A Major Project Final Report on

Pneumonia Detection Using Convolution Neural Network

Submitted in Partial Fulfillment of the Requirements for the

Degree of **Bachelors of Engineering in Computer**

Under Pokhara University

Submitted by:

Akshata Pradhan, 15301

Manish Ghimire, 15320

Mini Maharjan, 15321

Date:

25th Nov, 2019

Department of Computer Engineering

NEPAL COLLEGE OF

<u>INFORMATION TECHNOLOGY</u>

Balkumari, Lalitpur, Nepal

Abstract

Pneumonia is an infection in the lungs that can be caused by bacteria, viruses, or fungi. It can be difficult to diagnose because the symptoms are similar to cold or influenza. It is generally diagnosed using physical symptoms, blood test, and chest X-rays. Detecting pneumonia is not that easy, well-experienced doctors are required to diagnose accurately. We have used VGG16 classifier based on convolution neural network which will detect and classify the presence of pneumonia from a chest X-ray image. Chest X-ray is currently the best method for diagnosing pneumonia. With the computer-aided diagnosis, physicians can make chest X-ray diagnosis more quickly and accurately. User will upload a Chest X-ray image in the web application and Convolutional Neural Network (CNN) will detect whether a patient has pneumonia or not based on X-ray image of their chest.Validation will be done using confusion matrix.

Keywords*:* *pneumonia, chest X-ray, convolution neural network, binary classification, detection, web application*

Contents

List of figures

1. Introduction

This project, Pneumonia Detection using Convolution Neural Network focuses on detecting pneumonia with the help of chest X-ray image. We aim to develop a Convolutional Neural Network (CNN) that can detect whether a patient has pneumonia or not, based on an X-ray image of their chest. This project is also proposed to study the different kinds of pre-existing models and tools for detecting a disease from radiology. Chest X-rays are currently the best method for diagnosing pneumonia**.** But there is still a lack of access with almost two-thirds of the world's population to radiology diagnostics. It is also much more difficult to make clinical diagnoses with chest X-rays than with other imaging modalities such as CT scan or MRI [1]. This leads to inaccurate results, our project aims to reduce the cost of detecting pneumonia with just a chest X-ray image which could cost a lot if CT scan or MRI has to be done and it can take several days. So, by using better technologies we could make tests more accurate.

Pneumonia is an infection in the lungs that can be caused by bacteria, viruses, or fungi. It is a serious and life-threatening disease. The lungs become inflamed, and the tiny air sacs, or alveoli, inside the lungs fill up with fluid. It can occur in young and healthy people, but it is most dangerous for older adult and infants, people with other diseases, and those with impaired immune systems. Children younger than 5 years age worldwide die due to pneumonia [2].

The risk of pneumonia is immense for many, especially in developing nations where billions face energy poverty and relay on pollution forms of energy. The WHO estimates that over four millions of premature death occur annually from household pollution-related disease including pneumonia [3]. Based on the data published by Patan hospital an average of 500 patient are diagnosed with pneumonia yearly, among which most of the patients are of age under 5. The first symptom of pneumonia is similar to that of cold or flu, slowly it can turn into high fever, chills, and cough with sputum. Other symptoms of pneumonia include:

- Fast heartbeat
- Fast breathing and shortness of breath
- Chest pain that usually worsens when taking a deep breath
- Sweating, nausea and vomiting, diarrhea, muscle pain

- Dusky or purplish skin color, or cyanosis, from poorly oxygenated blood.

Pneumonia can be difficult to diagnose because the symptoms are so variable and are similar to those seen in a cold or influenza. To diagnose medical history, physical examination and some tests like a blood test, chest X-ray, pulse oximetry and sputum test are done [4].   .

1.1.Problem Statement

Pneumonia is not that easy for doctors to detect, even doctors having good experience might not be able to detect pneumonia by checking x-ray of the patients. There is still a lack of access with almost two-thirds of the world's population lacking access to radiology diagnostics. It is also much more difficult to make clinical diagnoses with chest X-rays than with other imaging modalities such as CT scan or MRI. But CT scan and MRI are expensive techniques. This may lead to inaccurate results. In the context of Nepal, many of the rural government health posts lack basic equipment, and some have not been staffed for years. Rural areas of Nepal have one doctor for every 150,000 people so it is difficult for patients to be diagnosed properly.

To eradicate such defects in pneumonia detection, better technologies can be developed that can improvise detection and testing. Automating this detection task would greatly improve the efficiency of radiologists.

1.2. Project Objective

This project involves both the studying of the existing system and combining the methods of the existing systems in such a way that they give an optimal solution. The objective of the project is to

- Come up with the system for conformation of Pneumonia in patients by doctors or a normal person.
- Develop tool for pneumonia detection in a remote area where experienced doctors are not available.
- To develop a user-friendly Web-application for Pneumonia detection.

1.3. Significance of study

The study helps in the development of the system that detects pneumonia. It provides accurate information to the users regarding pneumonia. It also helps in making the system dynamic as per the requirement of the user as the data about the patients can be added, updated or deleted as per requirements. Finally, the study also looks into other similar kinds of system in use to explore the benefits of the system as well as its drawbacks.

2. Literature review

The progress that artificial intelligence (AI) has made with regard to radiology has indeed exceeded any and all expectations in terms of providing accurate, automated diagnoses, and, most cases, even outshined human healthcare professionals.
One prominent example includes the algorithm, ‘CheXNet,’ an artificial neural network designed to detect pneumonia from chest X-rays, at a performance rate greater than the average radiologist. The system was developed by Stanford University and tested against four practicing radiologists to diagnose 14 diseases and achieved “state-of-the-art results” on all of them[5].
Anjana Tiha detected pneumonia form Chest X-ray image using Custom Deep Convolution Neural Network and by retraining pre-trained model “IncpectionV3” with 5856 images of X-ray with testing accuracy 89.53 % and loss 0.41 [6].
YuanTian evaluated CNN classifier by using separate train sets and test set with the conclusion that CNN classifier is 91% accurate [7]. Latest improvements in deep learning models and the availability of huge datasets have assisted algorithms to outperform medical personnel in numerous medical imaging tasks such as skin cancer classification [8], hemorrhage identification [9], arrhythmia detection [10] , and diabetic retinopathy detection [11]. Automated diagnoses enabled by chest radiographs have received growing interests. These algorithms are increasingly being used for conducting lung nodule detection [12] and pulmonary tuberculosis classification [13].
Benjamin Antin, Joshua Kraitz and Emil Martayan detected pneumonia in Chest X-ray with Supervised Learning with the conclusion that logistic regression does not adequately capture the complexities of dataset [14]. Rahin H. Abiyev and Mohammad Khaleel Sallam Ma’aitah designed CNN for diagnosis of chest diseases with the conclusion that CNN is better than other deep CNN models such as GIST, VGG16, and VGG19 [15]. Alishba Imran trained a CNN to detect Pneumonia with the accuracy of 88.89% [16].

2.1. Supervised Machine Learning:

Supervised learning, in the context of artificial intelligence (AI) and machine learning, is a type of system in which both input and desired output data are provided. Input and output data are labelled for classification to provide a learning basis for future data processing. The term supervised learning comes from the idea that an algorithm is learning from a training dataset, which can be thought of as the teacher. Supervised machine learning systems provide the learning algorithms with known quantities to support future judgments. Supervised learning systems are mostly associated with retrieval-based AI but they may also be capable of using a generative learning model [17].

The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output **Y = f(X)**. The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data [18].

2.2. Convolution Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered.
This independence from prior knowledge and human effort in feature design is a major advantage.

A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification, segmentation and also for other auto correlated data.
A convolution is essentially sliding a filter over the input. One helpful way to think about convolutions is this quote from Dr. Prasad Samarakoon: “A convolution can be thought as “looking at a function’s surroundings to make better/accurate predictions of its outcome.”
Rather than looking at an entire image at once to find certain features it can be more effective to look at smaller portions of the image [19].

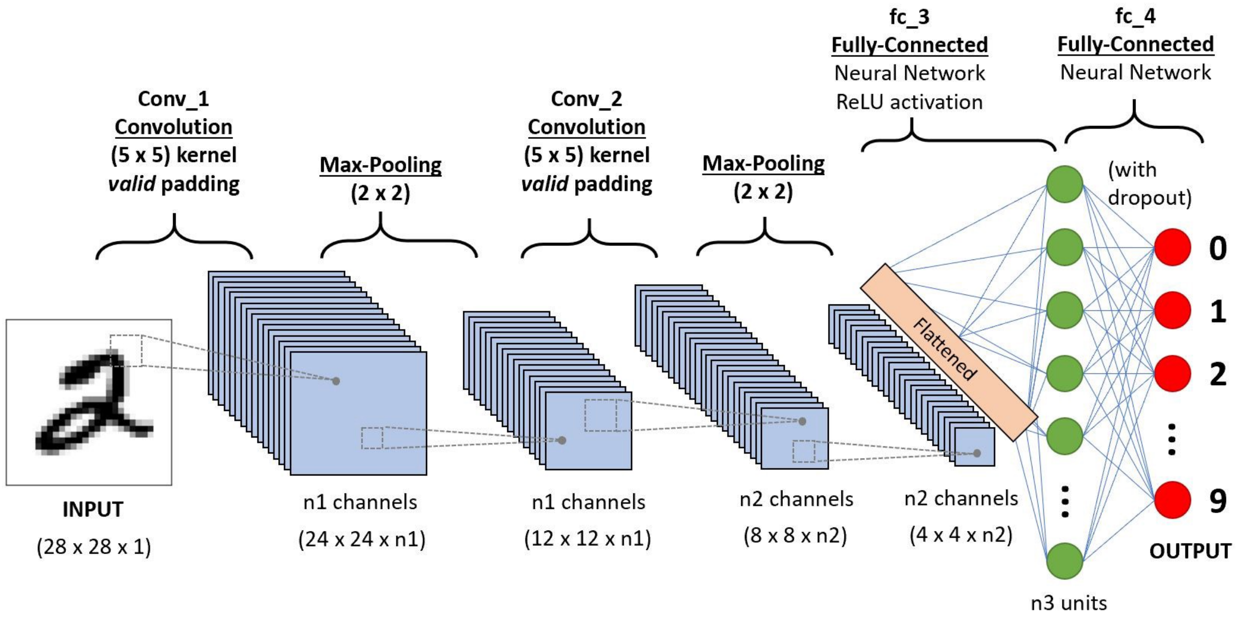


Figure 1. 1: A CNN sequence to classify handwritten digits

2.2.1. Layers in CNN

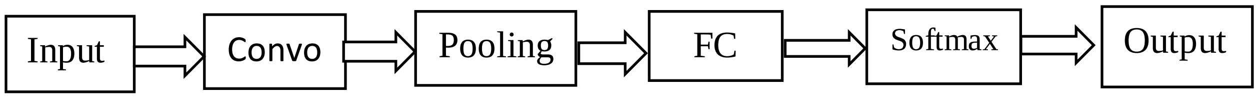


Figure 1. 2: Layers in CNN

2.2.1.1. Input Layer

Input layer in CNN should contain image data. Image data is represented by three dimensional matrix. We need to reshape it into a single column.

2.2.1.2. Convo Layer

Convo layer is sometimes called feature extractor layer because features of the image are extracted within this layer. First of all, a part of image is connected to Convo layer to perform convolution operation and calculating the dot product between receptive field(it is a local region of the input image that has the same size as that of filter) and the filter. Result of the operation is single integer of the output volume. Then we slide the filter over the next receptive field of the same input image by a Stride and do the same operation again. We have to repeat the same process again and again until we go through the whole image. The output will be the input for the next layer.
Convo layer also contains ReLU activation to make all negative value to zero.

2.2.1.3. Pooling Layer

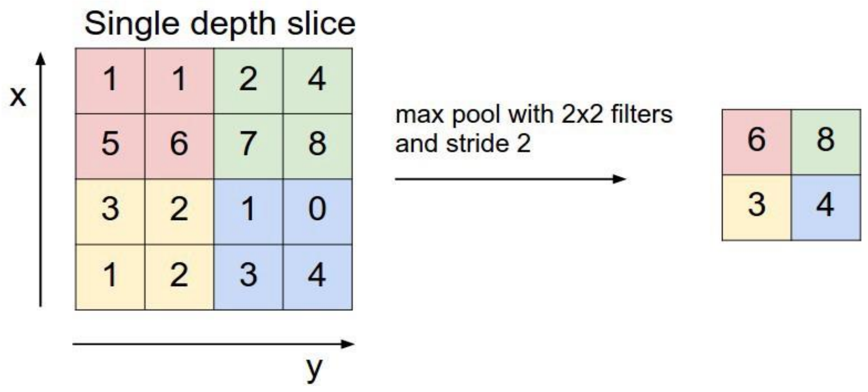


Figure 1. 3: Pooling layer

Pooling layer is used to reduce the spatial volume of input image after convolution. It is used between two convolution layer. If FC layer is applied after Convo layer without applying pooling or max pooling, then it will be computationally expensive . So, the max pooling is only way to reduce the spatial volume of input image. Max pooling is applied in single depth slice with Stride of 2. We can observe the 4 x 4 dimension input is reduce to 2 x 2 dimension.
There is no parameter in pooling layer but it has two hyperparameters — Filter(F) and Stride(S).
In general, if input dimension is W1 x H1 x D1, then
$W2 = (W1-F)/S+1$
$H2 = (H1-F)/S+1$
$D2 = D1$
Where W2, H2 and D2 are the width, height and depth of output.

2.2.1.4. Fully Connected Layer(FC)

Fully connected layer involves weights, biases, and neurons. It connects neurons in one layer to neurons in another layer. It is used to classify images between different category by training.

2.2.1.5.Softmax / Logistic Layer

Softmax or Logistic layer is the last layer of CNN. It resides at the end of FC layer. Logistic is used for binary classification and softmax is for multi-classification.

2.2.1.6. Output Layer

Output layer contains the label which is in the form of one-hot encoded (medium, 2019) https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529.

2.3. Backpropagation

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

Backpropagation is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respects to all the weights in the network.

Backpropagation repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. In other words, backpropagation aims to minimize the cost function by adjusting network's weights and biases. The level of adjustment is determined by the gradients of the cost function with respect to those parameters.

Computing Gradients:

- Gradient of a function C(x_1, x_2, …, x_m) in point x is a vector of the partial derivatives of C in x.

$$\frac{\partial C}{\partial x} = \left[\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \dots, \frac{\partial C}{\partial x_m}\right]$$

Equation for derivative of C in x

- The derivative of a function C measures the sensitivity to change of the function value (output value) with respect to a change in its argument x (input value). In other words, the derivative tells us the direction C is going.

- The gradient shows how much the parameter x needs to change (in positive or negative direction) to minimize C.

Computation of those gradients happens using a technique called chain rule.

For a single weight $(w\_jk)^l$ the gradient is:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} \qquad \textit{chain rule}$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad by\ defination$$

$$where, m = number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\ a_k^{l-1} \qquad final\ value$$

Equations for derivative of C in a single weight $(w\_jk)^l$:

Similar set of equations can be applied to $(b\_j)^l$:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} \qquad \textit{chain rule}$$

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} 1 \qquad final\ value$$

Equations for derivative of C in a single bias b_j^l :

The common part in both equations is often called *"local gradient"* and is expressed as follows:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \qquad local\ gradient$$

Equation for local gradient

The *"local gradient"* can easily be determined using the chain rule. I won't go over the process now but if you have any questions, please comment below.

The gradients allow us to optimize the model's parameters:

While (termination condition not met)

$$w := w - \in \frac{\partial C}{\partial w}$$

$$b := b - \in \frac{\partial C}{\partial b}$$

end

Algorithm for optimizing weights and biases (also called "Gradient descent")

- Initial values of *w* and *b* are randomly chosen.
- Epsilon (*e*) is the learning rate. It determines the gradient's influence.
- *w* and *b* are matrix representations of the weights and biases. Derivative of *C* in *w* or *b* can be calculated using partial derivatives of *C* in the individual weights or biases.
- Termination condition is met once the cost function is minimized.

I would like to dedicate the final part of this section to a simple example in which we will calculate the gradient of *C* with respect to a single weight w_{22}^2 .

Weight w_{22}^{2} connects a_2^2 and z_2^2 , so computing the gradient requires applying the chain rule through z_2^3 and a_2^3 :

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} . \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} . \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} . a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} . f'\left(z_2^{(3)}\right) . a_2^{(2)}$$

Equation for derivative of C in w_{22}^{2}

Calculating the final value of derivative of *C* in a_2^3 requires knowledge of the function *C*. Since *C* is dependent on a_2^3 , calculating the derivative should be fairly straightforward [20].

2.4. Activation Function

Activation functions are an extremely important feature of the artificial neural networks. They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored.

$$\text{Y = Activation}(\Sigma(weight * input) + bias)$$

The activation function is the non-linear transformation that we do over the input signal. This transformed output is then send to the next layer of neurons as input.

When we do not have the activation function the weights and bias would simply do a linear transformation. A linear equation is simple to solve but is limited in its capacity to solve complex problems. A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks. We would want our neural networks to work on complicated tasks like language translations and image classifications. Linear transformations would never be able to perform such tasks.

Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases. Without the differentiable non-linear function, this would not be possible [21].

2.4.1. Types of Activation Function

2.4.1.1. Sigmoid / Logistic Function

Advantages

- Smooth gradient, preventing "jumps" in output values.
- Output values bound between 0 and 1, normalizing the output of each neuron.
- Clear predictions—For X above 2 or below -2, tends to bring the Y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.

Disadvantages

- Vanishing gradient—for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.
- Outputs not zero centered.
- Computationally expensive.

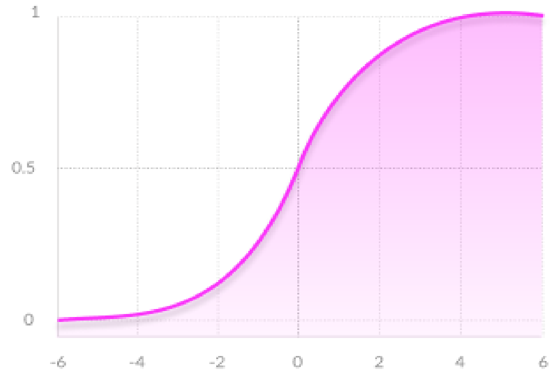


Figure 1. 4: Sigmoid Function

2.4.1.2.TanH / Hyperbolic Tangent

Advantages

- Zero centered—making it easier to model inputs that have strongly negative, neutral, and strongly positive values.
- Otherwise like the Sigmoid function.

Disadvantages

- Like the Sigmoid function.

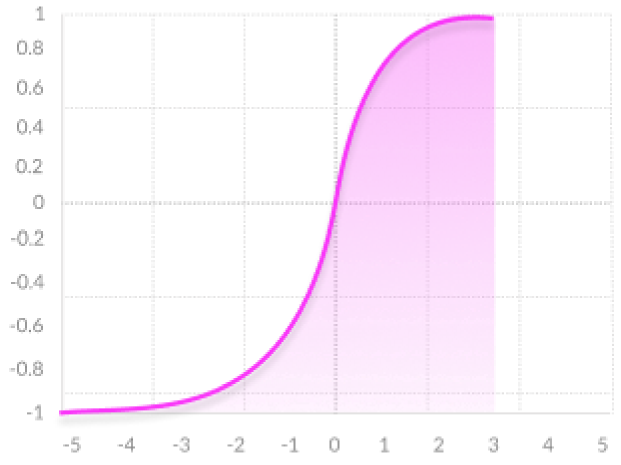


Figure 1. 5: TanH function

2.4.1.3. ReLU (Rectified Linear Unit)

Advantages

- Computationally efficient—allows the network to converge very quickly
- Non-linear—although it looks like a linear function, ReLU has a derivative function and allows for backpropagation
-

Disadvantages

- The Dying ReLU problem—when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

Mathematically it is defined by:

$$f(x) = \max(0, x) = \begin{cases} x_i \ if \ x_i > 0 \\ 0 \ if \ x_i < 0 \end{cases}$$

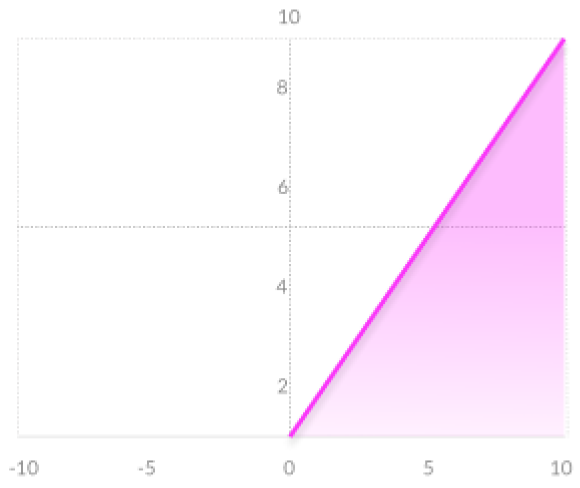


Figure 1. 6: ReLU function

2.4.1.4. Leaky ReLU

Advantages

- Prevents dying ReLU problem—this variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values
- Otherwise like ReLU.

Disadvantages

- Results not consistent—leaky ReLU does not provide consistent predictions for negative input values.

$$f(x)=max(\alpha\partial,\partial)$$

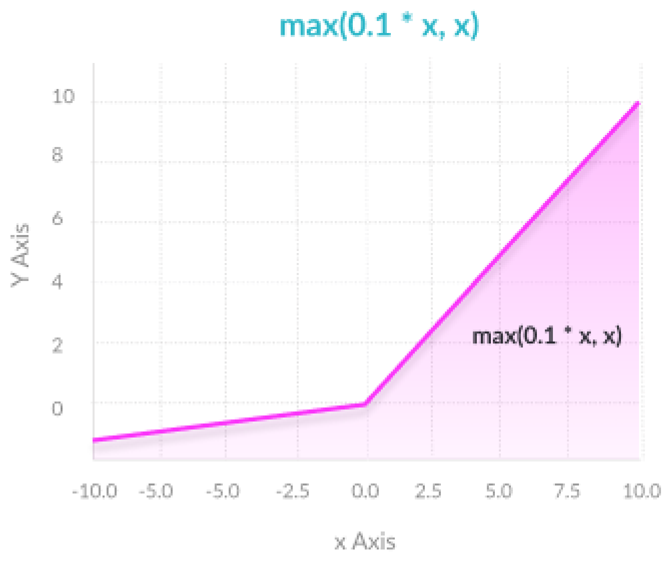


Figure 1. 7: Leaky ReLU function

2.4.1.5. ReLU

Advantages

- Allows the negative slope to be learned—unlike leaky ReLU, this function provides the slope of the negative part of the function as an argument. It is, therefore, possible to perform backpropagation and learn the most appropriate value of α.
- Otherwise like ReLU

Disadvantages

- May perform differently for different problems.

$$\sigma(z)j= \frac{ezj}{\sum_{k=1ezk}^{k}} \quad for\ j= 1....k.$$

2.4.1.6. Softmax

Advantages

- Able to handle multiple classes only one class in other activation functions—normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class.
- Useful for output neurons—typically Softmax is used only for the output layer, for neural networks that need to classify inputs into multiple categories.

2.4.1.7. Swish

Swish is a new, self-gated activation function discovered by researchers at Google. It performs better than ReLU with a similar level of computational efficiency. In experiments onImageNet with identical models running ReLU and Swish, the new function achieved top -1 classification accuracy 0.6-0.9% higher [22].

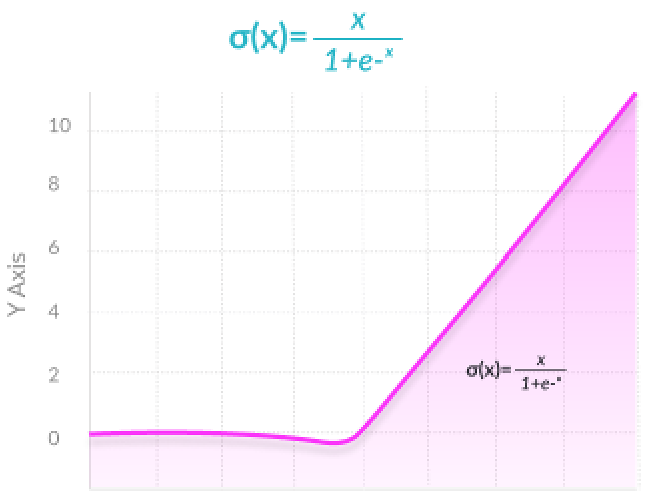


Figure 1. 8: Swish function

2.5. VGG16 – Convolutional Network for Classification and Detection

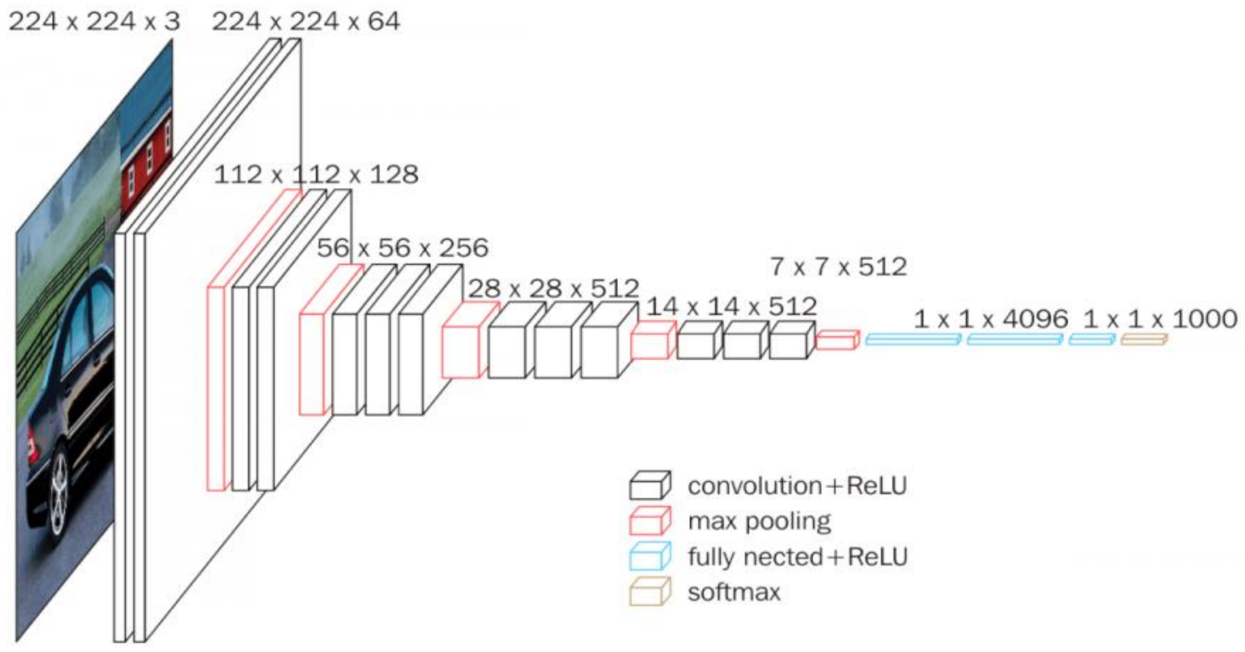
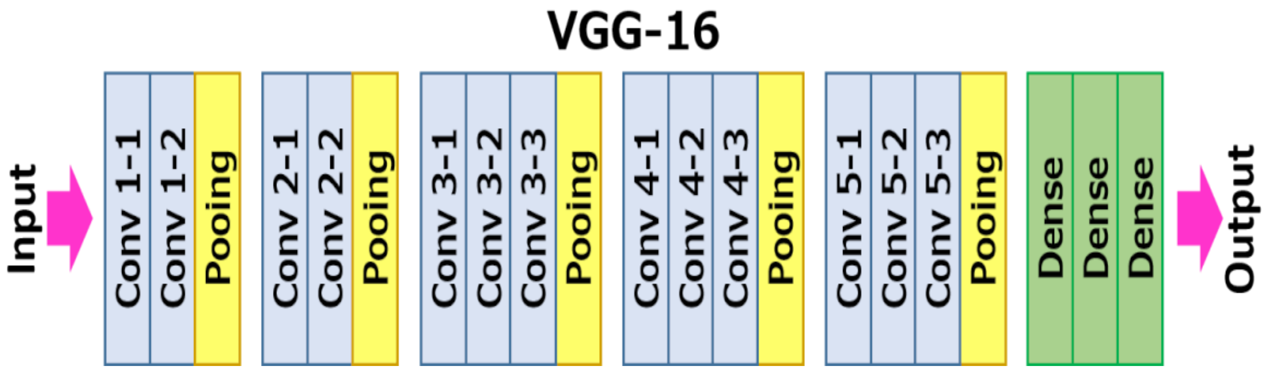


Figure 1. 9: Architecture of VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU’s [23].



2.6. Transfer Learning

Transfer learning is an approach used in machine learning where a model that was created and trained for one task, is reused as the starting point for a secondary task. Transfer learning differs from traditional machine learning because it involves using a pre-trained model as a springboard to start a secondary task.

Advantages of Transfer Learning:

- Less training data—starting to train a model from scratch is a lot of work and requires a lot of data. For example, if we want to create a new algorithm that can detect a frown, we need a lot of training data. Our model will first need to learn how to detect faces, and only then can it learn how to detect expressions, such as frowns. Instead, if we use a model that has already learned how to detect faces, and retrain this model to detect frowns, we can accomplish the same result using far less data.

- Models generalize better—using transfer learning on a model prepares the model to perform well with data it was not trained on. This is known as generalizing. Models that were trained using transfer learning are better able to generalize from one task to another because they were trained to learn to identify features that can be applied to new contexts.

- Makes deep learning more accessible—working with transfer learning makes it easier to use deep learning. It's possible to obtain the desired results without being an expert in deep learning, by using a model that was created by a deep learning specialist and applying it to a new problem.

Types of Transfer Learning:

- Domain adaptation

In this approach, a dataset on which the machine was trained is different from (but still related to) the target data set. A good example of this would be a spam email filtering model. Let's say this model was trained to identify spam email for user A. When the model is then used for user B, domain adaptation will be used, because even though the task is the same (filtering emails), user B receives different types of emails from user A.

- Multitask learning

This method involves two or more tasks being resolved simultaneously so that similarities and differences can be leveraged. It is based on the idea that a model that has been trained on a related task can gain skills that improve its ability in the new task.
Going back to our spam email filtering model, let's say this model is learning what features it should look for when identifying spam mail for user A and user B. Because the users are very different, the model needs to look for different features in order to identify each users' spam mail. For example, user A is an Italian speaker so an Italian language email should not be a red flag. However, user B is a Chinese speaker, so an email in Italian might be considered a spam feature. While simultaneously learning to identify spam features for user A and B, the model learns that regardless of the language, emails requesting credit card details are more likely to be spam.

- Zero-shot learning

This technique involves a model trying to solve a task to which it was not exposed during training. For example, let’s say we are training a model to identify animals in pictures. To identify the animals, the machine is taught to identify two parameters: the color yellow and spots. The model is then trained on multiple pictures of chicks, which it learns to identify because they are yellow but do not have spots, and dalmations, which it knows has spots but are not yellow.

To expand on this example, you may not have pictures of giraffes on which to train the model, but the model knows that giraffes are yellow and have spots. When the model encounters an image of a giraffe, it will be able to identify it, even though it had never seen a giraffe in training.

- One-shot learning

This approach requires that a model learns how to categorize an object, after being exposed to it either once or just a few times. To do this, the model leverages information it has about known categories. For example, our animal classifying model knows how to identify a horse. The model is then exposed to a single photo of a zebra, which looks exactly like a horse but has white and back stripes. The model will then be able to classify zebras, without being exposed to additional pictures, because it transferred knowledge it already had about horses (missinglink, n.d.).

3. Methodology

3.1. Introduction

There are three main parts of the system. The first part is client which is built from Django. Its interface is used to upload the image and see the predicted result. The second part is a SQLite database which is the default database for Django. The last part is API which is built from flask which runs independently. Initially, the client inputs the chest x-ray image into the client part which saves the data to the database and passes the data to the Flask API. Then the API will analyze the image and predict output. In the API, the trained Pneumonia classifier model is loaded with its target size, the default target size for the VGG model is 224*224. The string encoded image is decoded into jpeg format. The decoded image is passed to preprocess image where the image is resized and formatted. Finally, image is passed to production algorithm. The prediction returns the output of an array (0, 0) value ranges from 0 to 1, where 0 means Pneumonia not detected and 1 means detected.

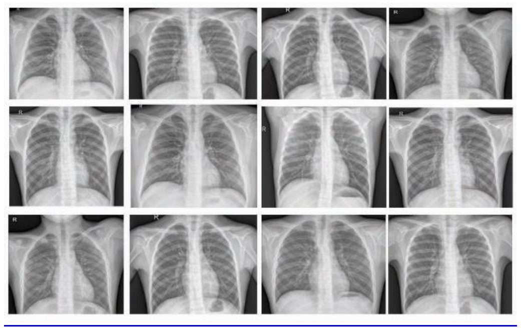
About model architecture, we take images of chest x-ray having one with Pneumonia and another without pneumonia. We divide the images into 3 sets: Train data, Test data and Validation data with the images of set 5216, 624, 16 respectively. Initially, we train our model using VGG16 with the weight provided by imagenet. Before training, we remove the top layer from ResNet50. Also, we added two activation functions ReLU and Sigmoid. For training, we run the model to 16 epochs. After Training the model to 16 epochs we got, loss: 0.0703, accuracy: 0.9743 – val loss: 0.0493 - val_acc: 1.0000.

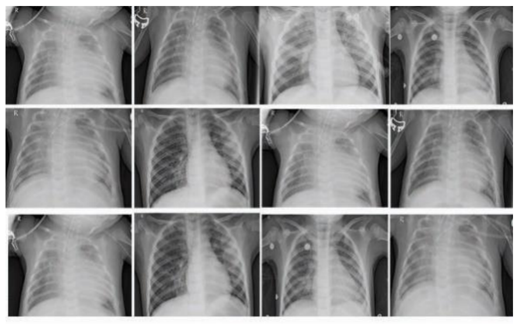
3.2. Data Sets

The dataset is taken from Kaggle. The dataset is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal). There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children’s Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients’ routine clinical care.

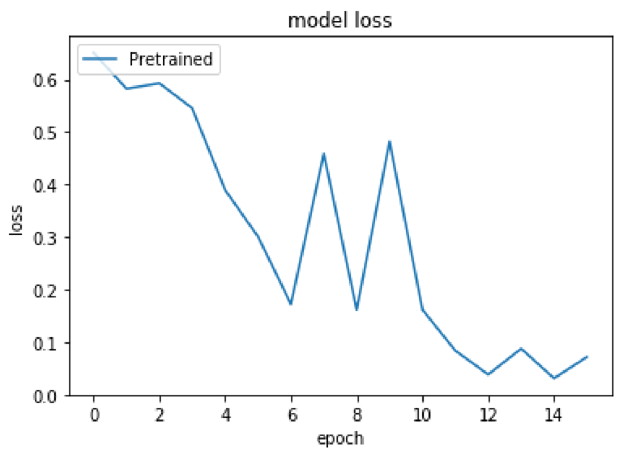
For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert (Kosovan, 2018)https://www.kaggle.com/kosovanolexandr/keras-nn-x-ray-predict-pneumonia-86-54

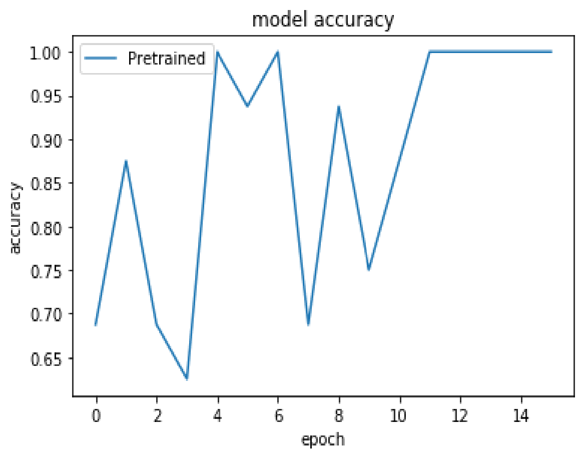


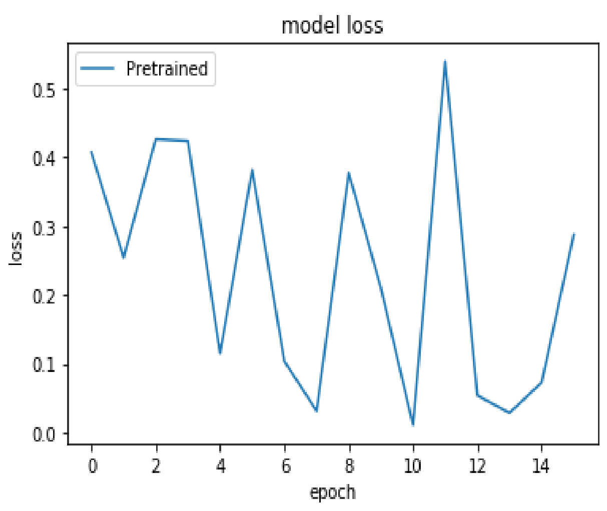


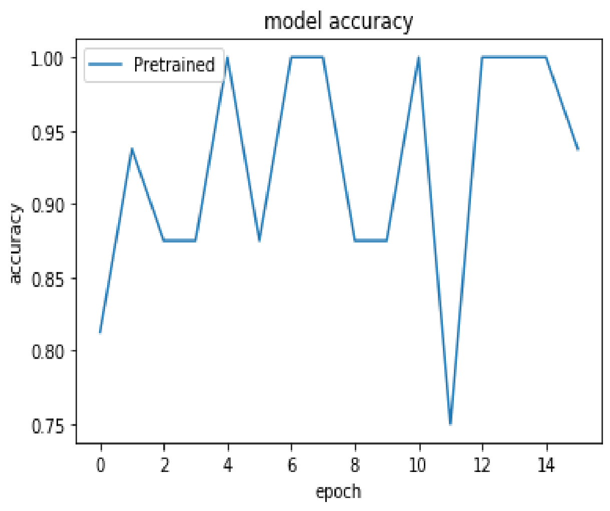
3.3. Training the Model

There are various ways to train a model. We have use a VGG16 model which we have customize the output classes for the prediction of custom dataset. Dataset are divided in 85% of data for training and 15% for testing. Dataset will be divided into three groups training, testing and validating where model will be trained only in training dataset keeping data set untouched and evaluated in test dataset. Model will be run in testing dataset for better results.









3.4. Software development life-cycle

Iterative Model – Design

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental) [24].

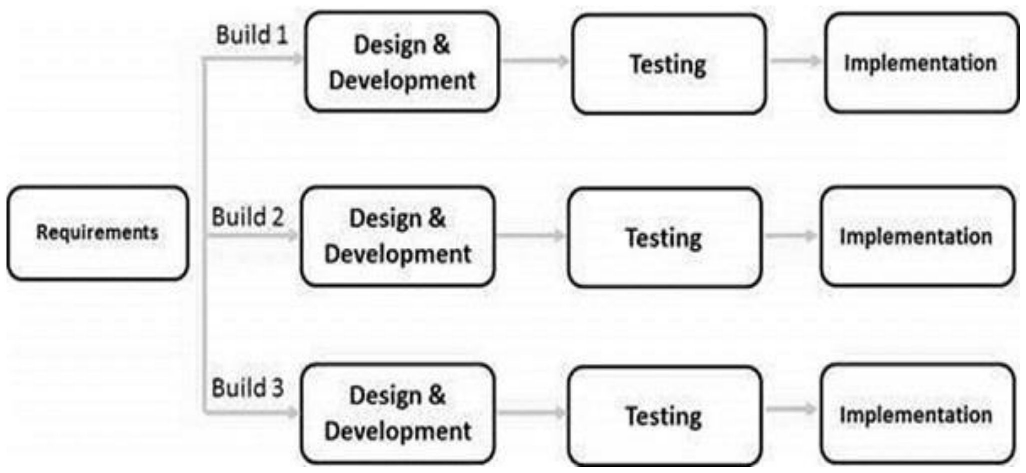


Figure 1. 10: Iterative Model - Design

3.5. Tools and technique

Python

Python is an interpreted high-level programming language for general-purpose programming. Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Mysql

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language. MySQL is free and open-source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses.

Django

Django is a Python-based free and open-source web framework, which follows the model template-view (MTV) architectural pattern. It is maintained by the Django Software Foundation (DSF), an independent organization Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself.

Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Opencv

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross platform and free for use under the opensource BSD license.

Keras

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

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. It is used for both research and production at Google.

Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.

CNN

A CNN is basically an Artificial Neural Network (ANN) with additional Convolutional layers at the input.The architecture will vary from model to model, and it is this difference that causes the difference in model performance.

Confusion matrix

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

4. System Design

4.1. System Architecture

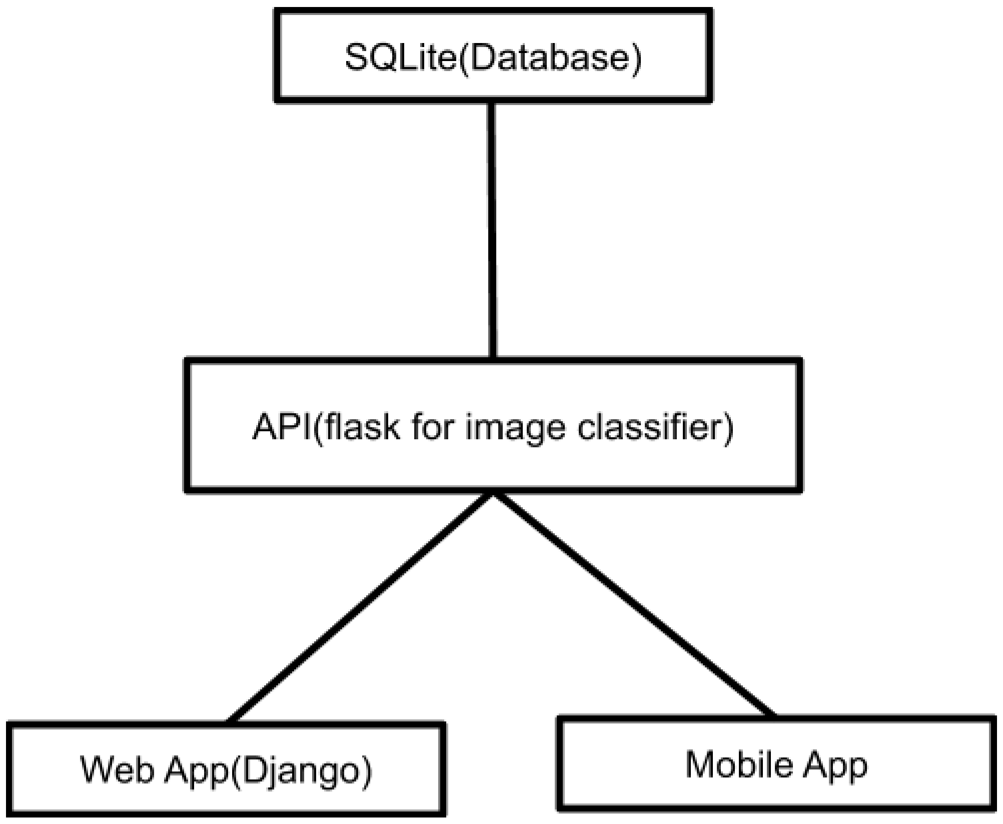


Figure 1. 11: System Architecture

4.2. API Architecture

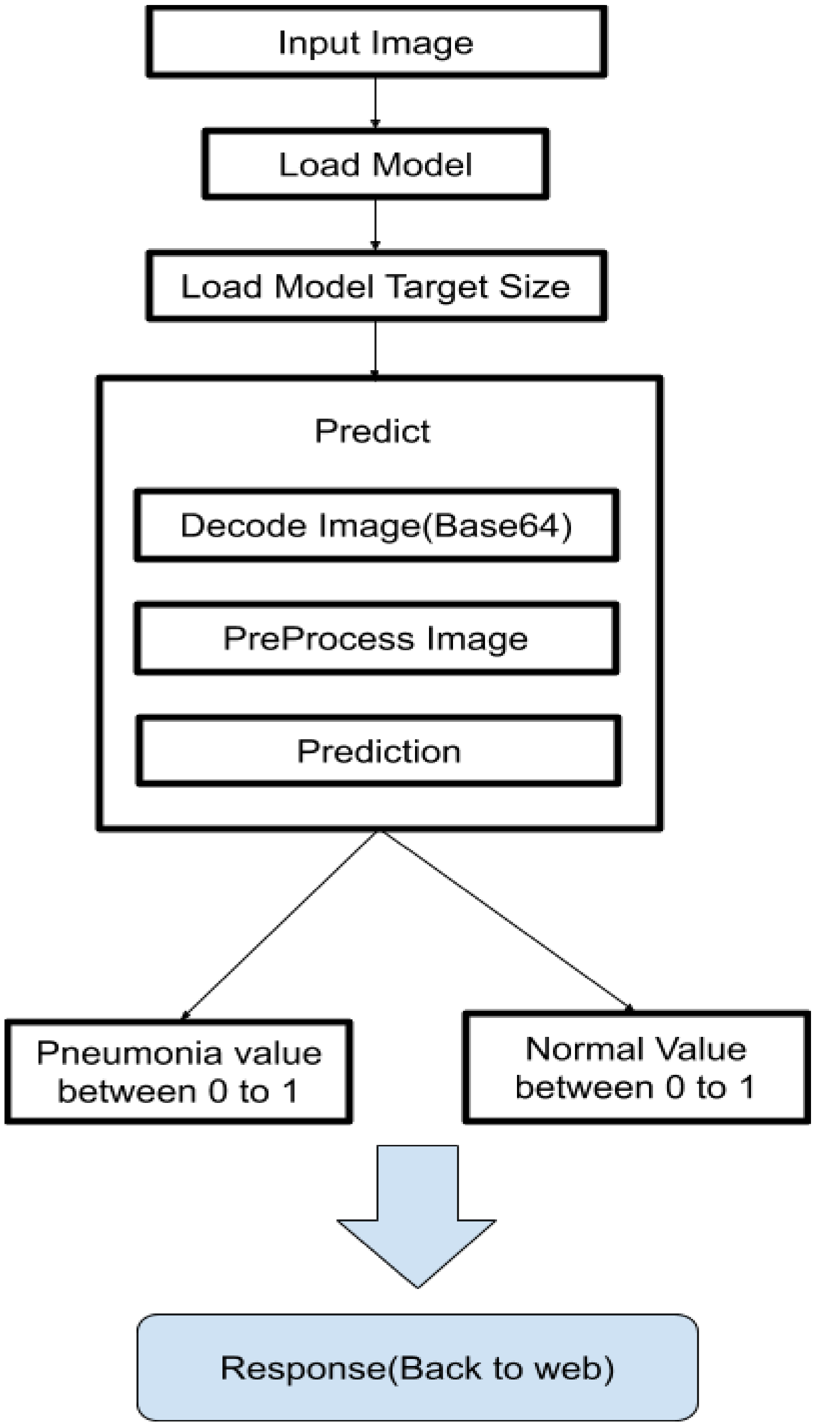
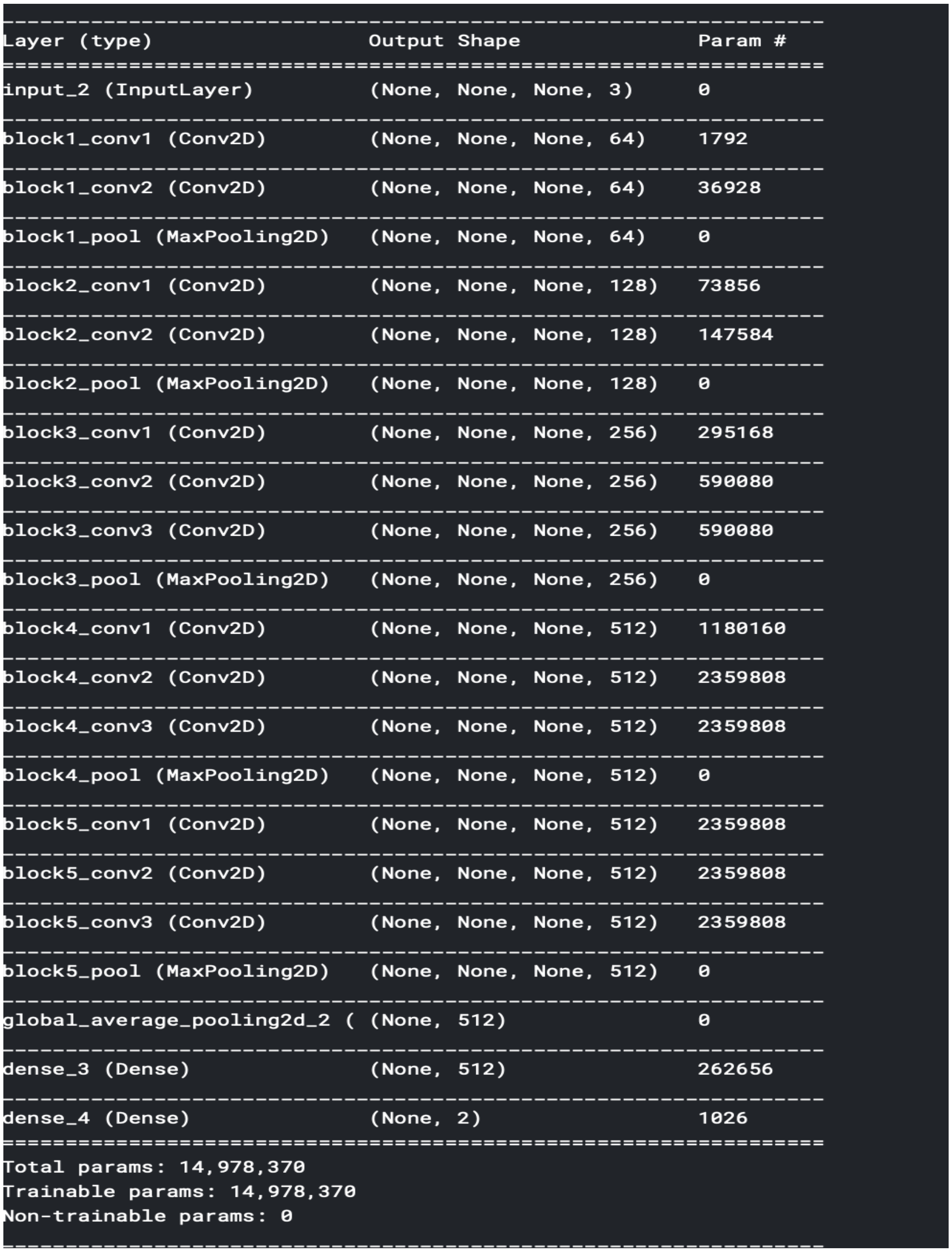


Figure 1. 12: API architecture

4.3. Customed VGG trained model architecture



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, None, None, 3)     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, None, None, 256)   295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, None, None, 256)   0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, None, None, 512)   0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
_________________________________________________________________
global_average_pooling2d_2 ( (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 1026
=================================================================
Total params: 14,978,370
Trainable params: 14,978,370
Non-trainable params: 0
_________________________________________________________________
```

Figure 1. 13: Customed VGG trained model architecture

4.4. Use Case Diagram

The use case model for any system consists of 'use cases'. Use cases represent different ways in which the system can be used by the user. A simple way to find all the use case of a system is to ask the questions "What the user can do using the system?" .The use cases partition the system behavior into transactions such that each transaction performs some useful action from the users' point of view.

The purpose of the use case to define a piece of coherent behavior without revealing the internal structure of the system. A use case typically represents a sequence of interaction between the user and the system.

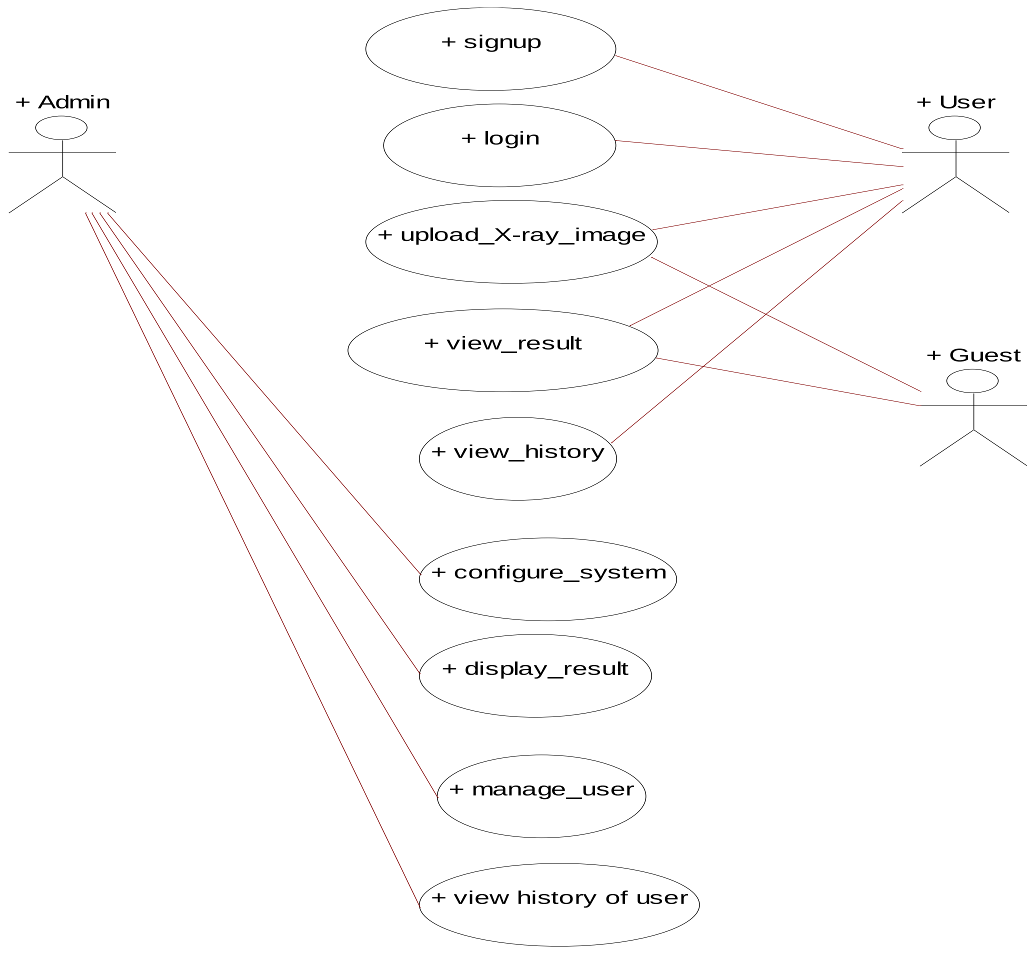


Figure 1. 14: Use case diagram of Pneumonia detection

Above, use case diagram has two actors that are admin, user and guest. Admin saves new registered user, verifies user that logins, saves the images uploaded by user or guest and displays result to user. User uses this system to detect pneumonia in the X-ray image. If user is new to the system he/she initially registers to the system then login to the system. After, logging into the system user uploads the X-ray image and views the result. Guest uses this system just to upload the X-ray image and view the result.

4.5. Class Diagram

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application. It describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages. It shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram [25].

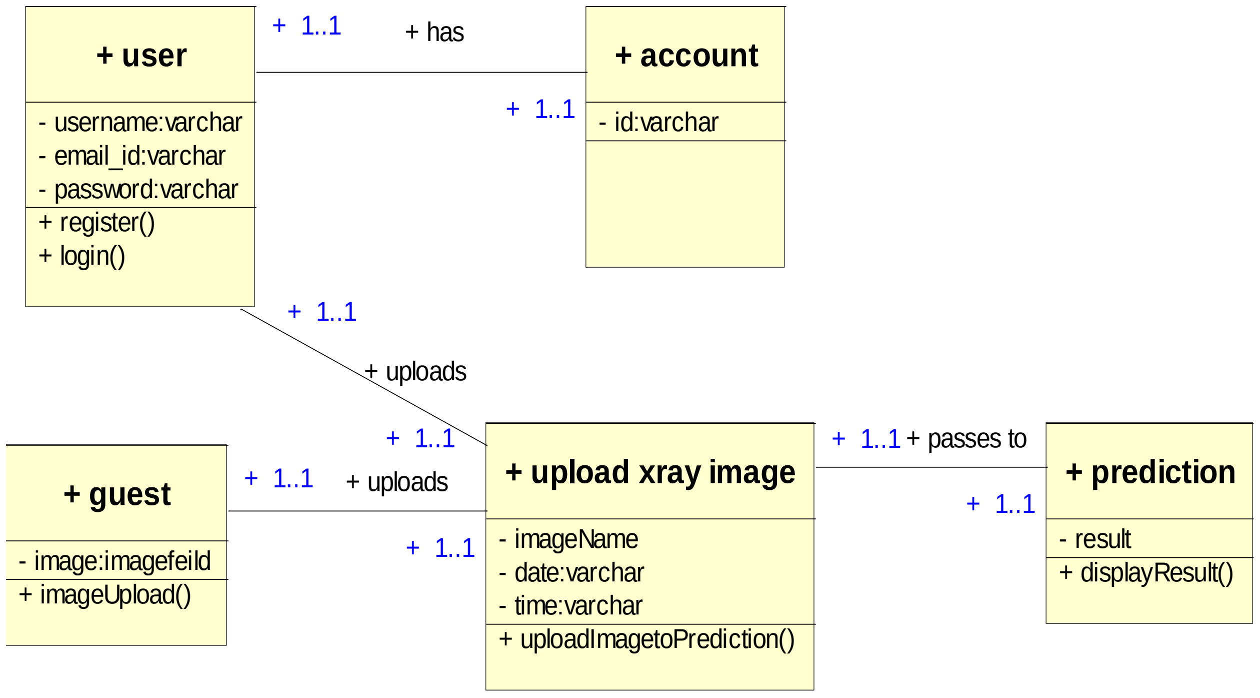


Figure 1. 15: Class diagram of Pneumonia detection.

Above, class diagram shows the relationship between the classes. Here each user has account and has unique id. New user registers to the system and then login to system. User and guest uploads X-ray image. Upload x-ray image passes the image uploaded by user to prediction. Prediction processes the image and displays the result.

4.6.Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function [26].

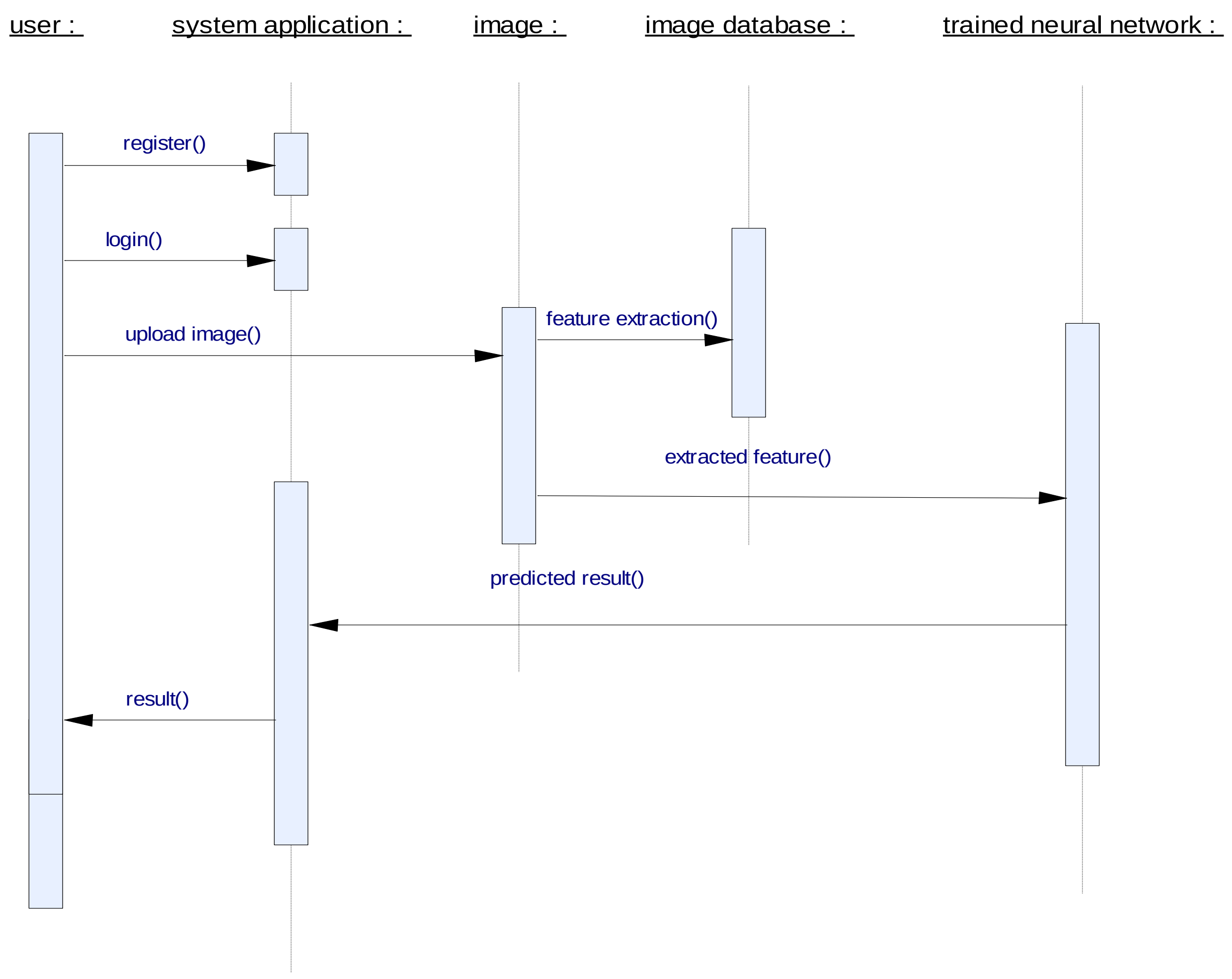


Figure 1. 16: Sequence diagram of Pneumonia detection

Above sequence diagram shows the interaction between different objects present in project. When user visits the website he/she registers to the system and then login. System application save the new registered users and verifies the logged in users. User uploads the image to system and feature is extracted from image database. The extracted feature is send to trained neural network, this predicts the pneumonia and passes the result to system application.

5. Result and Discussion

5.1. Data preparation

The data preparation involved standardizing the shape of the input images to small squares and subtracting the per-channel pixel mean calculated on the training dataset. During training, the input to our ConvNets is a fixed-size 224 × 224 RGB images. The only preprocessing we do is subtracting the mean RGB value, computed on the training set, from each pixel. One approach described involved first training a model with a fixed but smaller image size, retaining the model weights, then using them as a starting point for training a new model with a larger but still fixed-sized image. This approach was designed to speed up the training of the larger model. To obtain the fixed-size 224×224 ConvNet input images, they were randomly cropped from rescaled training images.

5.2. Data preprocessing

Images in standard datasets are in grayscale. If the images form a source are not in gray scale, those image are converted in grayscale. After conversion of image into grayscale, Image is converted into 2D array using numpy array. The converted array data in then passed into the trained model which is pipe for further stages.

5.3. Pneumonia detection with CNN

In this CNN model there are five convolutional blocks comprise of convolutional layer, max-pooling and batch-normalization. On top of five convolution blocks, flatten layer is used which is followed by four fully connected layers. Then dropout is used in between to reduce over-fitting. Activation function- ReLU is used throughout except the last layer. Sigmoid is used in last layer which is binary classification problem. Adam is used as optimization of the model and cross-entropy as loss. Before training the model two callback functions are used which 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 (reducing model capacity, increasing training data, data augumentation, regularization, dropout, etc).

Often a practical and efficient solution is to stop training when the generalization gap is getting worse (Medium, 2019).

Model is trained for 10 epochs with batch size of 32.

5.4. Feature Extraction

Out of 5216 images of chest x-ray, with a two class pneumonia 3875 images and normal 1341 images are taken. This network gains knowledge from this data, which is compiled as “weights” of the network. These weights can be extracted and then transferred to any other neural network. Instead of training the other neural network from scratch, we “transfer” the learned features.

For the feature extraction pre trained VGG is used. Output layer is removed from the pertained model because the default output layer have a 1000 classes for image classification which we don't required. Entire network is used as a fixed feature extractor for the new data set.

5.5. Building the Network

After all data are loaded and preprocessing is done convolution neural network (CNN) is defined. Firstly, a convolution layer is used with input of 62*62 size of image which extracts features from the image or parts of an image. Max pooling layer is used as the second layer of the network that reduces the dimensionality of each feature to focus on the most important elements. There are several rounds of convolution and max pooling used. Max pooling takes the input of (31,31,32).A fully connected layer that takes a flattened form of the features identified in the previous layers, and uses them to make a prediction about the image. Finally, two dense layer is used and the input is given to the final fully connected layer which results in two classes. The total parameters of the model are 813,217.

5.6. Validation

Since class score is calculated, validation is done with confusion matrix of all class score. Predicated labels and truth value used to calculate recall, precision, f1 score and f1 beta score. These are calculated for every epoch whose validation loss is below the previous one. Here, we save lowest validation loss in a variable is below the previous one. Here, we save lowest validation loss in a variable and if any next validation loss is lower than saved then recall, presion, f1 score and f1 beta score are calculated. Following are the formula in True positive, True negative, False positive and False negative for calculating Recall, Precision, F1 score and F1 beta score.

$$Recall(r) = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$Precision(p) = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$F1score = \frac{2 * p * r}{p + r}$$

$$F1_{beta}score = (1 + \beta)\frac{p * r}{\beta * p + r}$$

The beta parameter determines the weight of precision in the combined score. Beta < 1 lends more weight to precision, while beta > 1 favors recall (beta=0 considers only precision, beta = ∞ only recall).

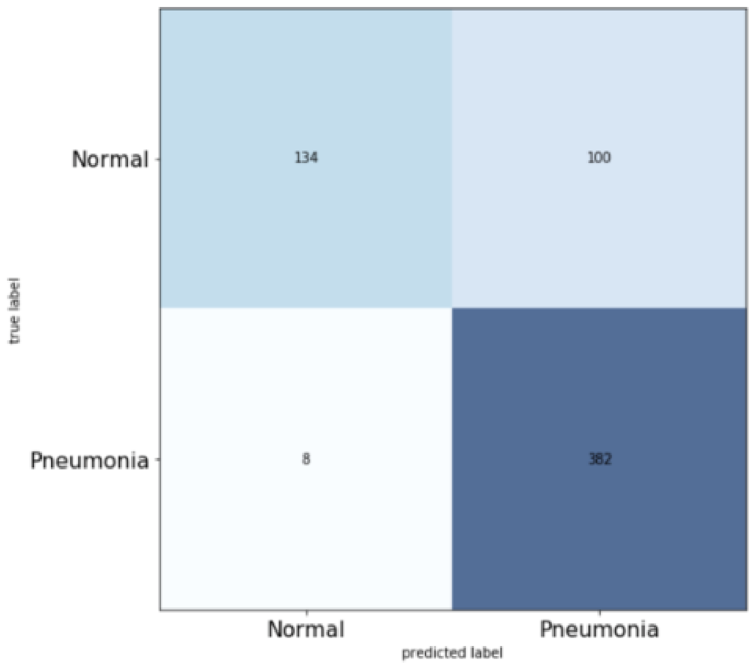


Figure: Confusion Matrix for test data.

5.7. Verification

In each 10 epoch, best model i.e when validation loss is lowest is saved with value of recall, f1 score and precision score. The final test result of the model accuracy is 89.26, precision is 88.36, recall is 95.38 and f1 score 91.73 which is shown below in figure.

6. Bibliography

1. Imran, A. (2019, Feb 13). Retrieved from Data Driven Investor: https://medium.com/datadriveninvestor/training-a-cnn-to-detect-pneumonia-c42a44101deb
2. (2017, May 31). Retrieved from healthline: https://www.healthline.com/health/pneumonia
3. *Household air pollution and health.* (2018). Retrieved from https://www.who.int/news-room/fact-sheets/detail/household-air-pollution-and-health
4. (2019). Retrieved from American lung association: https://www.lung.org/lung-health-and-diseases/lung-disease-lookup/pneumonia/symptoms-and-diagnosis.html
5. (2019). Retrieved from GE Healthcare: https://www.gehealthcare.com/feature-article/ai-could-hold-the-key-to-identifying-pneumonia-via-x-ray
6. Tiha, A. (2018, October). Retrieved from https://github.com/anjanatiha/Pneumonia-Detection-from-Chest-X-Ray-Images-with-Deep-Learning
7. Tian, Y. (2018, June 4). Retrieved from https://becominghuman.ai/detecting-pneumonia-with-deep-learning-3cf49b640c14
8. Esteva, A. (2017). Retrieved from https://scholar.google.com/scholar_lookup?title=Dermatologist-level+classification+of+skin+cancer+with+deep+neural+networks&author=E.+Andre&author=K.+Brett&author=A+Roberto+et+al.&publication_year=2017
9. M. Grewal, M. M. (2018, Jan 3). Retrieved from https://arxiv.org/pdf/1710.04934.pdf
10. (2017, july 6). Retrieved from https://scholar.google.com/scholar_lookup?title=Cardiologist-level+arrhythmia+detection+with+convolutional+neural+networks&author=R.+Pranav&author=Y.+H.+Awni&author=H.+Masoumeh&author=B.+Codie&author=Y.+N.+Andrew&publication_year=2017
11. G. Varun, P. L. (2016, Jan 12). Retrieved from https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45732.pdf
12. P. Huang, S. P. (2018, Jan). Retrieved from https://pubs.rsna.org/doi/pdf/10.1148/radiol.2017162725

13. *P. Lakhani and B. Sundaram*. (2017, April 4). Retrieved from https://pubs.rsna.org/doi/full/10.1148/radiol.2017162326

14. Benjamin Antin, J. K. (2017). Retrieved from http://cs229.stanford.edu/proj2017/final-reports/5231221.pdf

15. Ma'aitah, R. H. (2018, Aug 1). Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6093039/

16. *An Efficient Deep Learning Approach to Pneumonia Classification in Healthcare.* (2019). Retrieved from https://www.researchgate.net/publication/332049903_An_Efficient_Deep_Learning_Approach_to_Pneumonia_Classification_in_Healthcare

17. n.d.). Retrieved from techtarget: https://searchenterpriseai.techtarget.com/definition/supervised-learning

18. n.d.). Retrieved from Machine learnig masterry: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

19. (n.d.). Retrieved from medium: https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7

20. (n.d.). Retrieved from Medium: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

21. n.d.). Retrieved from Analytic Vidya: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

22. n.d.). Retrieved from missinglink: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

23. 2018, November 20). Retrieved from neurohive: https://neurohive.io/en/popular-networks/vgg16/

24. (n.d.). Retrieved from tutorialspoint: https://www.tutorialspoint.com/sdlc/sdlc_iterative_model

25. (2019). Retrieved from tutorialspoint: https://www.tutorialspoint.com/uml/uml_class_diagram.htm

26. n.d.). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/

Appendix

Code:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
from os import listdir, makedirs
from os.path import join, exists, expanduser
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model, load_model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K
import tensorflow as tf
from keras.utils.data_utils import Sequence
import sys
from PIL import *
sys.modules['Image'] = Image
print(os.listdir("../input/chest-xray-pneumonia/chest_xray/chest_xray/test"))
img_width, img_height = 224, 224
train_data = '../input/chest-xray-pneumonia/chest_xray/chest_xray/train'
test_data = '../input/chest-xray-pneumonia/chest_xray/chest_xray/test'
val_data = '../input/chest-xray-pneumonia/chest_xray/chest_xray/val'
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)
val_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    train_data,
    target_size=(224, 224),
    batch_size=16,
    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
    test_data,
    target_size=(224, 224),
    batch_size=16,
    class_mode='categorical')

validation_generator = val_datagen.flow_from_directory(
    val_data,
    target_size=(224, 224),
    batch_size=16,
    class_mode='categorical')
#import inception with pre-trained weights. do not include fully #connected layers
from keras.applications.resnet50 import ResNet50
model = ResNet50(weights='imagenet', include_top=False)
result = model.output
result = GlobalAveragePooling2D()(result)

# add a fully-connected layer
result = Dense(512, activation='relu')(result)
```

```
# and a fully connected output/classification layer
predictions = Dense(2, activation='sigmoid')(result)
inception_transfer = Model(inputs=model.input, outputs=predictions)
inception_transfer.compile(loss='categorical_crossentropy',
            optimizer=optimizers.SGD(lr=1e-4, momentum=0.9),
            metrics=['accuracy'])
import tensorflow as tf
from keras.models import Sequential
with tf.device("/device:GPU:0"):
    history_pretrained = inception_transfer.fit_generator(
        test_generator,
        steps_per_epoch=len(test_generator),
        epochs=16,
        shuffle = True,
        verbose = 1,
        validation_data = test_generator,
        validation_steps=1,
        use_multiprocessing=True,
    )
import matplotlib.pyplot as plt
# summarize history for accuracy
plt.plot(history_pretrained.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Pretrained'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history_pretrained.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Pretrained'], loc='upper left')
plt.show()
from keras.applications.vgg16 import VGG16
model = VGG16(weights='imagenet', include_top=False)
result = model.output
result = GlobalAveragePooling2D()(result)
# add a fully-connected layer
result = Dense(512, activation='relu')(result)
# and a fully connected output/classification layer
predictions = Dense(2, activation='sigmoid')(result)
inception_transfer = Model(inputs=model.input, outputs=predictions)
inception_transfer.compile(loss='categorical_crossentropy',
            optimizer=optimizers.SGD(lr=1e-4, momentum=0.9),
            metrics=['accuracy'])
import tensorflow as tf
with tf.device("/device:GPU:0"):
    history_pretrained = inception_transfer.fit_generator(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=16,
    shuffle = True,
    verbose = 1,
    validation_data = test_generator,
    validation_steps=1,
    use_multiprocessing=True,
)
import matplotlib.pyplot as plt
# summarize history for accuracy
plt.plot(history_pretrained.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Pretrained'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history_pretrained.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Pretrained'], loc='upper left')
plt.show()
scores = inception_transfer.evaluate_generator(test_generator,steps=1)
print('acc =',scores[1]*100)
inception_transfer.save('new_pneuomonia_model.h5')

from flask import Flask
from flask import request, jsonify
from flask import render_template
import base64
import numpy as np
import io
from PIL import Image
import keras
from keras import backend as k
from keras.models import Sequential
from keras.models import load_model
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator

app = Flask(__name__)
app.config["DEBUG"] = True

def get_model():
    global model
    # model = load_model('m.h5')
    model = load_model('m.h5')
    print("* MODEL LOADED !")

def target_size():
    return 224,224

def preprocess_image(image):
    image = image.resize(target_size())
    image = img_to_array(image)
    image = np.expand_dims(image,axis=0)

    return image

print(" * Loading Keras VGG Model...")
get_model()

@app.route('/predict', methods=['POST'])
def predict():
    message = request.get_json(force=True)
    encoded = message['image']
    decoded = base64.b64decode(encoded)
    image   = Image.open(io.BytesIO(decoded))
    processed_image = preprocess_image(image)

    prediction = model.predict(processed_image).tolist()

    print(prediction)
    response = {
        'prediction' : {
            'Pneumonia'   : 0,
            'Normal'   : 0
        }
    }
    return jsonify(response)

@app.route('/', methods=['GET'])
def home():
    return render_template("main.html")

app.run()
```