images of the body for diagnosis and analysis of disease detected. Similarly, to detect lung diseases, Lung imaging tests are used which take pictures of the lung or its airways to help healthcare providers diagnose and monitor lung conditions. Various imaging tests like Chest X-Rays, Chest Computed Tomography (CT) scans and Chest Magnetic resonance imaging (MRIs) are utilised in case of lung disease diagnosis [1][2]. The most commonly utilised methods to detect lung infections, the severity or stage of the infection, and the level of lung involvement or impairment after the infection are Chest X-rays and CT scans. Although CT scans can become much more impactful and developed in the near future, as of now chest X-ray has an advantage over CT in having low radiation dose, being cost-effective, easy availability, and fast results [3]. For our model, the dataset incorporated was "Pulmonary Chest X-Ray Defect Detection". The dataset contained chest x-rays for a variety of disease-affected lungs and unaffected lungs as well as the accurate masks of every x-ray.

In the recent years, the incorporation of Deep Learning in Medical fields has proved as nothing short of a miracle. Deep learning plays a critical role in analysis of large datasets because of its capacity to process huge amount of raw, natural data to uncover knowledge from the complex system. Deep learning methods use multiple layers to comprehend features of the data with multiple levels of abstraction. Deep Learning approaches allow computers to learn complicated concepts by building them out of simpler ones [4][5]. The most important utilisation of AI in medicine was in feature extraction algorithms for medical imaging. The emergence of DL in medical imaging has eliminated the need of feature extraction algorithms as the DL systems generate features internally and bypass the less effective feature extraction stage. In DL, the feature extraction is done internally to estimate the location of the desired shape by applying pixel-to-pixel characterization the output the estimated shape of an element in an image, thus providing an accurate projection of the shape and a generalized mechanism for segmentation images and videos. Deep Learning is implemented through the use of Neural Networks. Neural Networks are layers of nodes such that the nodes in every layer are connected to the adjascent layer. More the number of Neural Networks in the model, the deeper the network becomes [5][6] .

But, with increasing depth of neural networks, it becomes more difficult to train due to the problem of vanishing gradients. The accuracy of the model gets saturated after a particular level and results in higher training error. This is where Deep Residual Networks come into play. Deep Residual Networks utilise layers of residual blocks

to build a network. Through the use of DRNs, the training of the model is simplified such that adding more layers does not increase the training error of the network and allows the network to go more in depth [6]. To further rectify the problem of exploding gradients and training of a very deep architecture, Fully Convolutional Neural Networks are used which utilize an identity mapping on the deep residual learning frameworks and help is easier training. Fully Convolutional Networks combine layers of the feature hierarchy and refine the spatial precision of the output. They combine the final prediction layer with lower layers with finer strides, essentially turning into a Directed Acyclic Graph (DAG) with edges skipping connections from lower layers to higher layers [7].

Instead of using FCNs which skip over layers, UNET which is a U-shaped encoder-decoder network architecture is used. UNET is based on FCN but operates on fewer training samples. The Encoder network of the UNET consists of repeated application of two 3x3 convolutions, each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling while the Decoder network consists of an upsampling of the feature map followed by a 2x2 convolution that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. ResUNET is the combination of both the deep residual network framework and the UNET framework. ResUNET is built based on the architecture of U-Net but it uses residual units instead of plain neural units as basic blocks to build the deep ResUNET. The biggest advantage in using ResUNET is being able to build a deeper network without worrying about the training of the model and the problem of vanishing gradients as they are backpropagated from the outer layers to the deeper layers [8].

2 Literature Review

S. Bharati et al. [9] utilized a model incorporating CNN, Vanilla NN, VDSNet, VGG, and Capsule network, employing a hybrid deep learning approach to detect lung diseases from X-ray images collected from a sample dataset consisting of 5606 images. Their findings indicated that VDSNet achieved an accuracy rate of 73 percent. Adem Tekerek et. al [10]employed the approach for the Prediction of Lung Disease Using Chest X-ray Images Based on DenseNet and MobileNet which located and categorised 40,000 X-ray pictures from a sample dataset. The model used Convolution neural network based on chest X-ray classification of three categories of infection with the help of Deep learning. The proposed approach achieved a high accuracy of 96 percent and an ROC AUC score of 0.94. Shoji Kido et al [11] Detection and Classification of Lung Abnormalities by the use of Convolutional Neural Network (CNN) and Regions with CNN Features (R-CNN) used 163 patients with lung nodule cases on CT images. They used 372 patients with or without diffuse lung abnormalities on CT images. Image-based CADx by use of CNN was set to differentiate various kinds of lung abnormalities. CADx by use of R-CNN has proved useful for radiologists' Slung abnormalities diagnosis. Siddhanth Tripathi et. al [12] utilized a model that combines CNN (Convolutional Neural Network), VGG, data preprocessing, and Transfer Learning (TN) to detect

lung diseases using deep learning. They used sample data outputs for the respective lung X-rays as input. The CNN+VGG+data+TN model performed better than the other model (Vanilla CNN), achieving an accuracy of 69.3Kamran Javaid et. al [13] conducted a study on Lung nodule detection in Chest X-ray using deep learning and it was found that the combination of transfer learning model VGG16 and Custom CNN is better than other models with an accuracy of 88Syamala KPL et. al [14] conducted a study on the Detection and Classification of Lung Diseases using Machine and Deep Learning Techniques. Their paper contains a hypothesis that utilizes deep learning and machine learning. They have obtained an accuracy of 98.58 percent using EfficientNet B0. M. Jasmine Pemeena Priyadarsini et.al [15] conducted Lung Diseases Detection Using Various Deep Learning Algorithms. Their research drew upon a sample dataset sourced from Kaggle, encompassing a wide spectrum of lung-related problems. They implemented a state-of-the-art CNN with various models to classify lung diseases accurately. The results demonstrated F1 score of 98.55 percent and accuracy of 98.43 percent. Rakshit S. et. al [16] explored to understand the use of different pretrained deep learning models like Resnet and Densenet to perform the classification. They a proposed model (network of Resnet18) having few parameters for training and testing among the models which have been tested previously. Rasika Naik et.al [17] detected Lung Diseases using Deep Learning.Convolutional neural network (CNN) was proven to be effective in image recognition and classification. DenseNet is a CNN-based pre-trained model and Densenet-121 is the training model. AES algorithm is found to be one of the best. An average AUC of 0.843 was achieved. Marios Anthimopoulos et. al [18] conducted a study on Lung Pattern Classification for Interstitial Lung Diseases Using a Deep Convolutional Neural Network. It was found that they had proposed a deep CNN to classify lung CT image patches into 7 classes, including 6 different ILD patterns and healthy tissue. A neural network Architecture was also designed with this their proposed approach gave promising results with an accuracy of 85.5 percent. Yuanyuan Peng et. al [19] conducted a study on Pulmonary Lobe Segmentation in CT Images Based on LungAnatomy Knowledge and developed a system using developed in MATLAB and C++. To remedy the problem, they introduce a new framework based on lung anatomy knowledge for lung lobe segmentation. As a result, they developed a new lung lobe segmentation scheme that described method has a good performance in pulmonary fissure detection and lung lobe segmentation. Ishan Sen et. al [20] employed the Depth Analysis of Lung Disease Prediction method Using Machine Learning Algorithms.It was found that Random Forest gives a best performance than LR, Bagging, LMT and Bayes Net. They have obtained an accuracy of 90.15 percent using Random Forest.

3 Model Architecture

ResUNET is based on both UNET and Deep Residual Neural Networks, thus making its architecture quite similar to the UNET. A ResUNET consists of an expansion path, a contraption path, and a bridge connecting these networks, similar to the UNET. In order to facilitate the segmentation process, the contraption path collects the features from the image and copies them to higher levels. This transferring of features from

lower level to higher ones helps in better propagation of information, well-defined backpropagation and ease of training. The main feature of ResUNET is that it uses pre-activated residual blocks. The involvement of the residual block helps in better training of the model in deeper neural networks. The residual neural networks thus used constitute of layers of stacked residual units where each residual unit had batch normalisation functions and ReLU activation functions, thus making these residual units pre-activated [21]. The combined use of UNET and residual blocks solved the problem of training deep neural networks with limited data samples by utilising a pre-trained model and helped ease the degradation problem by skipping connections and using lesser parameters to attain better results. An overview of the architecture of the ResUNET model used is shown in Fig. 1.

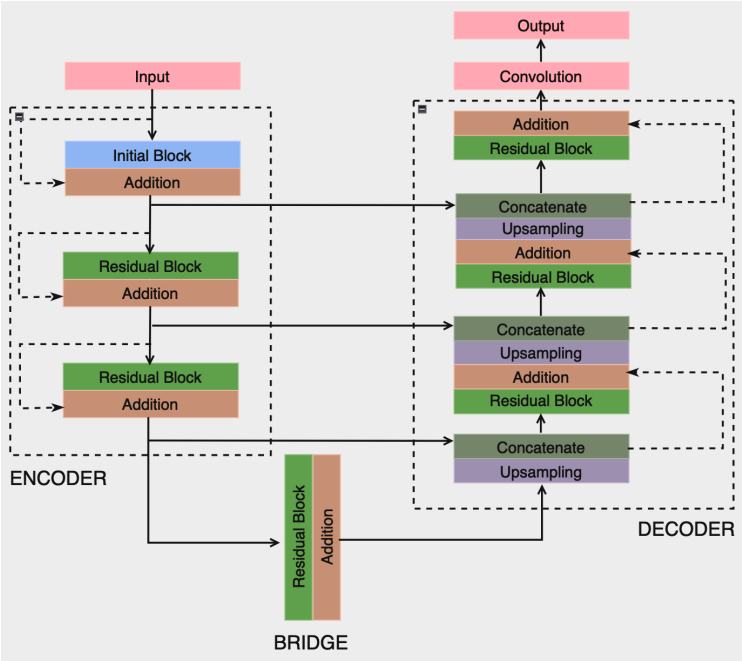

Fig. 1 ResUNET Architecture

4 Materials and Methods

In this section, we will discuss about the classification of our model, the dataset considered for our model and the model's performance metrics.

4.1 Classification

To evaluate the model and check the prediction and optimization of our model, the Binary Cross-Entropy classification is utilised. The incorporation of the Binary classification is done to segregate our observations based on the segmentation of our sampled

features. In our model, binary classification is being used to predict the new mask for each existing original masks by means of binary comparisons. But the prediction of how correct the output is still remains unclear. To tackle this issue, cross-entropy loss function is being used. Loss functions tell us how well our model is performing, based on which optimizations are made. Entropy is the machine learning metric that is a measure of the unpredictability associated with the model or system. More the probability of discrepancy, more is the entropy associated. But, entropy alone is associated with only one distribution at a time. In order to collate and differentiate between two different distributions, what we use is cross-entropy. Cross-entropy is the measure of entropy difference between two probability distributions. When optimizing classification models, cross-entropy is commonly employed as a loss function. Binary cross entropy is a loss function that is used in binary classification tasks. The Binary Cross-Entropy is very convenient for training and solving classification problems by reducing each each classification to a binary choice. The comparison between the predicted and the actual binary outcomes, in our case the comparison between the predicted and true masks, is done by Binary Cross-Entropy loss function, thus emphasizing the importance of this classification [22] .

4.2 Dataset Description

The dataset taken for our model was from Kaggle, titled "Pulmonary Chest X-Ray Defect Detection" [9] [10] [11] [23] [24] [25] . The dataset contained 1408 images of 2 types, 704 images of chest x-rays of varied lungs and 704 images of each lung's original mask. The resolution of each image in the dataset was –. The x-rays of lungs in the dataset are both of disease-affected and disease-unaffected lungs. The mask for each variety of lung was also provided. Fig 2 shown an example of the dataset images.

4.3 Performance Metrics

Performance metrics are used to determine whether the model training is on the right track and find out how well the model is working. The most common metric for evaluating a binary classification method in image segmentation is using Accuracy (1). In our case, Jaccard Index (2) and Dice Coefficient (3) are also employed along with Accuracy.

Accuracy is the most used metric for evaluating the performance of a model in classification problems. It can be defined as the ratio of accurately classified data items to the total number of observations [26].

$$Accuracy = \frac{TP + TN}{(TP + FN) + (FP + TN)} \tag{1}$$

The Jaccard Index is a metric that is used to determine how similar varied distributions are. The value of Jaccard Index stresses resemblance between finite sample sets. The greater the value of Jaccard Index, more is the intersection between the two samples [27].

$$JaccardIndex = \frac{TP}{(TP + FN + FP)} \tag{2}$$

The Dice Coefficient is a statistical tool which measures the similarity between two sets of data. It is the most widely used metric for image segmentation. The Dice Coefficient is numerically equal to twice the area of overlap divided by the total number of pixels in both the images. In boolean description, it is expressed in the form of True Positive, False Positive, True Negative and False Negative [27].

$$DiceCoefficient = \frac{2 * TP}{(2 * TP + FN + FP)} \tag{3}$$

5 Experimental Setup

In this segment, we will discuss the data preprocessing and visualisation before using the data samples for training in the model. The implementation of the ResUNET model is also explained below.

5.1 Data Preprocessing

The images in the dataset, both the x-rays and the masks were processed for faster training. Initially, the compressed zip file containing the images was unzipped. Following this, the Python Imaging Library (PIL) was imported to help manipulate the images like opening, resizing , cropping etc. The x-ray image of lungs were then resized and displayed. Similarly, the mask image of the lung x-ray was first converted to Palette mode (P mode) and set to a mask color and then resized and displayed.

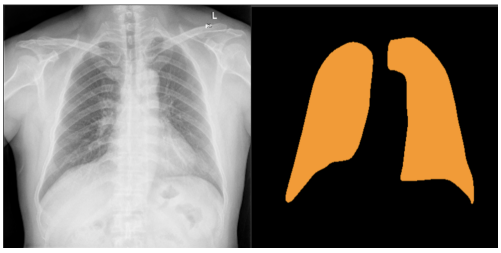

Fig. 2 Sample lung and mask image from the dataset.

A multitude of python and torch libraries like numpy, tqdm, albumentations, torch were then imported for ease of data visualisation and data processing of the model. 3 variables were also created. The first variable was the lung image directory which contained all the lung x-ray images, second was the mask directory which contained all the lung masks and third was the mask name directory which contained a list of names of all the lung masks. The data was shuffled such that 15 percent of the length of mask names of the total dataset is saved to an array. This array was randomised at 99 seed and permuted such that for every run, different images of the total dataset are considered. The data was then split such that 15 percent of data goes into validation set while 85 percent goes to training set. A class was created in which the images and masks were converted to grey-scale and into a float array. The white pixels of the masks were also converted to binary mask. The images in the training and validation sets

were transformed by resizing to 512x512, normalised between 0 to 1 and converted to Tensor and were then saved in training transformation and validation transformation variables respectively.

5.2 Data Visualisation

The training transformation and validation transformation mask names from the above class were split and saved into training dataset and validation dataset variables. DataLoader object was also used in which the data was loaded onto the train dataloader and validation dataloader variable. the batch size of the dataloader used was 4 i.e. in every dataloader iteration, 1/4th of the entire data was loaded onto the variables. A function was created to display the images which were in the tensor form. For this the tensor images were transposed and the pixel of each image was normalised by multiplying it with standard deviation and adding the mean into it.

$$img = img * std + mean \tag{4}$$

Iterations were then performed on the training dataset and validation dataset variables at each index to fetch the images. The x-ray and mask images were displayed in a grid of 4 images, with 2 images in every row as shown in Fig. 3.

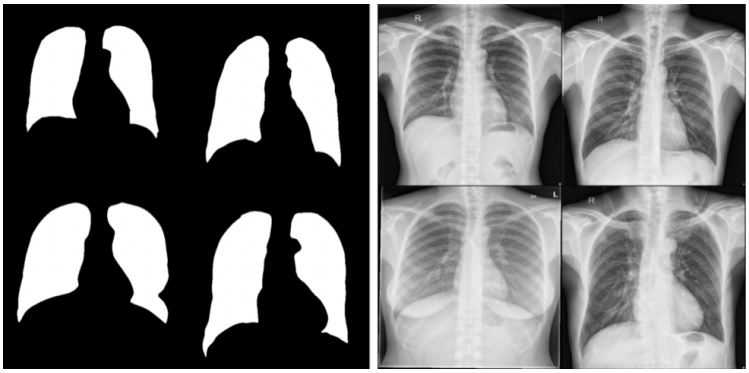

Fig. 3 Lung masks and x-ray samples after grey-scaling.

5.3 Implementation of ResUNET model

The ResUNET model architecture is made up of two class blocks: initial block and residual block, which are used for feature extraction and boosting model efficiency, respectively.

The first stage is the Initial block or the 'InitBlock'. The 'init' function of the initial block is used to initialize the attributes and properties of the object. There are several object initializations in the initial block like 'inchannels', 'outchannels', and 'stride'. These parameters allow the 'InitBlock' object to be customized when it is created. They define the block's configuration. Here, 'inchannels' refer to the number of input channels that this block is expecting. It is set to 3, which is the most common

value for RGB color pictures. 'Outchannels' refer to the number of output channels or feature maps generated by this block. The depth of the feature maps is determined by this parameter. The 'stride' parameter determines the size of the step while applying convolutional layers. It is set to 1 by default, indicating that no downsampling is performed. Within the 'InitBlock' class, the 'self.convblock' property is specified as a series of layers that is a neural network block. The 'self.convblock' has multiple layers of distinct properties. The first layer is the layer of Convolution or 'nn.Conv2d'. This is the initial layer of 'self.convblock' and has several parameters like inchannels, outchannels, kernel size, stride and padding. This layer conducts a 2D convolution process on the input data. In this layer, there are 3 input channels. The outchannels are the number of feature mappings or output channels created by this convolutional layer. The kernel size is the size of the convolutional kernel, which in this case is a 3x3 kernel. Stride is the convolution operation's stride, which influences the spatial dimensions of the output feature maps. The padding (=1) is being used to the input feature maps before convolution to preserve spatial dimensions. The next layer is 'nn.BatchNorm2d' which is to perform the normalization of batches. It has outchannels that apply batch normalization to the preceding convolutional layer's output. Batch normalization aids in training stability by normalizing activations within each mini-batch. It has the potential to speed up training and strengthen the network. Following the batch normalisation layer is the activation of a Rectified Linear Unit or 'nn.ReLU'. After batch normalization, a ReLU activation function is applied to the feature maps element by element. By setting negative values to zero and letting positive values to flow through intact, ReLU introduces non-linearity. Another layer of Convolution or 'nn.Conv2d' is introduced, containing parameters inchannels, outchannels, kernel size of 3 ans padding of 1. In self.convblock, this is the second convolutional layer. It applies additional 2D convolution operation to the feature maps produced following ReLU activation. The characteristics retrieved by the preceding layers are refined in this layer. To summarize, self.convblock describes a layer sequence often used in deep learning architectures, notably Convolutional Neural Networks. Convolutional layers are used for feature extraction, batch normalization is used for normalizing, and ReLU activation is used to introduce non-linearity. As part of a broader neural network architecture, this block is meant to collect and alter characteristics in input data.

Following the self.convblock, comes another block which is the 'self.convskip'. The InitBlock class's self.convskip attribute, like self.convblock, establishes another series of layers. It is used in the block to construct a skip or shortcut connection. The self.convskip block also has several layers. The first layer is the layer of Convolution or nn.Conv2d. nn.Conv2d has several parameters like inchannels, outchannels, kernel size of 3, stride and padding of 1. On the input data, it conducts a 2D convolution process. The inchannels are the number of input channels, similar to the self.convblock inchannels parameter. It defines the number of convolution operation input channels. The outchannels parameter is also the same as that of the self.convblock outchannel parameter. The size of the convolutional kernel is 3x3. The stride for the convolution process controls how the spatial dimensions of the output feature maps vary. The padding(of 1 here) is added to the input feature maps before convolution to retain the

spatial dimensions. The next layer in self.convskip is the nn.BatchNorm2d for normalization of batches. This phase like batch normalization in self.convblock, aids in the stabilization of training and the convergence of the neural network. The objective of self.convskip is to build an alternative channel, known as a skip connection or identity shortcut, that avoids some of the processes in self.convblock. The skip connection is used to add the original input to the processed output, allowing information from the input to flow directly to the output. This architectural element is frequently encountered in residual neural networks (ResNets) and aids in the mitigation of the vanishing gradient problem during training. To summarize, self.convskip is a layer sequence that performs a basic convolution operation and batch normalization. It is used to construct a skip connection within the InitBlock to assist the flow of information from the input to the output while enabling the major feature extraction procedures in self.convblock to take place. Fig. 4 shows a layout of the methods used in the initial block. This block's forward pass is defined by the 'forward' method of the InitBlock class. It goes through two concurrent pathways with the input data x, self.convblock and self.convskip. The feature maps generated by these routes are element-wise joined together to form the block's final output. Because of the skip connection, the model can learn both fine-grained features from the convolutional layers and valuable information from the input.

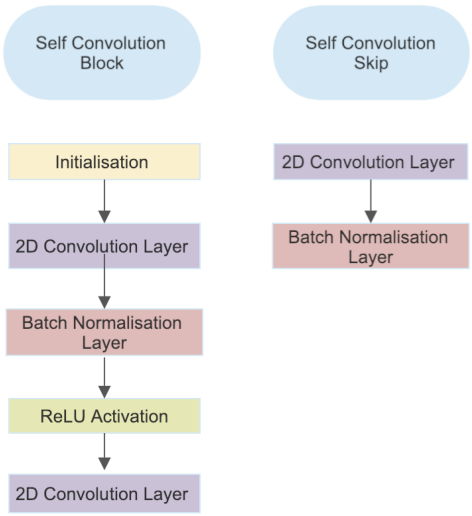

Fig. 4 Propagation in methods of Initial Block

The second stage of the implementation is through the ResBlock which represents the residual block often used in deep neural networks, notably in ResNet topologies. Residual blocks can enhance deep network training and performance by minimizing the vanishing gradient problem. The first method which is the init method loads the ResBlock module. It requires three inputs, inchannels, outchannels, and stride which control the number of input channels, number of output channels (filters), and stride of

the convolutional layers. Similar to the InitBlock, the ResBlock also has self.convblock and self.convskip methods. The self.convblock is a series of convolutional and batch normalization layers that extract and manipulate features. The procedure begins with batch normalization or nn.BatchNorm2d, followed by ReLU activation or nn.ReLU. Following that is a 2D convolutional layer nn.Conv2d with input channels and output channels, a 3x3 kernel, and padding of 1. This layer decreases the spatial dimensions by the supplied stride value. The first convolutional layer is followed by another batch normalization and ReLU activation. Finally, there is a second 2D convolutional layer with outchannels input and output channels, a 3x3 kernel, and padding of 1. This layer keeps spatial dimensions intact. The self.convskip is a parallel branch that does a batch normalized 2D convolutional process. The convolutional layer has inchannels and outchannels with a 3x3 kernel and padding of 1. The stride parameter is also used here. Following the convolutional layer is batch normalization or nn.BatchNorm2d. A new method, the forward method is introduces which defines the ResBlock's forward pass. It accepts the tensor x as an input. The input tensor is filtered using self.convblock and self.convskip. Fig. 5 shows a layout of the methods used in the residual block. The ultimate result of the module is the element-wise sum of the outputs of self.convblock and self.convskip, which essentially implements the residual connection. Overall, the ResBlock module is intended to be a building block for deep neural networks, allowing the development of deep architectures while preserving stable and efficient training.

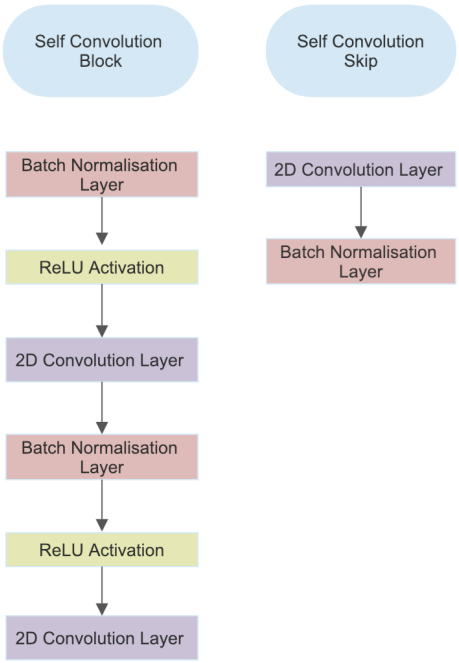

Fig. 5 Propagation in methods of Residual Block

After this, the 'ResUnet' PyTorch module, which depicts a U-Net design with residual blocks comes into direct implementation. There are several components in the ResUnet module. Firstly the init method which sets up the ResUnet module. It requires three parameters, inchannels, outchannels and features. Features are a list containing the number of features available at each level of the U-Net architecture (for example, '[64, 128, 256]'). The module begins with an initial block as previously specified (InitBlock) , which contains the next layer, the Input Layer or self.input. The supplied picture is processed by this layer. Next is the downsampling Path or 'self.downs'. A list of ResBlock modules is constructed for the U-Net's downsampling path. These modules decrease the spatial dimensions while increasing the amount of characteristics. Each ResBlock in the list doubles the amount of features and conducts downsampling with stride of 2. A list of operations is prepared for the U-Net's Upsampling path layer or self.ups. It has 'nn.ConvTranspose2d' layers for upsampling and ResBlock modules for processing the concatenated feature maps while upsampling. Next is the output Layer or self.output which is a 2D convolutional layer that transfers the features to the chosen number of output channels in the final output layer. Finally, 'forward' method is applied which defines the ResUnet's forward pass. It begins with the input layer and collects skip connections at each downsampling step. The skip connections and feature maps are then processed during the upsampling phases. By transferring the result via the output layer, the final output is obtained.

Overall, the ResUnet module combines the U-Net architecture with residual blocks, allowing for more efficient training and improved performance in applications like image segmentation. This module may be instantiated and used for image segmentation tasks by sending input pictures through it. A for loop iterates over a list of 'features' in the supplied code snippet, and each iteration adds a 'ResBlock' module to the 'self.downs' list. Each ResBlock is built with unique input and output channel sizes ('feature' and 'feature * 2', respectively) with a stride of 2. For input features, the loop iterates over a list of 'features'. Each 'feature' indicates the number of channels or feature maps in the input of a convolutional layer. A new ResBlock module is produced and added to the self.downs list for each 'feature' in the list. The ResBlock looks to be intended to increase the number of channels (features) while decreasing the spatial dimensions of the feature maps, often by using stridden convolutions or pooling layers with a stride. For output features, each ResBlock's output channels are given as 'feature * 2', thereby doubling the number of channels relative to the input. This increase in channels is frequent in neural network encoding layers to capture more complicated information. In a nutshell, it is building an encoder component of a neural network, with each ResBlock responsible for downsampling feature maps and increasing the number of channels as part of a hierarchical feature extraction process. This is common in picture segmentation designs such as U-Net or similar encoder-decoder layouts. A for loop iterates across a list of 'features' in reverse order. Each iteration adds two components to the neural network architecture. First, a transposed convolution layer or 'nn.ConvTranspose2d' which is added to the list of 'self.ups'. This layer essentially upsamples the feature maps, increasing their spatial dimensions by a factor of two (stride = 2). For this layer, the input and output channel sizes are both set to '2 * feature', thus doubling the number of channels. After the transposed convolution

layer is the Residual Block ResBlock in which a ResBlock module is added to the list of self.ups modules. This block is connected with an increase in the number of channels and is meant to process the upsampled feature maps. The input and output channel sizes are stated as '2 * feature + feature' and 'feature', respectively, suggesting that this block mixes upsampled feature maps with feature maps from a previous encoding step. This block's stride is set to 1, indicating that it does not do any additional downsampling. In addition, there is a final convolutional layer nn.Conv2d specified as the self.output after the loop. This layer uses a kernel of size to reduce the feature maps produced by the upsampling procedure to the required number of output channels. In essence, this segment of the model specifies the decoder portion of a neural network, where transposed convolutional layers are used for upsampling and ResBlock modules are used to handle the upsampled feature maps. The final output layer generates feature maps with the number of output channels chosen. In semantic segmentation models, this architecture is frequently used to extract high-resolution segmentation masks. The forward function of the neural network class is in charge of the network's forward pass. It illustrates how incoming data travels through the network's layers to generate an output. The forward method initializes an empty list 'skipconnections' to contain skip connections (intermediate feature maps) from the encoding stage. The input data 'x' is transmitted through the first layer self.input, which is frequently the neural network's first encoding layer. The generated feature map is appended to skipconnections. Under encoding, a for loop iterates over the encoding levels specified in 'self.downs'. The input 'x' is transmitted via the encoding layer 'down' during each iteration, which generally conducts downsampling and feature extraction. In 'x', the resultant feature map is updated. If the current iteration index 'i' is less than 2, which indicates the first two encoding layers (zero-based index), the feature map 'x' is appended to skipconnections. These skip connections are needed later in the decoding process. The order of skip connections in skipconnections is reversed after encoding to guarantee that throughout the upsampling operation, the skip connections are appropriately aligned with the decoding layers. Under the decoding segment, second for loop iterates over the decoding layers specified in self.ups. These are often transposed convolutional layers that are in charge of upsampling. The input 'x' is routed via the upsampling layer at each iteration. This technique expands the feature map's spatial dimensions. The matching skip connection from skipconnections is fetched and saved in skipconnections. If the current 'x' feature map's shape does not match the shape of the relevant skip connection, the 'x' feature map is enlarged to fit the shape of the skip connection. This resizing aligns the feature maps in preparation for concatenation. The 'torch.cat' function is used to concatenate the skip connection and the scaled 'x' feature map along the channel dimension (dimension 1). The outcome is saved. Finally, the concatenated feature map 'concatskip' is traversed by the matching residual block to enhance and condense the characteristics. The decoding process's final feature map is supplied through the output layer self.output . This layer decreases the number of channels to match the number of output channels required, which is generally one for binary segmentation. The general architecture presented in this forward technique is a U-Net or similar encoder-decoder neural network, which is often employed for applications like picture segmentation where spatial information

and hierarchical characteristics are important. Using skip connections throughout the decoding process helps to maintain and use characteristics from many scales.

6 Results

The following section describes the evaluation metrics, final results and the learning and loss curves associated with the model.

6.1 Evaluation Metrics

To find out how efficient and optimal our model is, we deployed a loss function. As discusses above, for ease of comparison between the true and the predicted masks, binary classification is being utilised. Corresponding to this, a similar loss function known as the Binary Cross Entropy loss function is being used. Our model uses the standard equation for Binary Cross Entropy loss function to calculate the epoch loss in our model (5) [22] .

$$J_{bce} = -\frac{1}{M}\sum_{m=1}^{M}[y_m log(h_o(x_m)) + (1-y_m)(log(1-h_o(x_m))] \tag{5}$$

For our research, the evaluation metrics used are Dice Coefficient (6) and Intersection over Union or IoU (7). The Dice coefficient is used to gauge the similarities between the two distributions, which in our case are the true masks and the predicted masks. The higher the Dice score, greater is the overlap with the ground truth.

$$DiceCoefficient = \frac{2|X \cap Y|}{|X| + |Y|} \tag{6}$$

IoU is a number from 0 to 1 that specifies the amount of overlap between the predicted and ground truth distributions similar to the predictions of Jaccard Index [27].

$$IoU = \frac{|X \cap Y|}{|X| + |Y|} \tag{7}$$

While metrics like Dice and Jaccard are used in optimisizing loss, IoU is mainly used to increase accuracy of the model.

6.2 Training Results

The training result of our model is done through the use of epochs. The optimal number of epochs for every model depends on the dataset considered and the main factor which comes into picture is the training and validation error and the loss with every epoch. In our case, 20 epochs are taken. At the end of 20 epochs, the accuracy of the model comes out to be 0.9810 or 98.10 percent as shown in Table 1.

The training and validation curve for loss function and performance and evaluation metrics is also developed as shows in Fig. 6 and Fig. 7.

Table 1 Epoch Training Results

S.No.	*Epoch No.*	*Loss*	*Accuracy*	*Dice*	*IoU*
1	1/20	0.7736	0.9076	0.8302	0.7205
2	2/20	0.3646	0.9629	0.9261	0.8634
3	3/20	0.2550	0.9746	0.9493	0.9041
4	4/20	0.2372	0.9764	0.9530	0.9107
5	5/20	0.2449	0.9757	0.9513	0.9077
6	6/20	0.2051	0.9798	0.9597	0.9229
7	7/20	0.2004	0.9802	0.9606	0.9245
8	8/20	0.1973	0.9805	0.9611	0.9255
9	9/20	0.1981	0.9804	0.9609	0.9252
10	10/20	0.1949	0.9807	0.9616	0.9264
11	11/20	0.1939	0.9808	0.9618	0.9267
12	12/20	0.1942	0.9808	0.9616	0.9265
13	13/20	0.1940	0.9808	0.9616	0.9265
14	14/20	0.1927	0.9809	0.9621	0.9273
15	15/20	0.1927	0.9809	0.9619	0.9270
16	16/20	0.1933	0.9809	0.9620	0.9271
17	17/20	0.1929	0.9809	0.9619	0.9270
18	18/20	0.1936	0.9808	0.9619	0.9269
19	19/20	0.9809	0.1923	0.9622	0.9275
20	20/20	0.1915	0.9810	0.9622	0.9276

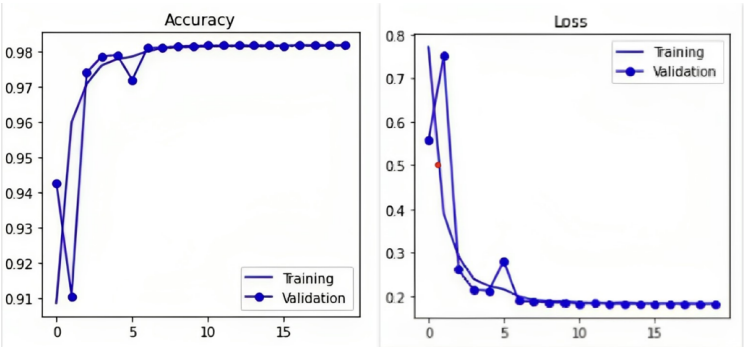

Fig. 6 Training and validation curve for Accuracy and Loss

6.3 Final Output

The derived output from our model based on binary classification and ResUNET is the predicted mask. The closeness of our mask with the true masks from the dataset is given in Fig. 8. As observed by the samples of predicted and true masks as well as based on the 98.1 percent accuracy, we can say that the utilisation of ResUNET and the training of the model has been done successfully.

7 Conclusion

In this research paper we introduce the ResUnet model, for detecting lung diseases using high-resolution chest X ray images. Our approach combines the benefits of Deep learning and the U-Net Architecture and Residual Neural Networks. By utilizing skip

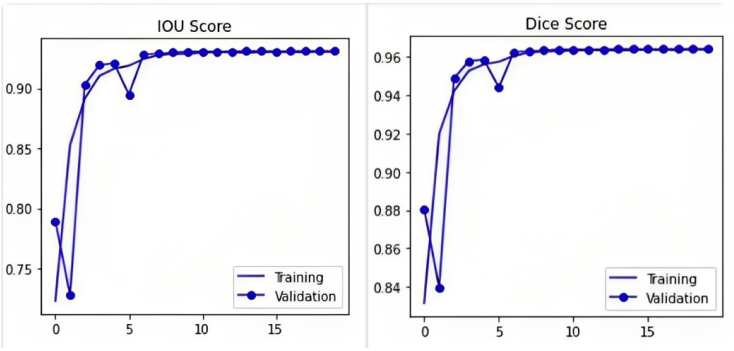

Fig. 7 Training and validation curve for IoU and Dice

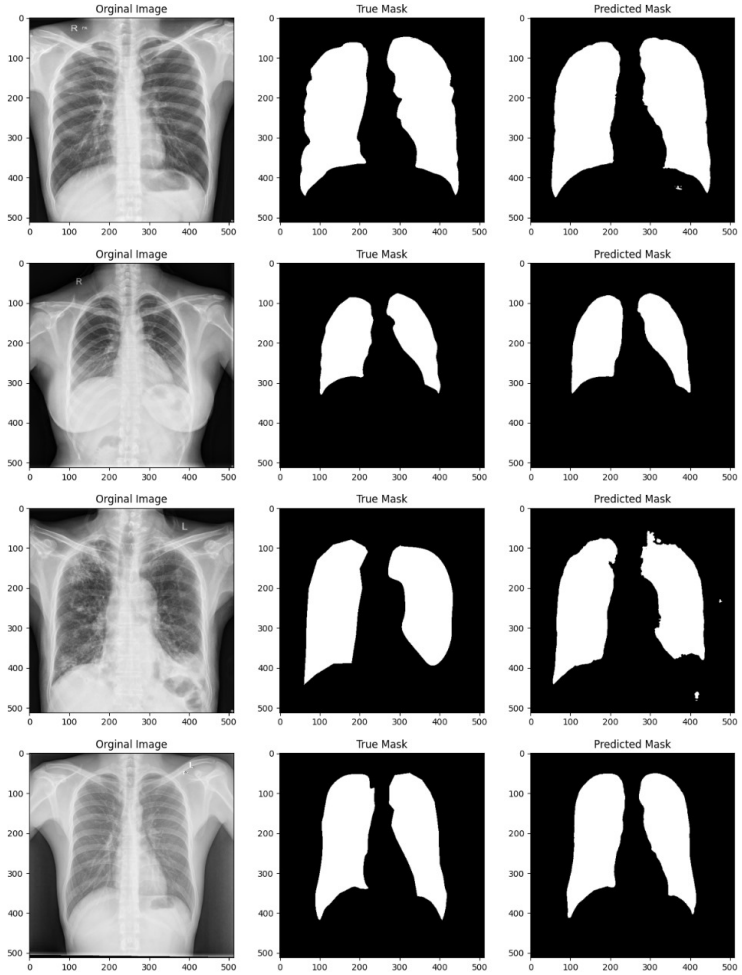

Fig. 8 Final output: Predicted and True masks after model training

connections within the residual blocks and between the downsampling and upsampling paths, our network effectively extracts features and propagates information in both forward and backward computations. This unique architecture incorporated the best state of the art deep learning approaches for lung disease detection using binary classification. Our proposed model not only simplifies training but also enables us to design networks that are both straightforward and powerful.

References

[1] C.Bhuvaneswari, P.Aruna, D.Loganathan, "Classification of Lung Diseases by Image Processing Techniques Using Computed Tomography Images", International Journal of Advanced Computer Research ,Volume-4 Number-1 Issue-14 March-2014.

[2] Vliegenthart R, Fouras A, Jacobs C, Papanikolaou N. "Innovations in thoracic imaging: CT, radiomics, AI and x-ray velocimetry. Respirology", 2022 Oct;27(10):818-833. doi: 10.1111/resp.14344. Epub 2022 Aug 14. PMID: 35965430; PMCID: PMC9546393.

[3] Wang G., Yu H., De Man B. "An outlook on X-ray CT research and development". Med. Phys. 2008;35:1051–1064. doi: 10.1118/1.2836950.

[4] Matiur R. Minar, Jibon Naher. Recent Advances in Deep Learning: An Overview. 10.13140/RG.2.2.24831.10403.

[5] Haruna Chiroma, Shafi'i M. Abdulhamid, Ibrahim A. T. Hashem, Kayode S. Adewole, Absalom E. Ezugwu, Saidu Abubakar, Liyana Shuib. "Deep Learning-Based Big Data Analytics for Internet of Vehicles: Taxonomy, Challenges, and Research Directions". Volume 2021.

[6] Mainak Biswas, Venkatanareshbabu Kuppili, Luca Saba, Damodar Reddy Edla, Harman S. Suri, Elisa Cuadrado-Godia, John R. Laird, Rui Tato Marinhoe, João M. Sanches, Andrew Nicolaides, Jasjit S. Suri. "State-of-the-art review on deep learning in medical imaging". 2019, 24(3), 380–406. https://doi.org/10.2741/4725

[7] Jonathan Long, Evan Shelhamer, Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". 1411.4038.

[8] Zhengxin Zhang, Qingjie Liu, Yunhong Wang. "Road Extraction by Deep Residual U-Net". IEEE Geoscience and Remote Sensing Letters. May 2018. 10.1109/lgrs.2018.2802944.

[9] Bharati, S., Podder, P., and Mondal, M. R. H. (2019). "Hybrid deep learning for detecting lung diseases from X-ray images". Informatics in Medicine Unlocked, 20, 100391. https://doi.org/10.1016/j.imu.2020.100391

[10] Tekerek A, Al-Rawe IAM. "A Novel Approach for Prediction of Lung Disease Using Chest X-ray Images Based on DenseNet and MobileNet" . Wirel Pers Commun. 2023 May 12:1-15. doi: 10.1007/s11277-023-10489-y. Epub ahead of print. PMID: 37360137; PMCID: PMC10177707.

[11] S. Kido, Y. Hirano and N. Hashimoto, "Detection and classification of lung abnormalities by use of convolutional neural network (CNN) and regions with CNN features (R-CNN)," 2018 International Workshop on Advanced Image Technology (IWAIT), Chiang Mai, Thailand, 2018, pp. 1-4, doi: 10.1109/IWAIT.2018.8369798.

[12] Siddhanth Tripathi, Sinchana Shetty, Somil Jain, Vanshika Sharma. (2021). "Lung Disease Detection Using Deep Learning". 10.35940/ijitee.H9259.0610821.

[13] Javaid, K., Rehman, F., and Haq, N. (2020). "Lung nodule detection in chest x-ray using deep learning" . Pakistan Science Bulletin, 70(1), 1-14.

[14] Syamala KPL, Niharika CS, Jenny AM, Pavani P (2022). "Detection and Classification of Lung Diseases using Machine and Deep Learning Techniques". J Comput Sci Software Dev 2: 1-10.

[15] Jasmine Pemeena Priyadarsini M, Kotecha K, Rajini GK, Hariharan K, Utkarsh Raj K, Bhargav Ram K, Indragandhi V, Subramaniyaswamy V, Pandya S. "Lung Diseases Detection Using Various Deep Learning Algorithms". J Healthc Eng. 2023 Feb 3;2023:3563696. doi: 10.1155/2023/3563696. PMID: 36776955; PMCID: PMC9918362.

[16] Rakshit, S., Saha, I., Wlasnowolski, M., Maulik, U., Plewczynski, D. (2019). "Deep Learning for Detection and Localization of Thoracic Diseases Using Chest X-Ray Imagery". In: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J. (eds) Artificial Intelligence and Soft Computing. ICAISC 2019. Lecture Notes in Computer Science(), vol 11509. Springer, Cham. https://doi.org/10.1007/978-3-030-20915-5-25

[17] Naik, Rasika and Wani, Tejas and Bajaj, Sakshi and Ahir, Shiva and Joshi, Atharva. "Detection of Lung Diseases using Deep Learning" (April 8, 2020). Proceedings of the 3rd International Conference on Advances in Science and Technology (ICAST) 2020.

[18] M. Anthimopoulos, S. Christodoulidis, L. Ebner, A. Christe and S. Mougiakakou, "Lung Pattern Classification for Interstitial Lung Diseases Using a Deep Convolutional Neural Network," in IEEE Transactions on Medical Imaging, vol. 35, no. 5, pp. 1207-1216, May 2016, doi: 10.1109/TMI.2016.2535865.

[19] Peng Yuanyuan, Zhong Hualan, Xu Zheng, Tu Hongbin, Li Xiong , Peng Lan. (2021). "Pulmonary Lobe Segmentation in CT Images Based on Lung Anatomy Knowledge". Mathematical Problems in Engineering. 2021. 1-15. 10.1155/2021/5588629.

[20] Ishan Sen, Md Hossain, Faisal Shakib, Asaduzzaman Imran, Faiz Faisal. (2020). "In Depth Analysis of Lung Disease Prediction Using Machine Learning Algorithms" . 10.1007/978-981-15-6318-8-18.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. "Identity Mappings in Deep Residual Networks". 2016. 1603.05027.

[22] Ruby, Usha, Yendapalli , Vamsidhar. "Binary cross entropy with deep learning technique for Image classification". International Journal of Advanced Trends in Computer Science and Engineering. 9. 10.30534/ijatcse/2020/175942020 (2020).

[23] Pandey, Nikhil. "Chest Xray Masks and Labels." Kaggle, https://www.kaggle.com/datasets/nikhilpandey360/chest-xray-masks-and-labels. Accessed 7 Sept. 2023.

[24] Jaeger S, Karargyris A, Candemir S, Folio L, Siegelman J, Callaghan F, Xue Z, Palaniappan K, Singh RK, Antani S, Thoma G, Wang YX, Lu PX, McDonald CJ. "Automatic tuberculosis screening using chest radiographs". IEEE Trans Med Imaging. February 2014.

[25] Candemir S, Jaeger S, Palaniappan K, Musco JP, Singh RK, Xue Z, Karargyris A, Antani S, Thoma G, McDonald CJ. "Lung segmentation in chest radiographs using anatomical atlases with nonrigid registration". IEEE Trans Med Imaging. February 2014.

[26] Nillmani, Sharma N, Saba L, Khanna NN, Kalra MK, Fouda MM, Suri JS. "Segmentation-Based Classification Deep Learning Model Embedded with Explainable AI for COVID-19 Detection in Chest X-ray Scans." Diagnostics (Basel). 2022 Sep 2;12(9):2132. doi: 10.3390/diagnostics12092132. PMID: 36140533; PMCID: PMC9497601.

[27] Ishu Anand, Himani Negi, Deepika Kumar, Mamta Mittal, Tai-hoon Kim,Sudipta Roy. "Residual U-Network for Breast Tumor Segmentation from Magnetic Resonance Images". Computers, Materials and Continua. DOI:10.32604/cmc.2021.014229. 30 November 2020.