A Project Report

on

DETECTION OF PNEUMONIA IN CHEST X-RAY IMAGES

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

by

D. N. V. Kiranmai (12WH1A1217)

G. Arun Jyothi (16WH1A1220)

M. Poojitha (16WH1A1243)

Under the esteemed guidance of

Ms P. Sree Lakshmi
Assistant Professor

Department of Information Technology

BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Accredited by NBA & NAAC '**A**' Grade, Approved by AICTE, and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

April 2020

DECLARATION

We hereby declare that the work presented in this project entitled **"DETECTION OF PNEUMONIA IN CHEST X-RAY IMAGES"** submitted towards completion of the major project in IV year of B.Tech IT at "BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN", Hyderabad is an authentic record of our original work carried out under the esteem guidance of Ms P. Sree Lakshmi, Assistant Professor, IT department.

D. N. V. Kiranmai
(12WH1A1217)

G. Arun Jyothi
(16WH1A1220)

M. Poojitha
(16WH1A1243)

BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Accredited by NBA & NAAC 'A' Grade, Approved by AICTE, and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

Department of Information Technology

Certificate

This is to certify that the Project report on "**Detection of Pneumonia in Chest X-Ray Images**" is a bonafide work carried out by **D.N.V.Kiranmai (12WH1A1217), G. Arun Jyothi (16WH1A1220), M. Poojitha (16WH1A1243)** in the partial fulfillment for the award of B.Tech degree in **Information Technology, BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide
Ms P. Sree Lakshmi
Assistant Professor
Department of IT

Head of the Department
Dr. Aruna Rao S L
Professor and HOD
Department of IT

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K. V. N. Sunitha, Principal**, BVRIT Hyderabad, for providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr**. **Aruna Rao S L, Head**, Department of Information Technology, BVRIT Hyderabad for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Ms P. Sree Lakshmi, Assistant Professor, Department of IT,** BVRIT Hyderabad for her constant guidance, encouragement and moral support throughout the project.

Finally, We would also like to thank our Project Coordinator, all the faculty and staff of IT Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

D. N. V. Kiranmai
(12WH1A1217)

G. Arun Jyothi
(16WH1A1220)

M. Poojitha
(16WH1A1243)

ABSTRACT

Pneumonia is a disease which is caused by a bacterial infection. The infection spreads in the lungs area of a human body. Early diagnosis is an important factor in terms of the successful treatment process. Chest X-rays are currently the best available method for diagnosing pneumonia, playing a crucial role in clinical care and epidemiological studies. However, detecting pneumonia in chest X-rays is a challenging task that relies on the availability of expert radiologists. Medical images diagnosing can be error-prone for inexperienced radiologists, while tedious and time-consuming for experienced radiologists. The appearance of pneumonia in X-ray images is often vague, can overlap with other diagnoses, and can mimic many other benign abnormalities. These discrepancies cause considerable variability among radiologists in the diagnosis of pneumonia. Therefore, computer-aided diagnosis systems are needed to guide the clinicians. Recent advances in computer vision demonstrate that Convolutional Neural Network (CNN) architectures achieve human-level performance on several image-processing tasks, including classification, segmentation, and object detection. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers, and normalization layers. Here, we use a Convolution Neural Network model that can automatically detect pneumonia from chest X-rays at a level exceeding practicing radiologists.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Term/Abbreviation	Definition
CNN	Convolutional Neural Networks
WHO	World Health Organization
CXR	Chest X-Ray
CDC	Centers for Disease Control and Prevention
COPD	Chronic obstructive pulmonary disease
AUC	Area Under Curve
RELU	Rectified Linear Unit
VGG	Visual Geometry Group
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
HCL	Hardware Compatability List
VGA	Video Graphics Array
CPU	Central Processing Unit
GPU	Graphics Processing Unit
API	Application Programming Interface
GUI	Graphical User Interface
PIL	Python Imaging Library
JPEG	Joint Photographic Experts Group
FCN	Fully Connected Layer
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
RMSprop	Root Mean Square Propagation
IDE	Integrated Development Environment

TABLE OF CONTENTS

1. INTRODUCTION

Pneumonia is inflammation of the tissues in one or both lungs that usually caused by a bacterial infection. The risk of pneumonia is immense for many, especially in developing nations where billions face energy poverty and rely on polluting forms of energy. The World Health Organization (WHO) estimates that over 4 million premature deaths occur annually from household air pollution-related diseases including pneumonia [1]. In such regions, the problem can be further aggravated due to the dearth of medical resources and personnel. For these populations, accurate and fast diagnosis means everything. It can guarantee timely access to treatment and save much needed time and money for those already experiencing poverty. Fortunately, pneumonia can be a manageable disease by using drugs like antibiotics and antivirals. However, early diagnosis and treatment of pneumonia is important to prevent some complications that lead to death [2].

Today's deep learning models can reach human-level accuracy in analyzing and segmenting an image. The medical industry is one of the most prominent industries, where deep learning can play a significant role, especially when it comes to imaging. All those advancements in deep learning make it a prominent part of the medical industry.

Deep learning can be used in wide variety of areas like the detection of tumors and lesions in medical images, computer-aided diagnostics, the analysis of electronic health-related data, the planning of treatment and drug intake, environment recognition and brain-computer interface, aiming to come up with decision support for the evaluation of the person's health. The key element of the success of deep learning is based on the capability of the neural networks to learn high level abstractions from raw input data through a general purpose learning procedure [5].

Although currently, deep learning still cannot replace doctors/clinicians in medical diagnosis, it can provide support for experts in the medical domain in performing time-consuming works, such as examining chest radiographs for the signs of pneumonia.

1.1 OBJECTIVE

Deep neural network models have conventionally been designed, and experiments were performed upon them by human experts in a continuing trial-and-error method. Therefore, a novel but simple model is introduced to automatically perform optimal classification tasks with deep neural network architecture.

The neural network architecture was specifically designed for pneumonia image classification tasks. The proposed technique is based on the convolutional neural network algorithm, utilizing a set of neurons to convolve on a given image and extract relevant features from them. Demonstration of the efficacy of the proposed method with the minimization of the computational cost as the focal point was conducted and compared with the exiting state-of-the-art pneumonia classification networks.

1.2 PROBLEM IN EXISTING SYSTEM

Chest X-ray (CXR) images are the best-known and the common clinical method for diagnosing of pneumonia. Diagnosing pneumonia from chest X-ray images is a challenging task for even expert radiologists. The process of pneumonia detection by reading X-ray images can be time consuming and less accurate. The reason is that, the appearance of pneumonia in X-ray images is often unclear, can confuse with other diseases and can behave like many other benign abnormalities. Therefore, accurate reading of images is highly desirable [6]. These inconsistencies caused considerable subjective decisions and varieties among radiologists in the diagnosis of pneumonia. So, several computer assisted diagnosis tools have been developed to provide an insight into X-ray images. However, these tools are not able to provide sufficient information to support doctors in making decisions.

1.3 SOLUTION

The above problem states that, there is a need for computerized support systems to help radiologists for diagnosing pneumonia from CXR images. Recent developments in deep learning field, especially convolutional neural networks (CNNs) showed great success in image classification. The power of computing is known worldwide and developing an identification model for finding pneumonia causes in clinical images can help in accurate and better understanding of the X-ray images. Deep learning based systems have been applied to increase the accuracy of image

classification. These deep networks showed superhuman accuracies in performing such tasks. This success motivated to apply these networks to medical images for Pneumonia classification tasks, and the results showed that deep networks can efficiently extract useful features that distinguish different classes of images. Most commonly used deep learning architecture is the convolutional neural network (CNN). CNN has been applied to various medical images classification due to its power of extracting different level features from images.

1.4 FEATURES

The main idea behind the CNNs is creating an artificial model like a human brain visual cortex. The main advantage of CNNs, it has the capability to extract more significant features from the entire image rather than handcrafted features. There have been developments of different CNN based deep networks and these networks achieved state of results in classification, segmentation, object detection and localization in computer vision [9]. Besides the natural computer vision problems, CNNs achieved very successful results in solving medical problems such as breast cancer detection, brain tumor segmentation, and Alzheimer disease diagnosing skin lesion classification.

Deep learning in medical big data has a huge dominance because the significant hierarchical correlation inside the data can be found by using deep learning models rather than tedious and laborious handcrafting of features. With the large dissimilarity in patient's data, regular learning approaches are untrustworthy. Over the past few years, there is big progress in Deep learning approaches due to its capability to adapt with complex data.

To design these models, specialists often have a large number of choices to make design decisions, and intuition significantly guides manual search process. CNNs possess a visual processing scheme that is equivalent to that of humans and extremely optimized structure for handling images and 2D and 3D shapes, as well as ability to extract abstract 2D features through learning. The max-pooling layer of the convolutional neural network is effective in variant shape absorptions and comprises sparse connections in conjunction with tied weights [12]. When compared with fully connected networks of equivalent size, CNNs have a considerably smaller amount of parameters. Most importantly, gradient based learning algorithms are employed in training CNNs and they are less prone to diminishing gradient problem.

2. LITERATURE SURVEY

The analysis of chest radiography has a crucial role in medical diagnostics and the treatment of the disease. The Centers for Disease Control and Prevention (CDC), Atlanta reported that about 1.7 million adults in the US seek care in a hospital due to pneumonia every year, and about 50,000 people died in United States from pneumonia in 2015 [15]. Chronic obstructive pulmonary disease (COPD) is the primary cause of mortality in the United States, and it is projected to increase by 2025 [16]. The World Health Organization (WHO), Geneva reported that it is one of the leading causes of death all over the world for children under 5 years of age, killing an estimated 1.4 million, which is about 18% of all children deaths under the age of 5 years worldwide. More than 90% of new diagnoses of children with pneumonia happen in the underdeveloped countries with few medical resources available [12]. Therefore, the development of cheap and accurate pneumonia diagnostics is required.

There has been previous studies done regarding pneumonia detection with chest x-rays via machine learning with the use of heat maps [3], which are images or maps representing the varying temperature or infrared radiation recorded over an area or during a period of time, and differentiation of pulmonary pathology, which is the subspecialty of surgical pathology which deals with the diagnosis and characterization of neoplastic and non-neoplastic diseases of the lungs from normal by using computerized lung sound analysis.

Recently, it has been proposed different artificial intelligence -based solutions for different medical problems. Convolutional neural networks (CNNs) have allowed researchers to obtain successful results in wide medical problems like breast cancer detection, brain tumor detection and segmentation, disease classification in X-ray images, etc. [7].

2.1 INFORMATION GATHERING

Wang et al. [11] contributed by proposing a new dataset called ChestX-ray8 that contains images of different 32,717 patients, having 108,948 frontal X-ray images. They achieved promising results using a deep CNN. They further stated that this dataset can be extended using more disease labels. Not only in chest X-rays, Ronneburger et al., used the power of data augmentation and CNN, and achieved great results, even training on small sample of images.

In another study, Roth et al. showed how deep CNN can be used to detect lymph nodes. They got promising results even in the presence of low contrast surrounding structures obtained from computer tomography.

Rajpurkar et al. [3] suggested CheXNeXt, a very deep CNN with 121 layers, to detect 14 different pathologies, including pneumonia, in frontal-view chest X-rays. First, neural networks were trained to forecast the probability of 14 abnormalities in the X-ray image. Then, an ensemble of those networks, were used to issue predictions by calculating the mean predictions of individual networks. Woźniak et al. [2] suggested a novel algorithm for training probabilistic neural networks that allows one to construct smaller networks. Gu et al. used a 3D deep CNN and a multiscale forecasting strategy, with cube clustering and multiscale cube prediction, for lung nodule detection. Ho and Gwak used a pretrained DenseNet-121 and four types of local features and deep features acquired by using the scale-invariant feature transform, Local binary patterns and a histogram of oriented gradients, and CNN features for classification of 14 thoracic diseases.

Jaiswal et al. [15] used Mask-Recurrent CNN, a deep neural network, which utilizes both global and local features for pulmonary image segmentation combined with image augmentation, alongside with dropout and L2 regularization, for pneumonia identification. Jung et al. use a 3D deep CNN which has shortcut connections and a 3D Dense CNN with dense connections. The shortcuts and dense connections solve the gradient vanishing problem. Connections allow deep networks to capture both the general and specific features of lung nodules. Lakhani and Sundaram use AlexNet and GoogLeNet neural networks with data augmentation and without any pre-training to obtain an Area Under the Curve (AUC) of 0.94–0.95. An improved architecture using transfer learning and a two-network ensemble achieved an AUC of 0.99.

Li et al. [14] used a CNN-based approach combined with rib suppression and lung filed segmentation. For pixel patches acquired in the lung area, three CNNs were trained on different resolution images, and feature fusion was applied to merge all information. Liang et al. designed a novel network with residual structures, which has 49 convolutional layers, only one global average pooling layer and two dense connection layers for pediatric pneumonia diagnosis. Nam et al. used a deep CNN with 25 layers and eight residual connections. The outputs of three networks trained with different hyperparameter values were averaged for the prediction of malignant pulmonary nodules in chest radiographs. Nasrullah et al. used two 3D-customized mixed link

network (CMixNet) architectures. Lung nodule recognition was performed with faster Recurrent-CNN on features learned from customized mixed link network and U-Net like encoder–decoder, while a gradient boosting machine was used for classification. Pasa et al. proposed a custom neural network consisting of five convolutional blocks (each having two 3 × 3 convolutions with Rectified Linear Units (ReLUs), followed by a max-pooling operation), a global average pooling layer and a fully-connected softmax layer with two outputs.

Pezeshk et al. suggested a 3D fully CNN for the fast screening and generation of candidate suspicious regions. Next, an ensemble of 3-D CNNs was trained using extensive data augmentations obtained from the positive and negative patches. The classifiers were trained on false positive patches using different thresholds and data augmentation types. Finally, the outputs of second stage networks were averaged to produce the final prediction. Sirazitdinov et al. [6] suggested an ensemble of RetinaNet and Mask Recurrent-CNN networks for pneumonia localization. First, networks recognized the pneumonia-affected regions, then non-maximum suppression was applied in the predicted lung regions.

Souza et al. [7] used AlexNet-based CNN for lung patch classification. Then, a second CNN model, based on ResNet18, was employed to reconstruct the missing parts of the lung area. The output is obtained by the ensemble combination of initial segmentation and reconstruction. Taylor et al. used standard network architectures and explored the parameter space of pooling and flattening operations. Final results were obtained by fully connected layers followed by a sigmoid function.

Wang et al. [11] split X-ray images into six types of patches and used ResNet to classify six patches and recognize lung nodules. They used rotation, translation and scaling techniques for data augmentation. Xiao et al. [4] proposed a multi-scale heterogeneous 3D CNN, which uses multiscale 3D nodule blocks with contextual data as input; (2) two 3D CNNs to acquire expression features; (3) and a set of weights defined by backpropagation for feature fusion. Xie et al. suggested a semi-supervised adversarial classification framework that is trained using both labeled and unlabeled data. They used an adversarial auto encoder-based unsupervised network for reconstruction, a supervised network for classification and transition layers that map from the reconstruction network to the classification network.

Xu et al. [11] designed a hierarchical CNN CXNet-m1, which used a novel sin-loss loss function to learn from misclassified and indistinguishable images for anomaly identification on chest X-rays. Yates et al. retrained a final layer of the CNN model, Inceptionv3, and performed binary normality classification. In addition, da Nóbrega et al. compared the ResNet50 feature extractor combined with the SVM RBF classifier for recognition of the malignancy of a lung nodule.

Ke et al. [8] used the spatial distribution of Hue, Saturation and Brightness in the X-ray images as image descriptors, and an Artificial Neural Network with heuristic algorithms (Moth-Flame and Ant Lion optimization) to detect diseased lung tissues. Finally, Behzadi-khormouji et al. used transfer learning with CNNs pretrained on the ImageNet dataset and combined with a problem-based ChestNet architecture with unnecessary pooling layers removed, and a three-step pre-processing to support model generalization.

Zhang et al. [10] systematically investigated brain signal types and related deep learning concepts for brain signal analysis, and they covered open challenges and future directions for brain signal analysis with help of 230 contributions performed by researchers. Zhou et al. proposed three-stage deep features learning and a fusion framework for identifying Alzheimer disease and its prodromal status. Fan et al. proposed a fully convolutional network deep learning approach for image registration to predict the deformation field which become insensitive to parameter tuning and small hierarchical loss. Liu et al. [9] exploited the usage of a deep convolutional network to explore the uses of local description and feature encoding for image representation and registration. In another study, Shin et al. [10] addressed the problems of thoracic-abdominal lymph detection and interstitial lung disease classification using deep CNN. They developed different CNN architectures and obtained promising results with 85 percent sensitivity at three false positives per patient.

All these studies have performed well on radiological data except that the size of the data was restricted to few hundred samples of patients. Therefore, a detailed study is required to use the power of deep learning over thousand samples of patients to achieve the accurate and reliable predictions. Kallianos et al. [7] presented a state of art review stating the importance of artificial intelligence in chest X-ray image classification and analysis.

3. REQUIREMENT SPECIFICATION

To be used efficiently, all computer software needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements, minimum and recommended.

3.1 SOFTWARE REQUIREMENTS

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software project.

- Operating system : Windows XP/10
- Coding Language : Python
- IDE : Jupyter Notebook, Spyder (Anaconda)
- Cloud Service : Google Colaboratory
- Web Framework : Flask
- Web Development : Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS), Bootstrap

3.2 HARDWARE REQUIREMENTS

A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

- Processor : Intel Core i3-2330M @ 2.20GHz
- Hard Disk : 135 GB
- Monitor : 15 Video Graphics Array (VGA) Colour
- Mouse : Logitech
- RAM : 4 GB
- GPU : NVIDIA Tesla K80

3.3 LIBRARIES

- **Keras**

  Keras is an open-source neural-network library written in Python. It allows for easy and fast prototyping. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It runs seamlessly on Central Processing Unit (CPU) and Graphics Processing Unit (GPU). Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

- **Keras.applications**

  Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning. Weights are downloaded automatically when instantiating a model [16].

- **Keras.models**

  There are two main types of models available in Keras: the Sequential model, and the Model class used with the functional Application programming interface (API). These models have a number of methods and attributes in common:

  - model.layers() is a flattened list of the layers comprising the model.
  - model.inputs() is the list of input tensors of the model.
  - model.outputs() is the list of output tensors of the model.
  - model.summary() prints a summary representation of your model.

- **Keras.optimizers**

  An optimizer is one of the two arguments required for compiling a Keras model. You can either instantiate an optimizer before passing it to model.compile() or you can call it by its name. In the latter case, the default parameters for the optimizer will be used.

- **TensorFlow**

  TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. Its flexible architecture allows for the easy deployment of computation across a variety of platforms and from desktops to clusters of servers to mobile and edge devices.

- **NumPy**

  NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

- **Pandas**

  Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

- **Matplotlib**

  Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose Graphical User Interface (GUI) toolkits.

- **Scikit-learn**

  Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

- **Python Imaging Library (PIL)**

  It is a free library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

4. DESIGN OF THE SYSTEM

It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing techniques are carried out on this data, and the output data is generated by this system. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.

4.1 ARCHITECTURE

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a Rectified Linear Unit (RELU) layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

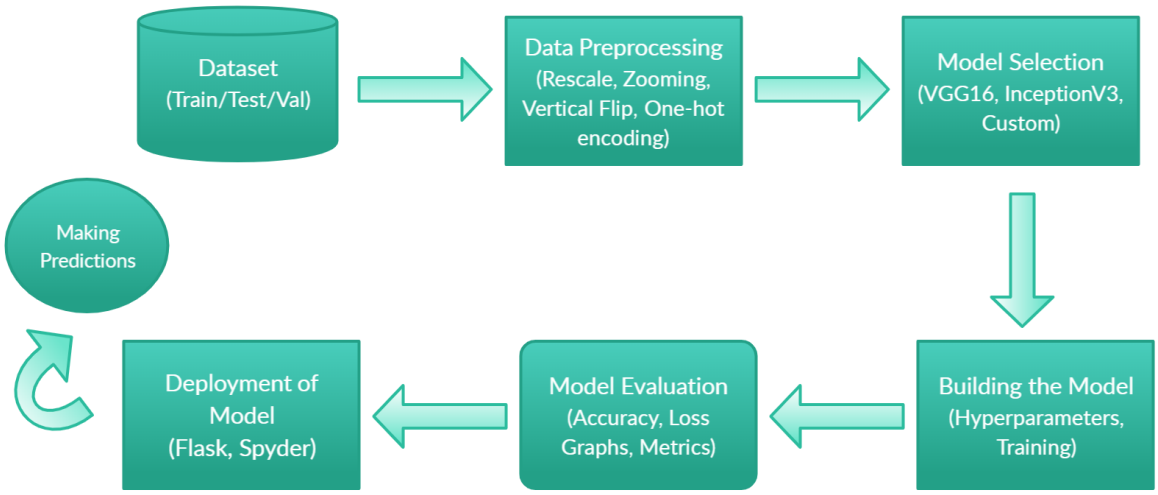

Figure 4.1: System Architecture

4.2 DATASET

In the dataset (Kermany, 2018) used for training and validation, it contained 5863 X-ray images of Joint Photographic Experts Group (JPEG) format were used and contains two categories: Normal and Pneumonia. A total of 5856 patients are present. The radiographic images were from pediatric patients one to five years old from the Medical Center in Guangzhou [2]. In this way, radiographs were performed as part of clinical care. All images in dataset underwent a treatment in order to remove all low-quality scans, as well as being classified by two specialist physicians and by a third party specialist, in order to prevent any misclassification. The images in the dataset are varying resolutions such as 712x439 to 2338x2025. There are 1583 normal case, 4273 pneumonia case images in the dataset. Fig. 4.2 shows some X-ray image samples from the dataset.

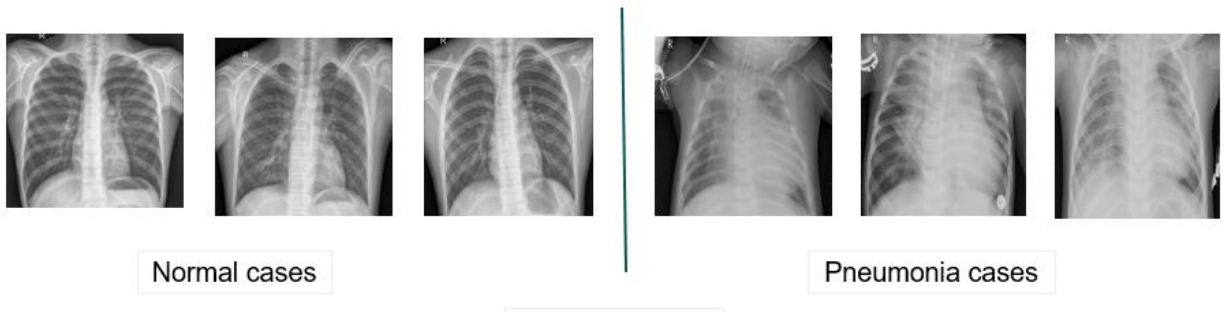

Figure 4.2: Data Samples from the Dataset

Table 4.1 represents the distribution of the data when training, validating and testing phases of the proposed model. In our models, 0 represents normal cases, 1 represents pneumonia cases.

Table 4.1: Distribution of Dataset

Category	Train	Validation	Test
Normal	1341	234	234
Pneumonia	3875	390	390
Total	5216	624	624

4.3 DATA PREPROCESSING

Data preprocessing is a technique which is used to transform the raw data in a useful and efficient format. All the images are from x-ray, which has limited color space, therefore, on the RGB scale, the image does not show difference on the edges or in the parts where certain features might be detected. Furthermore, other augmentation techniques were also used.

4.3.1 Data Augmentation and Transfer Learning

Deep learning needs a huge amount of data to obtain a reliable result. However, there may be not enough data in some problems. Especially on medical problems, to obtain and annotate the data is very expensive and time consuming process. Fortunately, there are some solutions to solve this problem. One of them is data augmentation which avoids the overfitting and improves the accuracy. In this project, training time data augmentation method was utilized. We used different augmentation methods such as shifting, zooming, and flipping. Another performance enhancing method in deep models especially in CNNs which is named as transfer learning.

Transfer learning is the idea of overcoming the isolated learning paradigm and utilizing knowledge acquired for one task to solve related ones. Nowadays, a few people train an entire CNN from scratch. Because it needs a huge amount of data. Instead, using pre-trained CNNs on very large dataset such as ImageNet which is contain 1.2 million image and 1000 class [1]. There are three different transfer learning approaches in CNNs. These are feature extractor, fine-tuning and pre-trained models [14]. First, we resized images to 64×64×3, 150x150x3. But last layers have more data specific features. Therefore, we fixed some early layers of our models and trained our models excluding fixed layers.

4.3.2 One Hot Encoding

Encode categorical features as a one-hot numeric array. The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are encoded using a one-hot (aka ‘one-of-K’ or ‘dummy’) encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array. For categorical variables, it can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

4.4 CNN ARCHITECTURES

Currently, the modern deep learning models in computer vision use convolutional neural networks (CNNs). These layers make the explicit assumption that any input to them is an image. Early convolutional layers in a network process an image and come up with detecting low-level features in images like edges. These layers are successful in capturing the spatial and temporal dependencies in an image. This is done with the help of filters. Unlike normal feed-forward layers, these layers have a much lower number of parameters and use a weight-sharing technique, thus reducing computation efforts. The learnable parameters of each layer consist of filters, extended through the full depth of the input volume but these have a small receptive field. When an input is subjected to forward pass, each kernel is convolved across the height and width of the input volume, creating a 2-D activation map of that filter. If 'N' filters are used, then stacking those 'N' activation maps along the depth forms the full output of the convolutional layer as in Figure. 4.3.

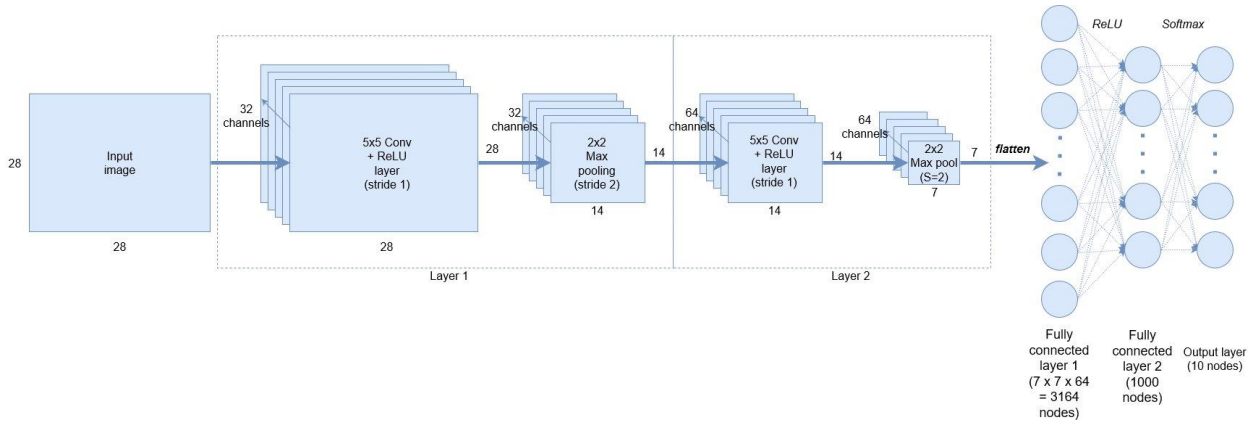

Figure 4.3: Filter Operation over Convolutional Layers

The activation layer is very useful as it helps to approximate almost any nonlinear function. The feature map from the convolutional layer is taken as input to the activation layer. Pooling layers are used to reduce the spatial size of representation generated by previous kernels after convolution. This helps in reducing the number of parameters, thus reducing the computation work. These layers are used to extract dominant features that are positional and rotational invariant. It is common practice to include a pooling layer in between two convolutional layers. The most common pooling layer is the max pooling layer; it separates input into squares of a given size, and

outputs the maximum value of each square. On the other hand, an average pooling layer finds the average of each square, both the methods reduce the dimensionality and computation efforts [15].

In the first stage we have the convolutional layer, which is one of the main layers, where it is the extraction of characteristics of the image by training structure and validation of filters. A filter is an array of values, called weights, trained to detect specific features. The filter moves over each part of the image to see if the feature that it should detect is present. When the feature is present in part of an image, the convolution operation between the filter and that part of the image results in a real number with a high value. If the resource is not present, the resulting value will be low. To provide a value that represents the confidence that a particular resource is present, the filter performs a convolution operation, which is an elementary product and sums between two arrays. Given a two-dimensional image, I, and a small array, K of size h x w (kernel), the convoked image, I ∗K, is calculated by superimposing the kernel at the top of the image all possible forms, and recording the sum of the elementary products between the image and the kernel equation.

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} . I_{x+i-1,y+j-1}$$

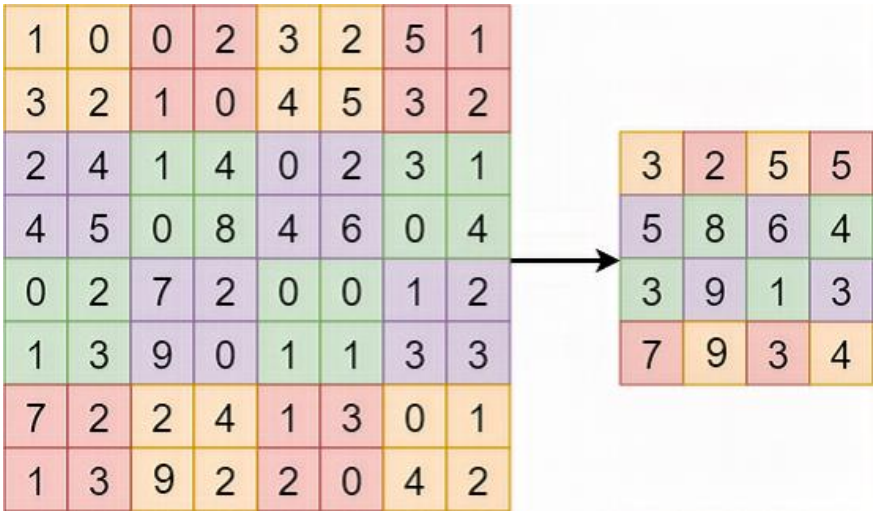

Figure 4.4: Max-pooling with 8x8 Image

After the convolution layer is the pooling layer, which is responsible for reducing the spatial size of the feature map, preserving the resources detected in a smaller representation. There are several

pooling layer options, with maxpooling being the most popular. Basically, maxpooling operates by locating the locations in the image that show the strongest correlation for each resource (the maximum value) are preserved and those maximum values combine to form a smaller space as shown in Figure 4.4. In general, they are used after the convolutional layers, both with the objective of progressively reducing computational costs in the network, as well as minimizing the probability of overfitting. After the pooling layer, we find the fully connected layers (FCN) that are used to make final predictions [16]. A FCN layer obtains resources in a vector form from a previous resource extraction layer, multiplies a weight matrix, and generates a new resource vector whose computation pattern is a dense matrix vector multiplication. Some FCNs are used in a cascade mode that ultimately produce the CNN classification result that generates a probability (a number ranging from 0 to 1) for each of the classification labels that the model is trying to predict.

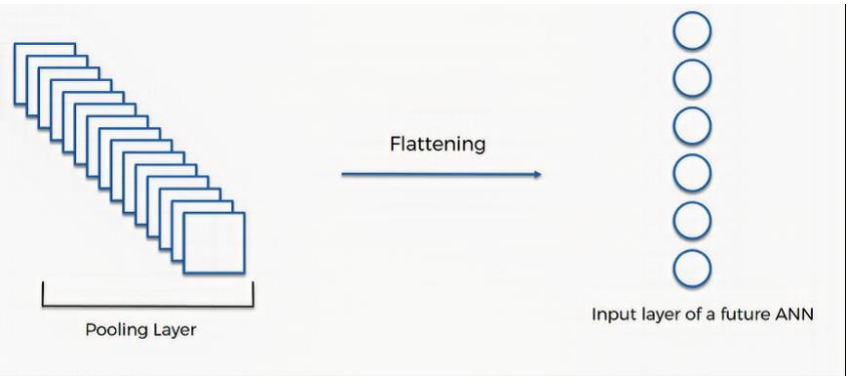

Figure 4.5: Flattening after the Convolutional Steps

Now, the current output is flattened, converted to one long vector, which will be crucial for the classification algorithms. Flatting is applied for converting the data to a more manageable version within the classification algorithm, artificial neural network, as it intakes the flattened data as input. After this point, the network obtained the feature map of the input value, which will proceed with a regular feed forward back propagation neural network.

4.5 METHODOLOGY

A study of the literature reveals several works pertaining to the use of handcrafted features for detecting pneumonia in chest radiographs. However, few studies reported the performance of deep learning methods applied to pneumonia detection in pediatric CXRs. Relatively few researchers attempted to offer a qualitative explanation of their model's learned behavior, internal computations, and predictions [1].

The proposed project uses two CNN architectures called as Visual Geometry Group (Vgg16) and Inceptionv3 based on transfer learning. There is also a custom model which has many layers that ultimately contributes to a deeper network has been developed from scratch using various optimizing features. The models are then trained and the best model is chosen based on the performance. The model is then saved and deployed using a web framework called as flask and Integrated Development Environment (IDE) called Spyder for making predictions.

Setup: In this project, all the data is image formatted, so we need to process the images to identify the patterns and based on that patterns need to classify the different classes. To perform all these tasks, we need a proper hardware system which can handle image processing task without system failure. Also, the requirement of proper GPU is must to train the model within minimum time.

Python Programming Language has been used in this project, but due to lack of hardware processing power, normal laptops or systems might be not able to handle the model building process smoothly. To overcome this issue, we used cloud service-based platform named Google Colaboratory provided by Google. It is a free Jupyter notebook environment that requires no prior setup and runs entirely on cloud. In this environment 12 GB of ram and 50 GB of storage space is provided initially as freemium service, also Google provides free NVIDIA Tesla K80 Graphical processor of about 12 GB for fast computing. To perform this study data is uploaded on the Google drive for maximum time availability. And all the task which are mentioned in the next section is performed on Google Colab notebook.

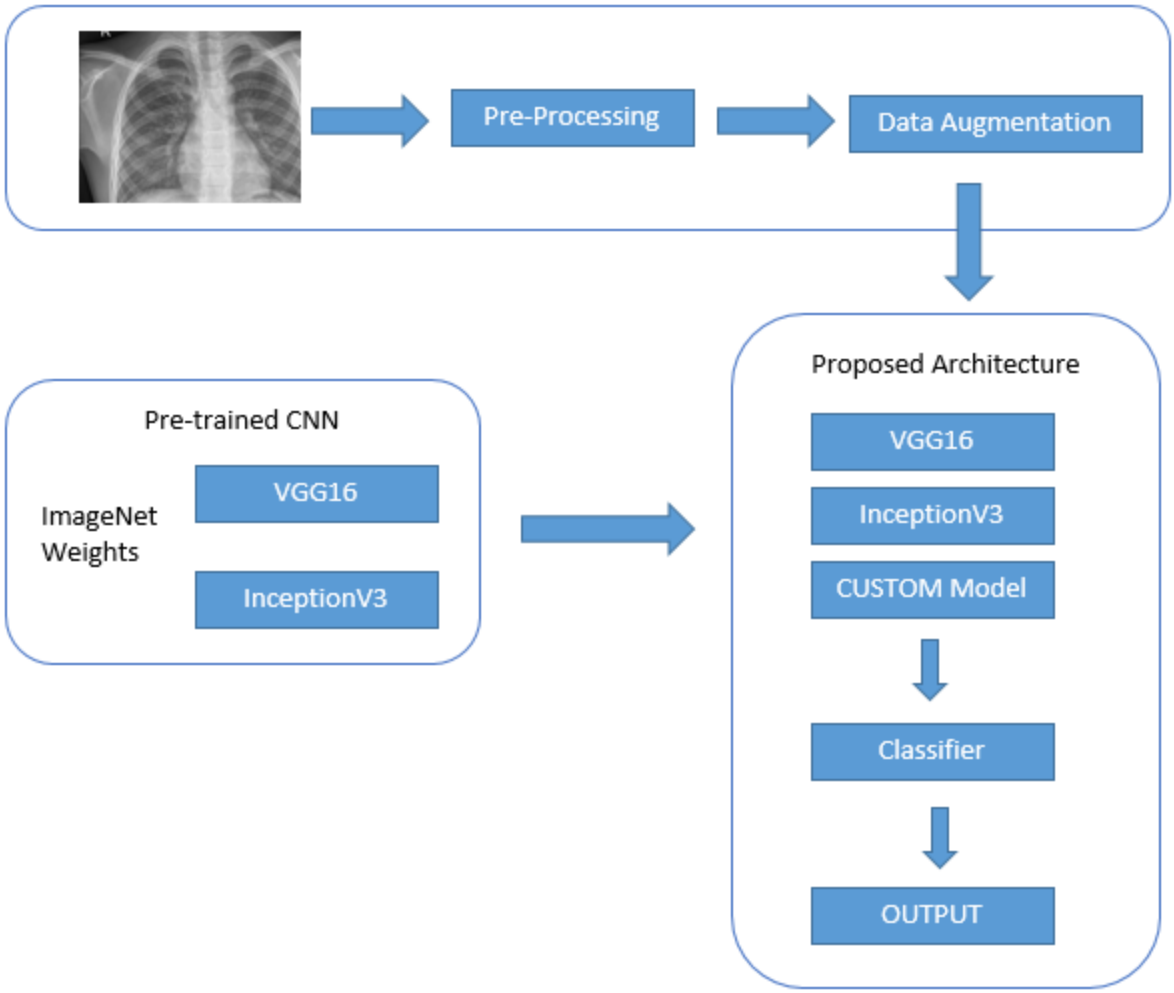

Figure 4.6: Outline of the Methodology

Our methodology is summarized in Figure 4.6. It includes the following steps: chest X-ray image preprocessing, data augmentation, transfer learning using Vgg16, InceptionV3 convolutional neural networks and a separate custom neural network, feature extraction and ensemble classification. The models which have been used are explained in more detail in the subsequent subsections.

5. MODELS

We employed two different pre-trained models, Vgg16 and InceptionV3 which were already trained on the ImageNet dataset and then used them on images from the Chest X-ray dataset. Apart from these two models, we used a custom model where the layers are built from scratch by adding dropout and batch normalization.

5.1 VGG16 MODEL

VGG16 is a convolution neural net (CNN) architecture which was used to win ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition in 2014. It is considered to be one of the excellent vision model architecture till date. Most unique thing about Visual Geometry Group (VGG) is that instead of having a large number of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has fully connected layers followed by a softmax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about approximately 138 million parameters [15].

Unfortunately, there are two major drawbacks with VGGNet:

1. It is painfully slow to train.
2. The network architecture weights themselves are quite large (concerning disk/bandwidth).

Due to its depth and number of fully-connected nodes, VGG16 is over 533MB. This makes deploying VGG a tiresome task.VGG16 is used in many deep learning image classification problems. However, smaller network architectures are often more desirable. But it is a great building block for learning purpose as it is easy to implement.

Table 5.1: Vgg16 Training Accuracy Data

Training Accuracy	Training Loss
0.9393417527745886	0.7712413567493145

Figure 5.1: Vgg16 Network

5.1.1 Root Mean Square Propagation (RMSprop) Optimizer

The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster.

5.1.2 Softmax Activation Function

The softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function. But it also divides each output such that the total sum of the outputs is equal to 1.

The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true.

Mathematically the softmax function is shown below, where z is a vector of the inputs to the output layer (if you have 10 output units, then there are 10 elements in z). And again, j indexes the output units, so j = 1, 2... K.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

5.1.3 Dense Layer

Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output.

Output = activation (dot (input, kernel) + bias)

Where,

- input represent the input data
- kernel represent the weight data
- dot represent numpy dot product of all input and its corresponding weights
- bias represent a biased value used in machine learning to optimize the model
- activation represent the activation function.

5.2 INCEPTIONV3 MODEL

Inceptionv3 is a convolutional neural network for assisting in image analysis and object detection, and got its start as a module for Googlenet. It is the third edition of Google's Inception Convolutional Neural Network, originally introduced during the ImageNet Recognition Challenge. Just as ImageNet can be thought of as a database of classified visual objects, Inception helps classification of objects in the world of computer vision.

The InceptionV3 model allows for increasing the depth and width of the deep learning network, but maintaining the computational cost constant at the same time. This model was trained on the original ImageNet dataset with over 1 million training images. It works as a multi-level feature generator by computing 1 × 1, 3 × 3 and 5 × 5 convolutions [13].

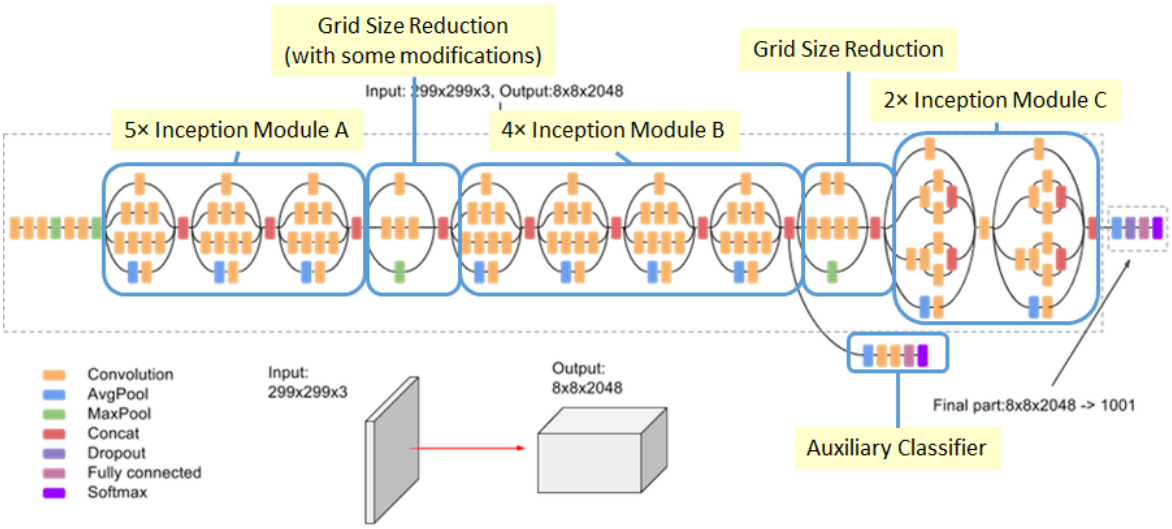

Figure 5.2: Inceptionv3 Network

This allows the model to use all kinds of kernels on the image and to get results from all of those. All such outputs are stacked along the channel dimension, and used as input to the next layer. This model achieved top performance for computer vision tasks, by using some advanced techniques. The architecture we use is shown in Figure 5.2.

Table 5.2: Inceptionv3 Training Accuracy Data

Training Accuracy	Training Loss
97.53971%	6.87396%

5.2.1 ADAM Optimizer

Adam is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself. It computes individual learning rates for different parameters. Its name is derived from adaptive moment estimation, and the reason it's called that is because Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network. N-th moment of a random variable is defined as the expected value of that variable to the power of n. More formally:

$$m_n = E[X^n]$$

Where, m - moment, X -random variable.

5.2.2 SIGMOID Activation Function

A sigmoid function is a type of activation function, and more specifically defined as a squashing function. Squashing functions limit the output to a range between 0 and 1, making these functions useful in the prediction of probabilities. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice. The function is differentiable. That means, we can find the slope of the sigmoid curve at any two points.

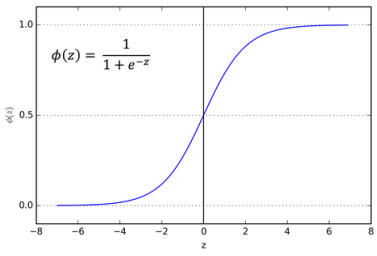

Figure 5.3: Sigmoid Function

5.3 CUSTOM MODEL

In this model, after the libraries required are imported, labels are created for all the images. Then data augmentation is performed where training set and test set images are fed to the network. Data augmentation is an effective way to increase the size of the training set. Augmenting the training images allow the network to see more diversified, but still representative, data points during training. Then, a couple of data generators are defined: one for training data, and the other for validation data. A data generator is capable of loading the required amount of data like a mini batch of images directly from the source folder, convert them into training data and training targets.

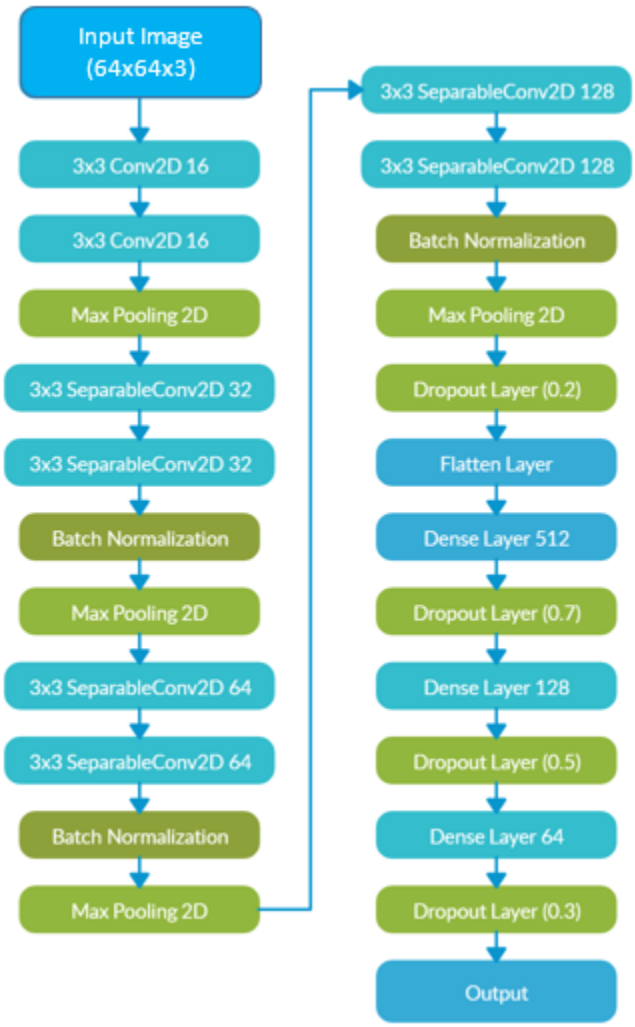

Figure 5.4: Custom CNN Architecture

The next step was to build the model. This can be described in the following 5 steps. Five convolutional blocks comprised of convolutional layer, max-pooling and batch-normalization. On top of it, a flatten layer and followed it by four fully connected layers. Also in between dropouts have been used to reduce over-fitting [4]. Activation function was Rectified Linear Unit (RELU) throughout except for the last layer where it was Sigmoid as this is a binary classification problem. Adam as the optimizer and cross-entropy as the loss are used. Before training the model, it is useful to define one or more callbacks. They are:

ModelCheckpoint: When training requires a lot of time to achieve a good result, often many iterations are required. In this case, it is better to save a copy of the best performing model only when an epoch that improves the metrics ends.

EarlyStopping: Sometimes, during training we can notice that the generalization gap (i.e. the difference between training and validation error) starts to increase, instead of decreasing. This is a symptom of overfitting that can be solved in many ways like reducing model capacity, increasing training data, data augmentation, regularization, and dropout. Often a practical and efficient solution is to stop training when the generalization gap is getting worse.

Table 5.3: Custom Model Training Accuracy Data

Training Accuracy	Training Loss
95.15564%	13.82808%

5.3.1 ReLU (Rectified Linear Unit) Activation Function

ReLU stands for rectified linear unit, and is a type of activation function. Mathematically, it is defined as $y = max (0, x)$. Visually, it looks like the following:

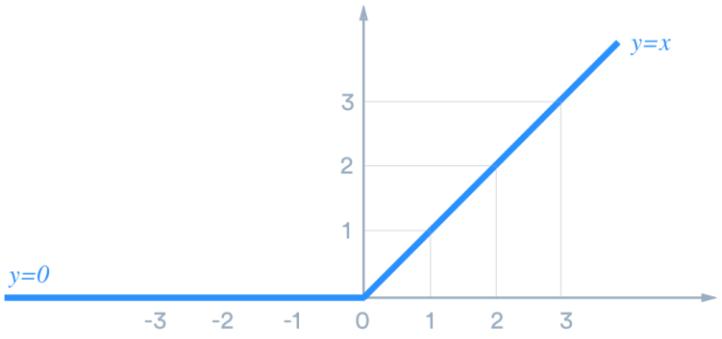

Figure 5.5: Relu Function

5.3.2 Separable Convolution Layer

Separable convolutions consist in first performing a depthwise spatial convolution which acts on each input channel separately followed by a pointwise convolution which mixes together the resulting output channels. Intuitively, separable convolutions can be understood as a way to factorize a convolution kernel into two smaller kernels, or as an extreme version of an Inception block [12].

Depthwise Convolution: In depth-wise operation, convolution is applied to a single channel at a time unlike standard CNN's in which it is done for all the M channels. So here the filters/kernels will be of size Dk x Dk x 1

Pointwise Convolution: The pointwise convolution is so named because it uses a 1x1 kernel, or a kernel that iterates through every single point. This kernel has a depth of however many channels same as the input image.

These type of CNN's are widely used because of the following two reasons –

- They have lesser number of parameters to adjust as compared to the standard CNN's, which reduces overfitting.
- They are computationally cheaper because of fewer computations which makes them suitable for mobile vision applications.

5.3.3 Batch Normalization

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. We can use higher learning rates because batch normalization makes sure that there's no activation that's gone really high or really low. And by that, images that previously couldn't get to train, it will start to train. It reduces overfitting because it has a slight regularization effects. To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

6. IMPLEMENTATION

6.1 Vgg16 Model

```
import os
import glob
import urllib.request
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.data import Dataset, Iterator

from google.colab import drive
drive.mount('/content/drive')
def get_labeled_files(folder):
    x = []
    y = []
    for folderName in os.listdir(folder):
        if not folderName.startswith('/'):
            if folderName in ['NORMAL']:
                label = 0
            elif folderName in ['PNEUMONIA']:
                label = 1
            else:
                label = 2
                continue
            for image_filename in os.listdir(folder + folderName):
                img_path = folder + folderName + '/' + image_filename
                if img_path is not None and str.endswith(img_path, 'jpeg'):
                    x.append(img_path)
```

```
                y.append(label)
    x = np.asarray(x)
    y = np.asarray(y)
    return x, y
x, y = get_labeled_files('/content/drive/My Drive/chest_xray/train/')
print(x,y)
NUM_CLASSES = 2
# This function takes image paths as arguments and reads corresponding images
def input_parser(img_path, label):
    # convert the label to one-hot encoding
    one_hot = tf.one_hot(label, NUM_CLASSES)
    # read the img from file and decode it using tf
    img_file = tf.io.read_file(img_path)
    img_decoded = tf.image.decode_jpeg(img_file, channels=3, name="decoded_images")
    return img_decoded, one_hot
# This function takes image and resizes it to smaller format (150x150)
def image_resize(images, labels):
    resized_image = tf.image.resize(images, (150, 150), align_corners=True)
    resized_image_asint = tf.cast(resized_image, tf.int32)
    return resized_image_asint, labels
# Since it uses lazy evaluation, images will not be read after calling build_pipeline_plan()
# Using iterator defined here in tf context
def build_pipeline_plan(img_paths, labels, batch_size):
    # Build a tensor of image paths and labels
    tr_data = Dataset.from_tensor_slices((img_paths, labels))
    # Read images in paths as jpegs
    tr_data_imgs = tr_data.map(input_parser)
    # Apply resize to each image in the pipeline
```

```
    tr_data_imgs = tr_data_imgs.map(image_resize)
    # batch images into small groups
    tr_dataset = tr_data_imgs.batch(batch_size)
    # create TensorFlow Iterator object directly from input pipeline
    iterator = tf.compat.v1.data.make_one_shot_iterator(tr_dataset)
    next_element = iterator.get_next()
    return next_element
# Function to execute defined pipeline in Tensorflow session
def process_pipeline(next_element):
    gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.333)
    with tf.Session(config=tf.ConfigProto(gpu_options=gpu_options)) as sess:
        images = []
        labels_hot = []
        while True:
            try:
                elem = sess.run(next_element)
                images = elem[0]
                labels_hot = elem[1]
            except tf.errors.OutOfRangeError:
                print("Finished reading the dataset")
                return images, labels_hot
def load_dataset(path, batch_size):
    tf.reset_default_graph()
    files, labels = get_labeled_files(path)
    p = tf.constant(files, name="train_imgs")
    l = tf.constant(labels, name="train_labels")
    next_element = build_pipeline_plan(p, l, batch_size=batch_size)
    imgs, labels = process_pipeline(next_element)
```

```
    return imgs, labels
x_train, y_train = load_dataset("/content/drive/My Drive/chest_xray/train/",6000)
x_test, y_test = load_dataset("/content/drive/My Drive/chest_xray/test/",6000)
x_val, y_val = load_dataset("/content/drive/My Drive/chest_xray/val/",6000)

#Building a Model
import keras
from keras import backend as K
from keras.models import Model
from keras.layers import Flatten, Dense, BatchNormalization, Dropout
from keras.applications.vgg16 import VGG16
K.clear_session()
NUM_CLASSES = 2
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
x = base_model.output
x = Flatten()(x)
x = Dense(NUM_CLASSES, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=x)
model.summary()
def print_layers(model):
    for idx, layer in enumerate(model.layers):
        print("layer {}: {}, trainable: {}".format(idx, layer.name, layer.trainable))
for layer in model.layers[0:20]:
    layer.trainable = False
print_layers(model)
model.trainable_weights
optimizer = keras.optimizers.RMSprop()
model.compile(loss='categorical_crossentropy',
```

```
        optimizer=optimizer,
        metrics=['accuracy'])
from keras.callbacks import ModelCheckpoint, TensorBoard, ReduceLROnPlateau,
EarlyStopping
# This callback saves weights of model after each epoch
checkpoint = ModelCheckpoint(
    'model/weights.epoch_{epoch:02d}.hdf5',
    monitor='val_loss',
    save_best_only=False,
    save_weights_only=False,
    mode='auto',
    verbose=1
)
from sklearn.utils import class_weight
y_labels = np.argmax(y_train, axis=1)
classweight = class_weight.compute_class_weight('balanced', np.unique(y_labels), y_labels)
print(classweight)
# prepare directory to store the model weights
os.makedirs('./model', exist_ok=True)
history = model.fit(
    x=x_train, y=y_train,
    class_weight=classweight,
    validation_split=0.3,
    callbacks=[checkpoint, tensorboard],
    shuffle=True,
    batch_size=64,
    epochs=30,
    verbose=1
)
```

6.2 InceptionV3 Model

```
import numpy as np
import matplotlib.pyplot as plt
import os
import pandas as pd
import math
from keras.applications import InceptionV3
from keras.models import load_model
from keras.applications.inception_v3 import preprocess_input as incep_preprocess_input
from keras.callbacks import ModelCheckpoint
from keras.layers import Flatten, Dense, BatchNormalization, Dropout
import tensorflow as tf

num_of_classes = 2
batch_size = 128
target_size = (299, 299)
print("Using Inception v3")
base_model = InceptionV3(weights='imagenet', input_shape=(299, 299, 3), include_top=False)

x = base_model.output
x = Flatten()(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.33)(x)
x = BatchNormalization()(x)
output = Dense(1, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=output)
for layer in base_model.layers:
    layer.trainable = False
```

```
opt = Adam(lr=0.0003)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
model.summary()
#Load first stage weights to fine tune off off
model.load_weights("/content/best_weights.hdf5")
train_datagen = ImageDataGenerator(preprocessing_function=incep_preprocess_input,
                                   shear_range=0.2, zoom_range=0.2,
                                   horizontal_flip=True, fill_mode='nearest')
train_generator = train_datagen.flow_from_directory('/content/drive/My Drive/chest_xray/train',
                                        target_size=target_size, color_mode='rgb',
                                        batch_size=batch_size, class_mode='binary',
                                        shuffle=True, seed=42)
val_datagen = ImageDataGenerator(preprocessing_function=incep_preprocess_input)
val_generator = val_datagen.flow_from_directory('/content/drive/My Drive/chest_xray/test',
                                      target_size=target_size, color_mode="rgb",
                                      batch_size=batch_size, shuffle=False, class_mode="binary")
step_size_train = train_generator.n // train_generator.batch_size
step_size_valid = val_generator.n // val_generator.batch_size
history = model.fit_generator(generator=train_generator,
                    steps_per_epoch=step_size_train,
                    validation_data=val_generator,
                    validation_steps=step_size_valid,
                    callbacks=[checkpoint],
                    epochs=30, verbose=1)


chkpt1 = ModelCheckpoint(filepath="ft1_inc_best_model_acc.hd5", monitor='accuracy',
verbose=1, save_best_only=True, save_weights_only=False)
chkpt2 = ModelCheckpoint(filepath="ft1_inc_best_model_loss.hd5", monitor='loss', verbose=1,
save_best_only=True, save_weights_only=False)
```

```
chkpt3 = ModelCheckpoint(filepath="ft1_inc_best_model_val_loss.hd5", monitor='val_loss', verbose=1, save_best_only=True, save_weights_only=False)

#Sample Predictions

from PIL import Image

def testImage(model, file, title, width=299, height=299, color=True):
    image = Image.open(file)
    if color:
        image = image.convert("RGB")
    plt.imshow(np.asarray(image))
    data = np.array(image.resize((width,height), Image.ANTIALIAS)).reshape(1, width, height, 3 if color else 1)/299
    plt.title(title)
    plt.xlabel("Prediction: " + ("PNEUMONIA" if model.predict(data) > 0.5 else "NORMAL"))
    plt.show()


testImage(
    model = model,
    file = '/content/drive/My Drive/chest_xray/test/PNEUMONIA/person100_bacteria_475.jpeg',
    title = "Pneumonia sample - Custom CNN"
)
testImage(
    model = model,
    file = '/content/drive/My Drive/chest_xray/test/NORMAL/IM-0001-0001.jpeg',
    title = "Normal sample - Custom CNN"
)
```

6.3 Custom Model

```
# Deep learning libraries
import keras.backend as K
from keras.models import Model, Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, SeparableConv2D, MaxPool2D, LeakyReLU, Activation
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
import tensorflow as tf


input_path = '/content/drive/My Drive/chest_xray'
fig, ax = plt.subplots(2, 3, figsize=(15, 7))
ax = ax.ravel()
plt.tight_layout()
for i, _set in enumerate(['/train', '/val', '/test']):
    set_path = input_path+_set
    ax[i].imshow(plt.imread(set_path+'/NORMAL/'+os.listdir(set_path+'/NORMAL')[0]),
cmap='gray')
    ax[i].set_title('Set: {}, Condition: Normal'.format(_set))

ax[i+3].imshow(plt.imread(set_path+'/PNEUMONIA/'+os.listdir(set_path+'/PNEUMONIA')[0]),
cmap='gray')
    ax[i+3].set_title('Set: {}, Condition: Pneumonia'.format(_set))
# Distribution of dataset
for _set in ['/train', '/val', '/test']:
    n_normal = len(os.listdir(input_path + _set + '/NORMAL'))
    n_infect = len(os.listdir(input_path + _set + '/PNEUMONIA'))
    print('Set: {}, normal images: {}, pneumonia images: {}'.format(_set, n_normal, n_infect))
```

```
input_path = '/content/drive/My Drive/chest_xray/'
def process_data(img_dims, batch_size):
    # Data generation objects
    train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, vertical_flip=True)
    test_val_datagen = ImageDataGenerator(rescale=1./255)
    # This is fed to the network in specified batch sizes and image dimensions
    train_gen = train_datagen.flow_from_directory(
    directory=input_path+'train',
    target_size=(img_dims, img_dims),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True)
    test_gen = test_val_datagen.flow_from_directory(
    directory=input_path+'test',
    target_size=(img_dims, img_dims),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True)
    test_data = []
    test_labels = []
    for cond in ['/NORMAL/', '/PNEUMONIA/']:
        for img in (os.listdir(input_path + 'test' + cond)):
            img = plt.imread(input_path+'test'+cond+img)
            img = cv2.resize(img, (img_dims, img_dims))
            img = np.dstack([img, img, img])
            img = img.astype('float32') / 255
            if cond=='/NORMAL/':
                label = 0
```

```
            elif cond=='/PNEUMONIA/':
                label = 1
            test_data.append(img)
            test_labels.append(label)
    test_data = np.array(test_data)
    test_labels = np.array(test_labels)
    return train_gen, test_gen, test_data, test_labels
# Hyperparameters
img_dims = 64
epochs = 35
batch_size = 64
# Getting the data
train_gen, test_gen, test_data, test_labels = process_data(img_dims, batch_size)
#Building the model
inputs = Input(shape=(img_dims, img_dims, 3))
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(inputs)
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
```

```
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.2)(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.2)(x)
x = Flatten()(x)
x = Dense(units=512, activation='relu')(x)
x = Dropout(rate=0.7)(x)
x = Dense(units=128, activation='relu')(x)
x = Dropout(rate=0.5)(x)
x = Dense(units=64, activation='relu')(x)
x = Dropout(rate=0.3)(x)
output = Dense(units=1, activation='sigmoid')(x)
# Creating model and compiling
model = Model(inputs=inputs, outputs=output)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Callbacks
checkpoint = ModelCheckpoint(filepath='best_weights.hdf5', save_best_only=True,
save_weights_only=True)
lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=2, verbose=2,
mode='max')
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=1, mode='min')
# Fitting the model
hist = model.fit_generator(
        train_gen, steps_per_epoch=train_gen.samples // batch_size,
        epochs=epochs, validation_data=test_gen,
```

```
        validation_steps=test_gen.samples // batch_size, callbacks=[checkpoint, lr_reduce])
loaded_model = tf.keras.models.load_model('/content/model4.h5')
loaded_model.layers[0].input_shape
model.summary()

#Sample Predictions
from PIL import Image
def testImage(model, file, title, width=64, height=64, color=True):
    image = Image.open(file)
    if color:
        image = image.convert("RGB")
    plt.imshow(np.asarray(image))
    data = np.array(image.resize((width,height), Image.ANTIALIAS)).reshape(1, width, height, 3
if color else 1)/255
    plt.title(title)
    plt.xlabel("Prediction: " + ("PNEUMONIA" if model.predict(data) > 0.5 else "NORMAL"))
    plt.show()
testImage(
    model = model,
    file = '/content/drive/My Drive/chest_xray/test/PNEUMONIA/person100_bacteria_475.jpeg',
    title = "Pneumonia sample - Custom CNN"
)
testImage(
    model = model,
    file = '/content/drive/My Drive/chest_xray/test/NORMAL/IM-0001-0001.jpeg',
    title = "Normal sample - Custom CNN"
)
```

7. TESTING

To evaluate and validate the effectiveness of the proposed approach, we conducted the experiments 10 times each for three hours, respectively. Parameter and hyperparameters were heavily turned to increase the performance of the model. Different results were obtained, but this project reports only the most valid.

7.1 METRICS OF EVALUATION

Evaluation metrics are used to measure the quality of the statistical or machine learning model. Evaluating machine learning models or algorithms is essential. There are many different types of evaluation metrics available to test a model. These include classification accuracy, logarithmic loss, confusion matrix, and others [1].

7.1.1 Training history visualization

The fit() method on a Keras Model returns a History object. The History.history attribute is a dictionary recording training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values (if applicable). Here are the models using matplotlib to generate loss & accuracy plots for training & validation:

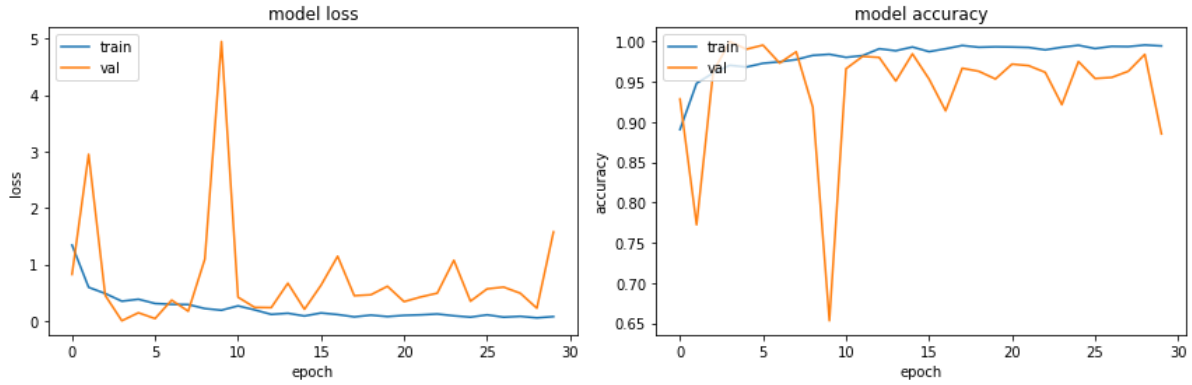

Figure 7.1: Vgg16 Model Loss and Accuracy Graphs

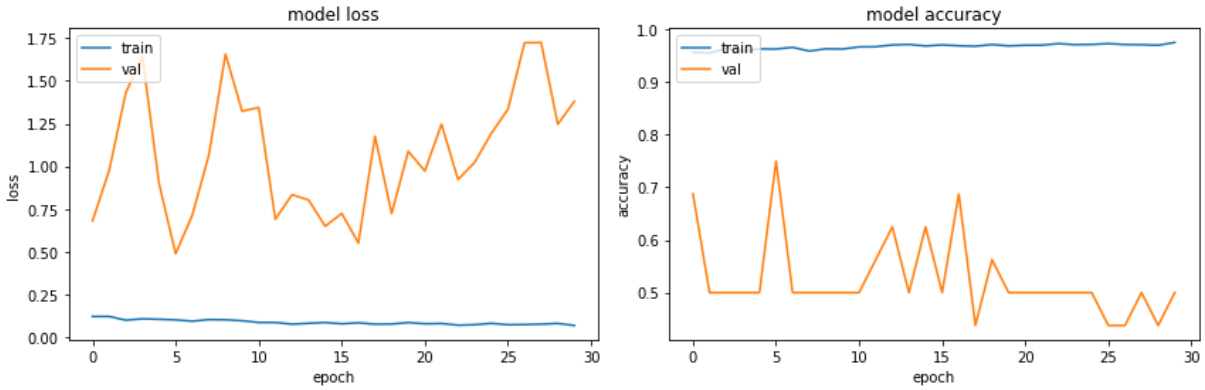

Figure 7.2: Inceptionv3 Model Loss and Accuracy Graphs

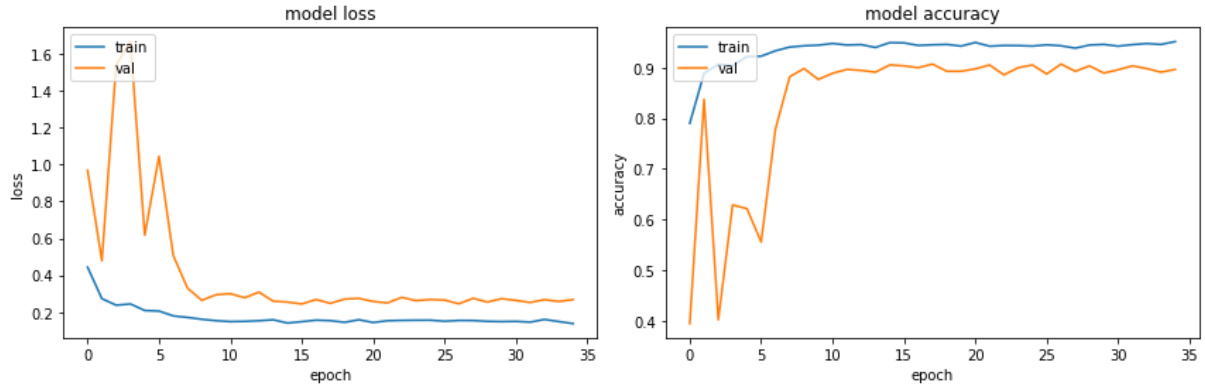

Figure 7.3: Custom Model Loss and Accuracy Graphs

7.1.2 Confusion Matrix

The confusion matrix is an array that contains correct and incorrect predictions of the algorithm and the actual situation as shown in table 7.1.

True Positive: Number of people who actually have pneumonia according to the algorithm.

False Negative: Number of people who are actually with pneumonia but categorized as healthy according to the algorithm.

Table 7.1: Confusion Matrix

Category	**Positive Predicted**	**Negative Predicted**
Positive	True Positive	False Negative
Negative	False Positive	True Negative

False Positive: Number of people who are actually healthy, but categorized as pneumonia, according to the algorithm.

True Negative: Number of people who are really healthy and categorized as healthy according to the algorithm.

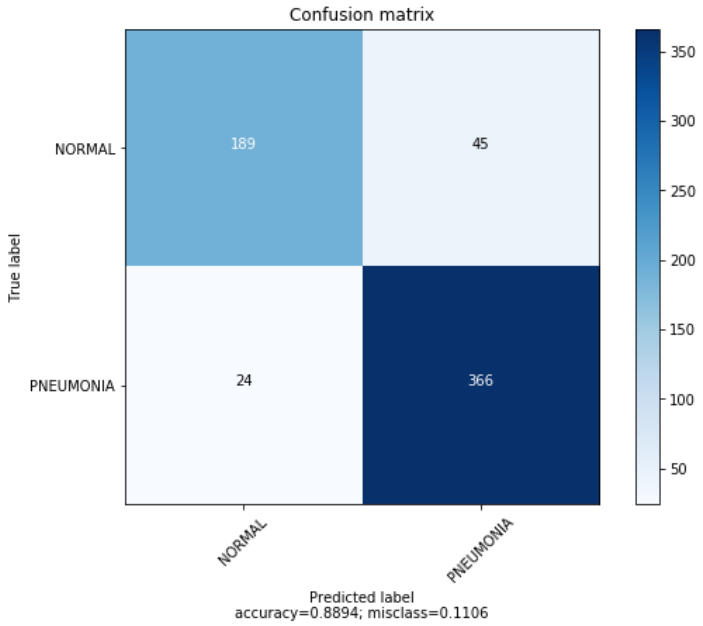

Figure 7.4: Vgg16 Model Confusion Matrix

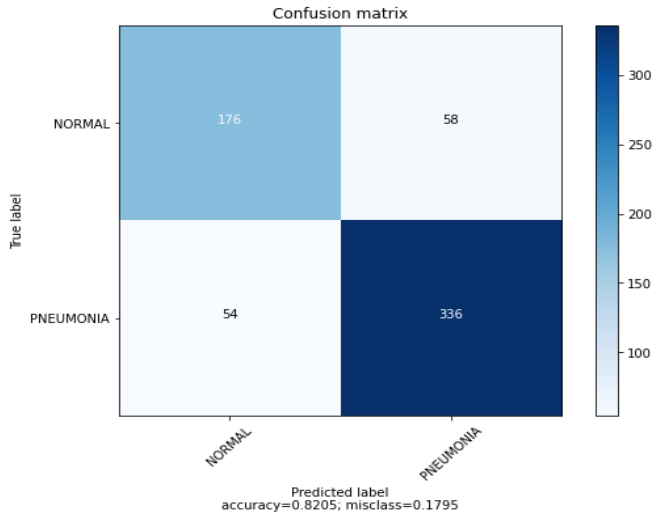

Figure 7.5: Inceptionv3 Model Confusion Matrix

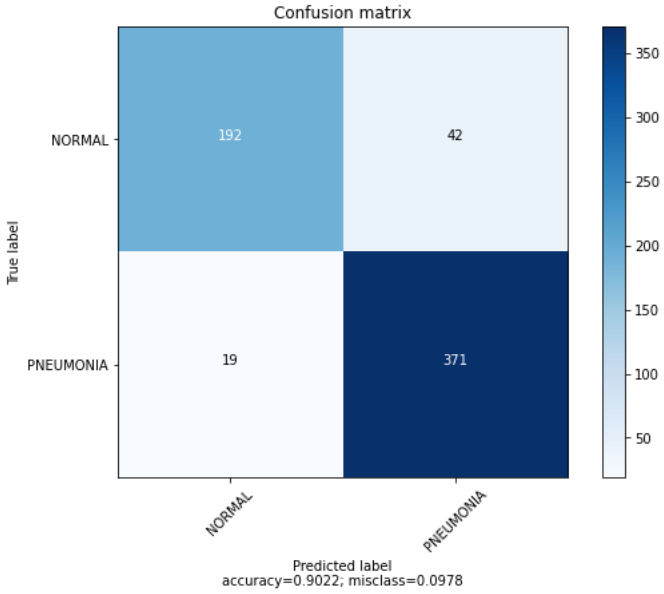

Figure 7.6: Custom Model Confusion Matrix

7.1.3 Precision

It is also called Positive predictive value. The ratio of correct positive predictions to the total predicted positives.

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

7.1.4 Recall

It is also called Sensitivity, Probability of Detection, and True Positive Rate. Ratio of correct positive predictions to the total positives examples.

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

7.1.5 F1 Score

It is also called the F Score or the F Measure. In other words, the F1 score conveys the balance between the precision and the recall.

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

Table 7.2: Comparison Chart of the CNN Architectures Evaluation Metrics

Model	Test Accuracy	Precision	Recall	F1 Score
Vgg16	88.94230%	89.05109%	93.84615%	91.38576%
InceptionV3	82.05128%	85.27918 %	86.15384 %	85.71428 %
Custom	90.22435%	89.83050%	95.12820%	92.40348%

7.2 SAMPLE PREDICTIONS

The following are the sample predictions made for each model when testing.

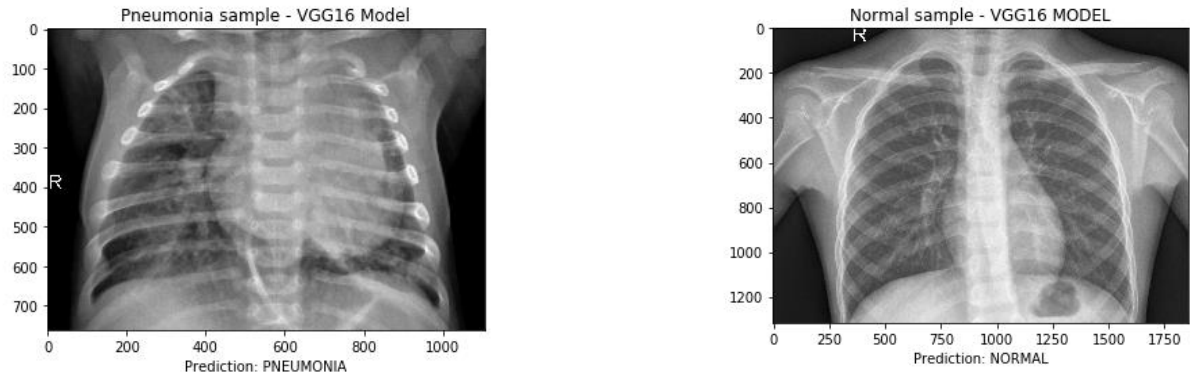

Figure 7.7: Vgg16 Model Predictions

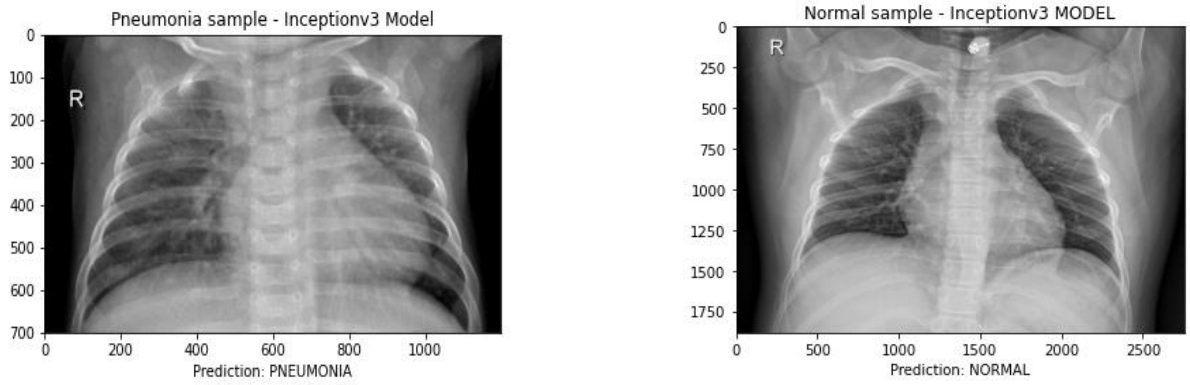

Figure 7.8: Inceptionv3 Model Predictions

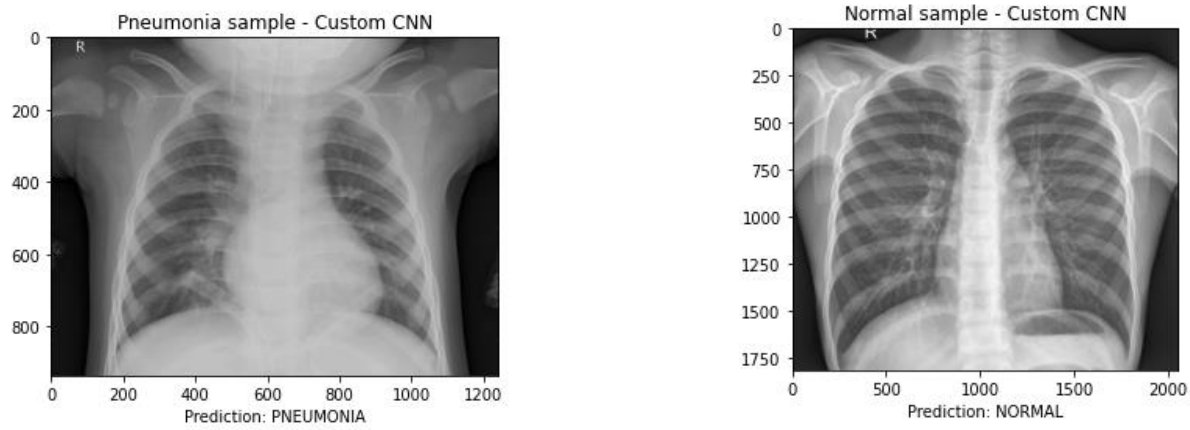

Figure 7.9: Custom Model Predictions

7.3 DEPLOYMENT OF THE MODEL

Deployment of deep learning models or putting models into production means making your models available to the end users or systems. Flask is a micro web framework written in Python. It is classified as a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions [6].

After running the python program in Spyder IDE, open http://127.0.0.1:5000/ in your web-browser, and the GUI as shown below should appear.

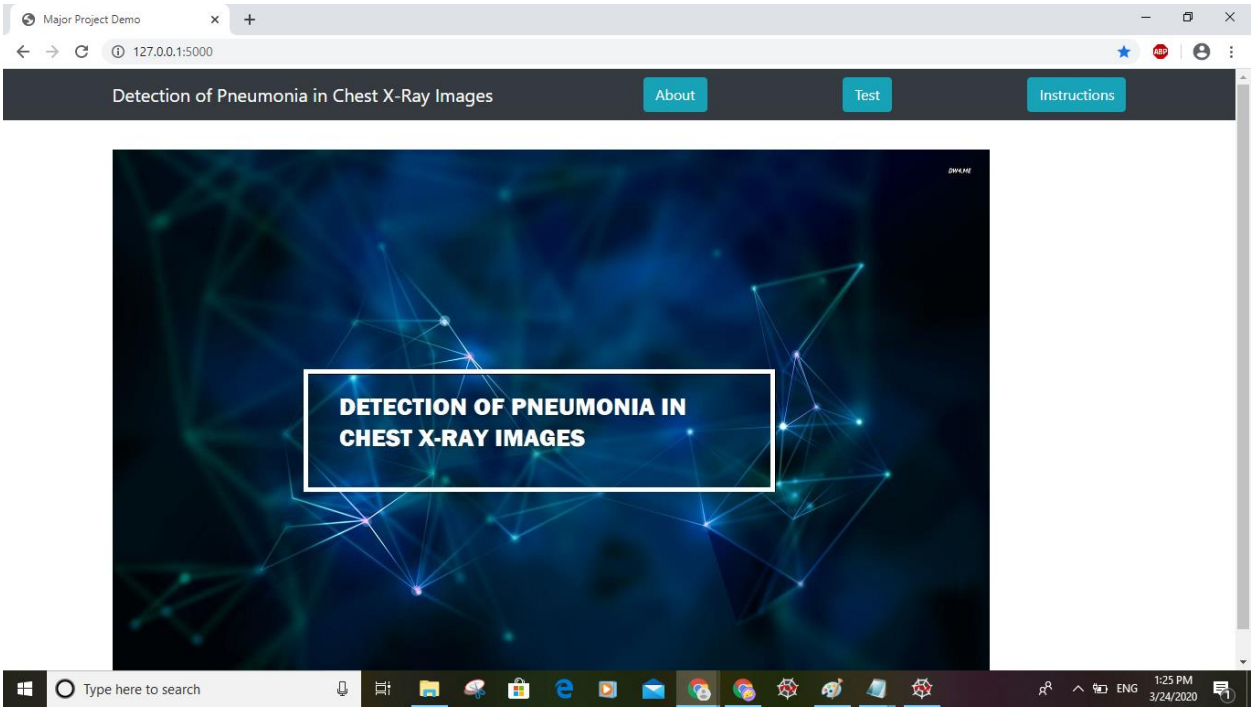

Figure 7.10: Home Page

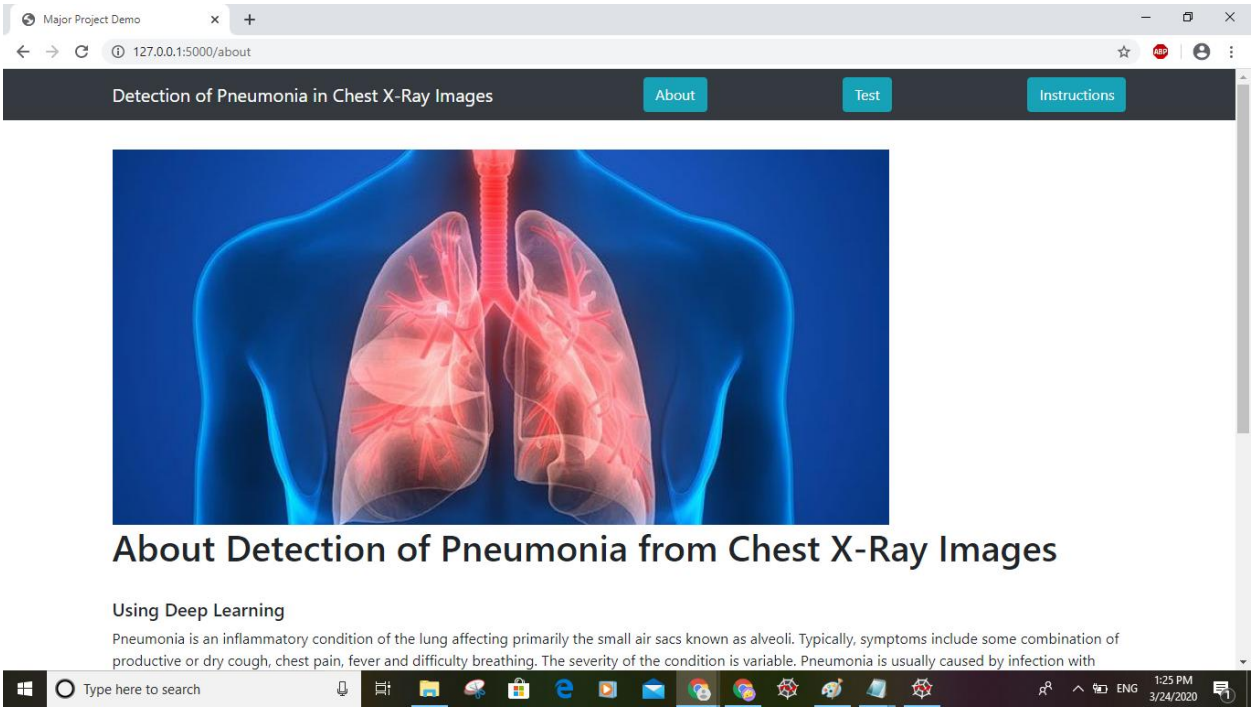

Figure 7.11: About Page

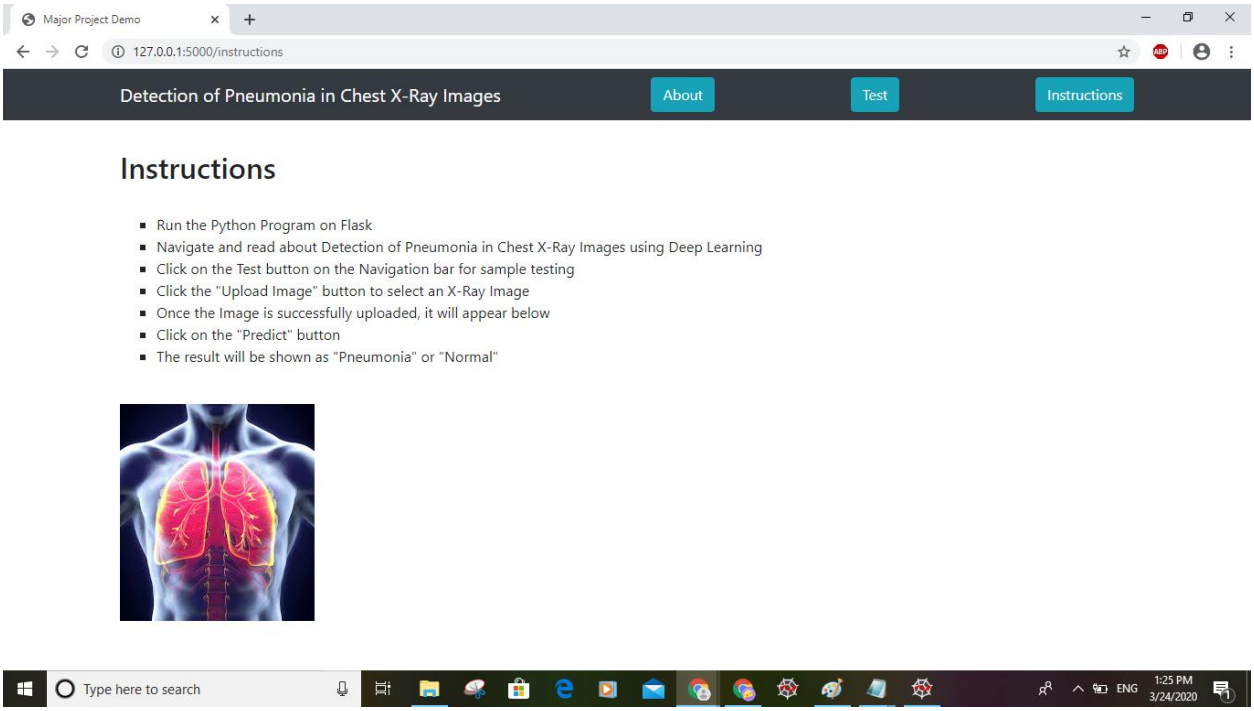

Figure 7.12: Instructions Page

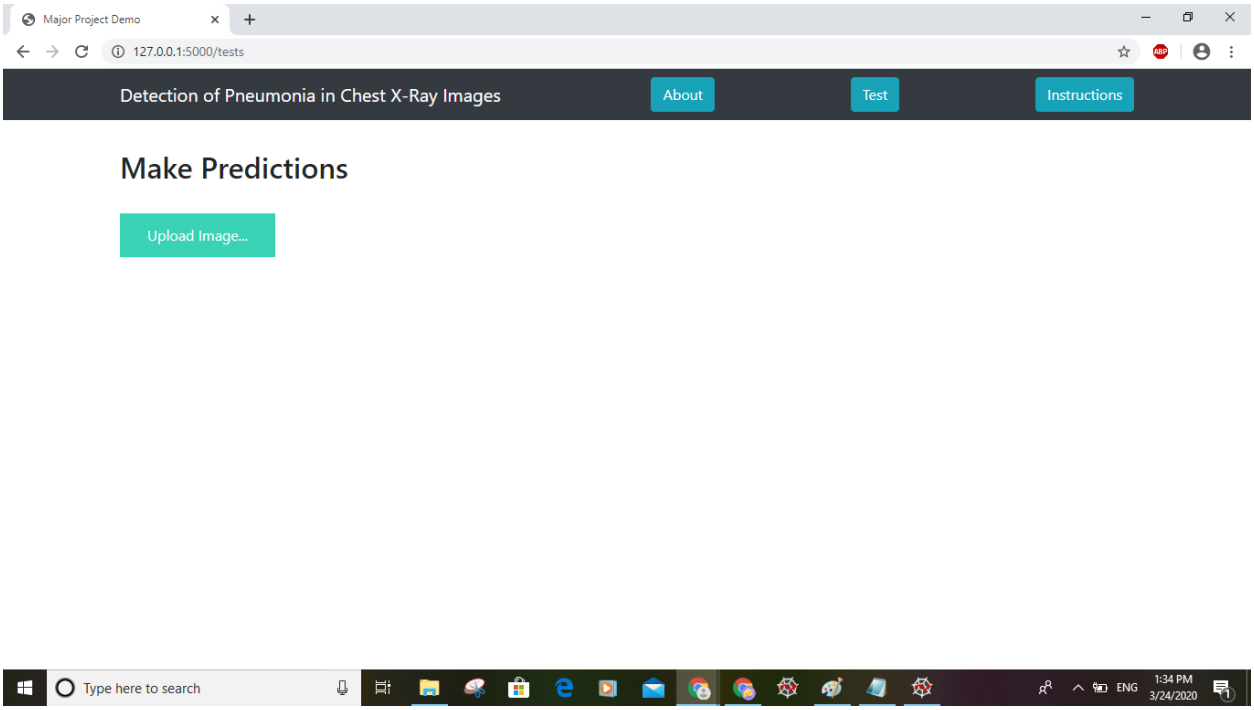

Figure 7.13: Test Page

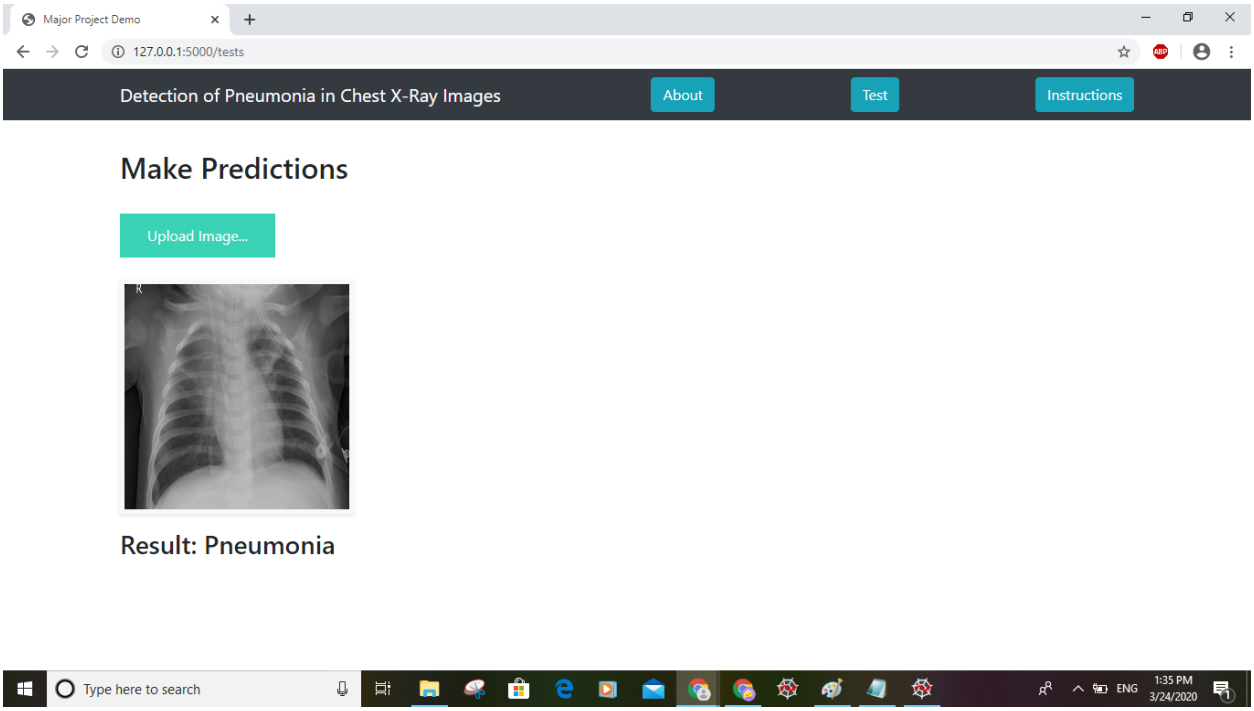

Figure 7.14: Sample Pneumonia Prediction

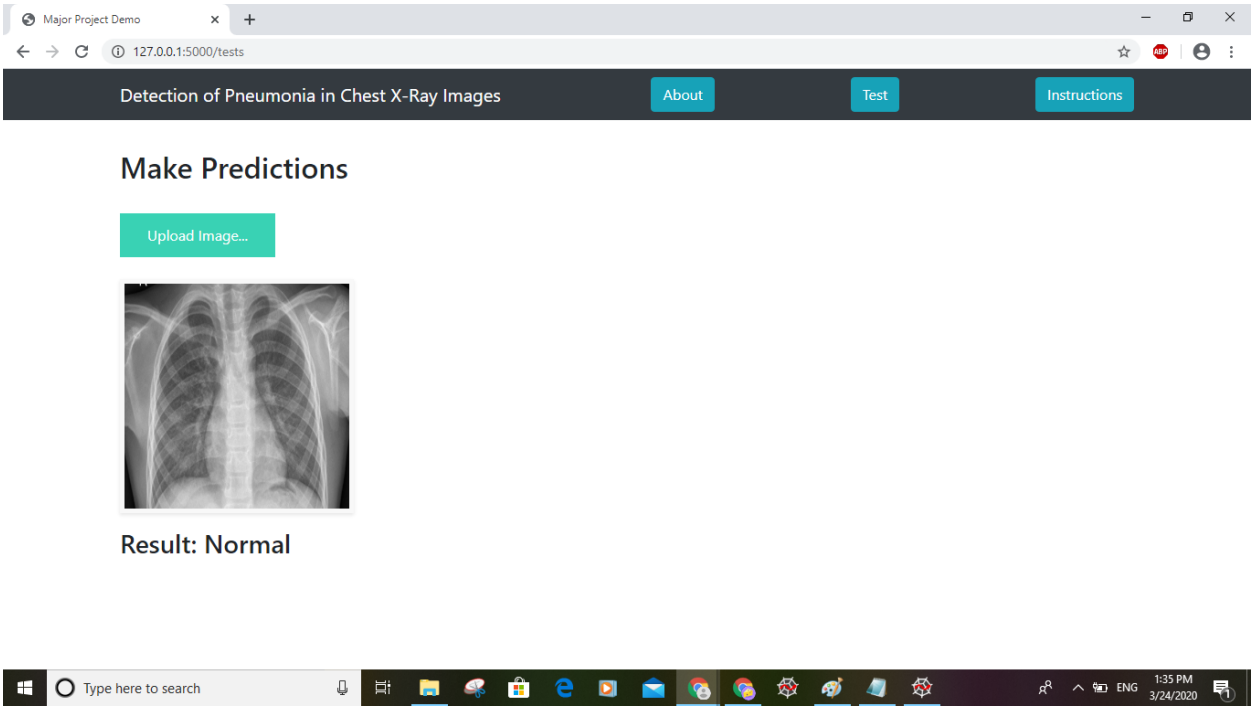

Figure 7.15: Sample Normal Prediction

8. CONCLUSION AND FUTURE SCOPE

8.1 CONCLUSION

Pneumonia is an awful disease of the lungs which is a leading cause of death worldwide. Here we have shown that a system for the accurate detection of pneumonia can be developed, using transfer learning and data augmentation, which will allow for the easy detection of pneumonia in patients using their chest x-rays. We have demonstrated how to classify positive and negative pneumonia data from a collection of X-ray images. We compared two CNN network's performance using transfer learning approach and a custom model which we build from scratch which separates it from other methods on the diagnosis of pneumonia disease. While training, our model we used from transfer learning and fine tuning. After the training phase, we compared three network test results in which vgg16 model showed an accuracy of 88.9%, custom model with 90.2% accuracy and Inceptionv3 model with 82.05% accuracy.

8.2 FUTURE SCOPE

With further refinement, this system may be used by medical professionals to reduce the amount of workload they feel for identifying pneumonia in patients by feeding chest x-rays into a system like the one presented here, medical professionals will only need to review outputted images which may be false negatives a small percentage of the whole. In the future, this project will be extended to detect and classify X-ray images consisting of lung cancer and pneumonia. Distinguishing X-ray images that contain lung cancer and pneumonia has been a big issue in recent times, and our next approach will tackle this problem.

REFERENCES

[1] Enes AYAN and Halil Murat ÜNVER, *Diagnosis of Pneumonia from Chest X-Ray Images using Deep Learning*, IEEE Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science, Vol. , pp.27 – 34, 2019.

[2] Goldbaum Michael, Kang and Z. Kermany Daniel, *Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification*, Mendely, Vol. 3, June 1, 2018.

[3] Aarti Bagul, Pranav Rajpurkar, et al., *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning*, arXiv:1711.05225v3 [cs.CV], December 25, 2017.

[4] Arata Saraiva, Jose Vigno Moura Sousa, Luciano Lopes de Sousa, and N. M. Fonseca Ferreira, *Classification of Images of Childhood Pneumonia using Convolutional Neural Networks*, ReasearchGate, January, 2019.

[5] Mangal Sain, Okeke Stephen and Uchenna Joseph Maduh, *An Efficient Deep Learning Approach to Pneumonia Classification in Healthcare*, Journal of Healthcare Engineering, Vol. 2019, March 27, 2019.

[6] Aditya Khamparia, Sanjay Kumar Singh, Victor Hugo C. de Albuquerque and Vikash Chouhan, *An Efficient Deep Learning Approach to Pneumonia Classification in Healthcare*, Molecular Diversity Preservation International and Multidisciplinary Digital Publishing Institute (MDPI) Journals, Vol.10, Issue 2, January 12, 2020.

[7] G. B. Van, J. Melendez, P. Maduskar et al., *A novel multiple instance learning-based approach to computer-aided detection of tuberculosis on chest x-ray*, IEEE Transactions on Medical Imaging, vol. 34, no. 1, pp. 179–192, 2015.

[8] A. Seff, H. C. Shin, J. Yao, L. Lu, L. Kim, and R. M. Summers, *Interleaved text/image deep mining on a large-scale radiology database for automated image interpretation*, Journal of Machine Learning Research, vol. 17, no. 107, pp. 1–31, 2016.

[9] Le Lu, Mohammadhadi Bagheri, Ronald M. Summers, Xiaosong Wang, Yifan Peng, and Zhiyong Lu, *ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, Vol 5, pp. 3462–3471, July 2017.

[10] J. Ker, J. Rao, L. Wang, and T. Lim, *Deep learning applications in medical image analysis*, IEEE Access, vol. 6, pp. 9375-9389, 2018.

[11] D. Wang, K. Weiss, and T. M. Khoshgoftaar, *A survey of transfer learning*, Journal of Big Data, vol. 3, p. 9, 2016.

[12] CDC Pneumonia Statistics. Available: https://www.cdc.gov/pneumonia/prevention.html

[13] "Image Classification Transfer Learning with Inception v3", Codelabs, Available: https://codelabs.developers.google.com/codelabs/cpb102-txf-learning

[14] "A Gentle Introduction to Transfer Learning for Deep Learning", Machine Learning Mastery, Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[15] "Step by step VGG16 implementation in Keras for beginners", Towards Data Science, Available: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

[16]"Models for image classification with weights trained on ImageNet", Available: https://keras.io/applications/