

General/Logo.png

Project Report

Pneumonia Detection

Author 1: Vikas Ramaswamy
Matriculation No.: 7024164
Course of Studies: Business Intelligence and Data Analytics

Author 2: Adarsh Pal
Matriculation No.: 7024418
Course of Studies: Business Intelligence and Data Analytics

Author 3: Sangram Patil
Matriculation No.: 7023678
Course of Studies: Technical Management

First examiner: Prof. Dr. Elmar Wings

Submission date: June 18, 2024

University of Applied Sciences Emden/Leer · Faculty of Technology ·
Mechanical Engineering Department
Constantiaplatz 4 · 26723 Emden · <http://www.hs-emden-leer.de>

Contents

Contents	i
List of Figures	vii
List of Tables	ix
Acronyms	xi
1. Introduction	1
1.1. Objective	1
1.2. Problem description	3
1.2.1. Problem statement	3
1.2.2. Scope of problem statement	3
1.3. Challenges	3
1.3.1. Data Quality	3
1.3.2. Bias	4
1.3.3. Data consistency	4
1.4. Proposed approach	4
2. Domain Knowledge	7
2.1. Medical imaging	7
2.2. Image recognition	7
2.3. ImageNet	8
2.4. Convolutional Neural Network	8
2.4.1. Pooling Layer	9
2.4.2. Fully-Connected Layer	10
2.4.3. CNN Training	10
2.5. Keras - Library	10
2.6. Visual studio	11
2.6.1. Installation	11
2.6.2. Requirements	12
2.7. TensorFlow Lite	13
2.8. Understanding the Domain	13
2.8.1. Problem	13
2.8.2. Data acquisition	13
2.8.3. Data quantity	15
2.8.4. Data quality	15

2.8.5. Data relevance	15
2.8.6. Outliners and Anomalies	16
3. Technical requirements	17
3.1. Software requirements	17
3.2. Hardware requirements	17
4. Knowledge Discovery in Databases Process	19
5. Data Mining - Algorithms for Pneumonia detection	21
5.1. Random Forest	21
5.1.1. Applications:	21
5.1.2. Relevance:	21
5.1.3. Hyperparameters:	21
5.1.4. Requirements:	22
5.1.5. Input:	22
5.1.6. Output:	22
5.2. Support Vector Machines (SVM)	22
5.2.1. Relevance:	23
5.2.2. Hyperparameters:	23
5.2.3. Requirements:	23
5.2.4. Input:	23
5.2.5. Output:	23
5.3. Convolutional Neural Networks	24
5.3.1. Relevance:	24
5.3.2. Hyperparameters:	24
5.3.3. Requirements:	24
5.3.4. Input:	25
5.3.5. Output:	25
5.4. Examples for the Algorithms	25
6. Development	29
6.1. Database Description	29
6.2. Dataset Description	29
6.2.1. Mendeley Chest X-Ray Dataset	29
6.2.2. Kaggle Chest X-Ray Dataset	32
6.3. Data Selection	33
6.4. Data Preparation	35
6.5. Data Transformation	35
6.6. Data Mining	36
6.6.1. Data Training	36
6.6.2. Transfer Learning	38

7. Python Package	41
7.1. Flask	41
7.1.1. Description	41
7.1.2. Installation	41
7.1.3. Manual	42
7.1.4. Example	43
7.1.5. Features:	43
7.1.6. Applications:	44
7.1.7. Further Reading	44
7.2. TensorFlow	44
7.2.1. Description	44
7.2.2. Installation	45
7.2.3. Manual	45
7.2.4. Example	46
7.2.5. Features:	46
7.2.6. Applications:	47
7.2.7. Further Reading	47
7.3. NumPy	47
7.3.1. Description	47
7.3.2. Installation	47
7.3.3. Manual	48
7.3.4. Example	49
7.3.5. Features:	49
7.3.6. Applications:	49
7.3.7. Further Reading	50
7.4. Matplotlib	50
7.4.1. Description	50
7.4.2. Installation	50
7.4.3. Manual	51
7.4.4. Example	51
7.4.5. Features	52
7.4.6. Applications	53
7.4.7. Further Reading	53
8. Development Environment: PyCharm	55
8.1. Version	55
8.2. PyCharm Benefits	55
8.3. Main Functions	55
8.4. Subfunctions	56
8.5. Installation	56
8.6. Configuration	56
8.6.1. General	56
8.6.2. Special	56
8.7. First Steps	57

8.8. Program "Hello World"	57
8.8.1. Description	57
8.8.2. Manual	57
9. Monitoring and Evaluation	59
9.1. Monitoring of the Model	59
9.1.1. Importance of updating the data	59
9.1.2. Privacy is critical	60
9.1.3. Robustness	61
9.1.4. Monitoring in the KDD process	61
9.2. Evaluation of Pneumonia Detection	62
9.2.1. Application	64
9.2.2. Result	64
9.3. General	65
10. Testing	69
10.1. Function	69
10.2. Class	70
10.3. Parts	70
10.4. Automation	71
10.5. Documentation	72
11. Development to Deployment	73
11.1. Tools and Installation	73
11.2. Saving the Model	75
11.3. File Structure	76
11.4. Loading Model	76
11.5. Description Of Process	77
12. Deployment	79
12.1. User Interface	79
12.2. Work flow	79
13. Monitoring the Model	83
13.1. Plan	83
13.2. Process Description	83
13.3. New Data	84
13.4. Checks	84
13.5. Functions	84
13.6. Program	85
14. Programme Flow	87
15. Application	89
15.1. Operating System	89

Contents	v
15.2. Constraints	89
15.3. Input Data	91
15.4. Conclusions	91
16. Open Questions	93
17. Conclusion	95
17.1. Conclusion	95
17.2. Next Step	95
17.3. To Do	97
17.4. Unanswered points	98
A. Appendix	99

List of Figures

1.1. Schematic Classification of infections in lungs. . .	2
1.2. X-ray Images 1) COVID-19, 2) viral pneumonia, 3) normal and 4) bacterial pneumonia.	5
2.1. A CNN sequence to classify handwritten digits	9
2.2. Pooling Layer in CNN	10
2.3. TensorFlow Lite Workflow.	14
4.1. KDD Process Workflow	19
6.1. KDD approach flow chart	30
6.2. Chest X-ray of (a) Healthy person and (b) Pneumonia	31
6.3. Transfer learning architecture	38
7.1. Flask server	43
7.2. NumPy Output	49
7.3. Matplotlib Output	52
9.1. Flowchart for Data updating	60
9.2. Process monitoring flowchart	63
10.1. pytest	69
10.2. Predict Pneumonia	70
10.3. Accuracy	70
10.4. model-loading	71
11.1. Saving Model Architecture	75
11.2. Saving Model For Python Scrip	76
11.3. Loading The Model	77
11.4. Description Of Process	78
12.1. Road Map For Web App	80
12.2. Web Page	80
12.3. Result	80
12.4. Deployment approach flow chart	81
15.1. User Interface	90

List of Tables

2.1. Visual Studio Requirements for MacOS	12
2.2. Visual Studio Requirements for Windows	12
2.3. Visual Studio Requirements for Linux	12
3.1. Software Bill of Materials for Pneumonia Detection	17
3.2. Hardware Bill of Materials for Pneumonia Detection	17
12.1.	82

Acronyms

CNN Convolutional Neural Network

KDD Knowledge Discovery in Databases

SVM Support Vector Machines

TFLM TensorFlow Lite Microcontrollers

1. Introduction

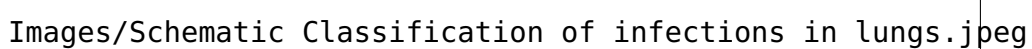
Pneumonia is a potentially fatal respiratory infection caused through bacteria, fungi or viruses that can cause the air sacs in one or both lungs to become inflamed. The likelihood of recovery and danger of complications can both be greatly increased and decreased, respectively, by early pneumonia diagnosis. Although chest X-ray imaging is a standard diagnostic method for identifying pneumonia, its interpretation can be arbitrary and error-prone.

1.1. Objective

The goal of designing a pneumonia detection system based on CNN and transfer learning is to evaluate patient chest X-ray pictures to detect the presence of pneumonia. This system extracts and analyzes X-ray image attributes using deep learning methods and computer vision techniques. The major purpose of this system is to enable reliable and efficient identification of pneumonia from X-ray pictures, which may then be used to aid medical professionals in making diagnosis in clinical situations.[Lim:2022]

The project's detailed objectives are as follows:

1. To create a pneumonia detection system that can detect the presence of pneumonia in chest X-ray pictures correctly and effectively.
2. To boost the detection system's accuracy, employ transfer learning to leverage pre-trained models such as Inception and ResNet.
3. To automate the process of detecting pneumonia, decreasing the workload of medical workers and reducing the chance of errors.
4. To provide a vital tool for medical practitioners to aid in the diagnosis of pneumonia, allowing patients to receive faster and more accurate treatment.

The image is a schematic diagram titled "Schematic Classification of infections in lungs". It is currently blank, with only the title text visible in the center of the frame.

Images/Schematic Classification of infections in lungs.jpeg

Figure 1.1.: **Schematic Classification of infections in lungs.**
Reference:[Lim:2022]

1.2. Problem description

1.2.1. Problem statement

Pneumonia is a serious and potentially fatal infection of the lungs. It is a leading cause of hospitalization and death, especially in young children, the elderly, and those with compromised immune systems. Early diagnosis and treatment of pneumonia are crucial for improving patient outcomes and lowering the risk of complications.

On the other hand, identifying pneumonia in chest X-ray images is a challenging task that demands particular training and skill. The slight changes in appearance between healthy and diseased lungs might make a radiologist's diagnosis of pneumonia difficult. Furthermore, the accuracy of the diagnosis can vary based on the radiologist's level of experience. [Zhou:2018].

1.2.2. Scope of problem statement

The goal of constructing a ML model that can effectively detect pneumonia from chest X-ray pictures is to enhance patient outcomes as well as aid in diagnosis. The model can be linked into existing medical systems to increase diagnosis efficiency and accuracy, particularly in places with limited access to medical expertise. Furthermore, the model can be utilized for research purposes to evaluate massive datasets of chest X-ray images, assisting in the identification of patterns and potential risk factors related with pneumonia. Overall, the problem statement's scope is to create an AI model that can help enhance pneumonia diagnosis, treatment, and research.

1.3. Challenges

1.3.1. Data Quality

One of the main challenges in developing a model for pneumonia detection from chest X-ray images is the high variability of chest X-ray images. Chest X-rays can be taken from different angles, with varying levels of exposure, and with different imaging protocols. This variability can make it difficult for models to learn generalizable features and accurately classify images. In a study by [Wang:2017], it was found that variability in imaging protocols and exposure levels can lead to decreased performance of models in classifying chest X-ray images.

1.3.2. Bias

When developing models for pneumonia detection there can be a potential bias in labeled datasets. The CXR dataset, which is commonly used for this task, has been shown to have class imbalance, with a much larger number of normal chest X-ray images than pneumonia images. This can lead to models being biased towards classifying images as normal and may lead to a higher rate of false negatives. In a study by [Ibrahim:2021], the authors found that a deep learning model trained on the CXR dataset had a high false-negative rate for pneumonia detection.

1.3.3. Data consistency

The biggest challenge would be related to model consistency on the prediction as most of the lung disorders can be observed similarly over a X-ray image. The figure 1.2.: shows 1) COVID-19, 2) viral pneumonia, 3) normal and 4) bacterial pneumonia respectively.

1.4. Proposed approach

In order to identify pneumonia in chest X-ray pictures, this project attempts to build a deep convolutional neural network utilizing transfer learning. By utilizing transfer learning, the model can take advantage of the characteristics discovered from a sizable dataset like ImageNet, which can enhance the model's performance on the medical imaging dataset. Popular pre-trained model InceptionV3 has demonstrated successful performance in image recognition challenges, making it an appropriate contender for our approach.

The approach involves using Python programming language and libraries like Keras, TensorFlow, Inception, and ImageNet to build, train and evaluate the deep learning model. The solution is applicable in the medical imaging field, specifically in image recognition and classification tasks.

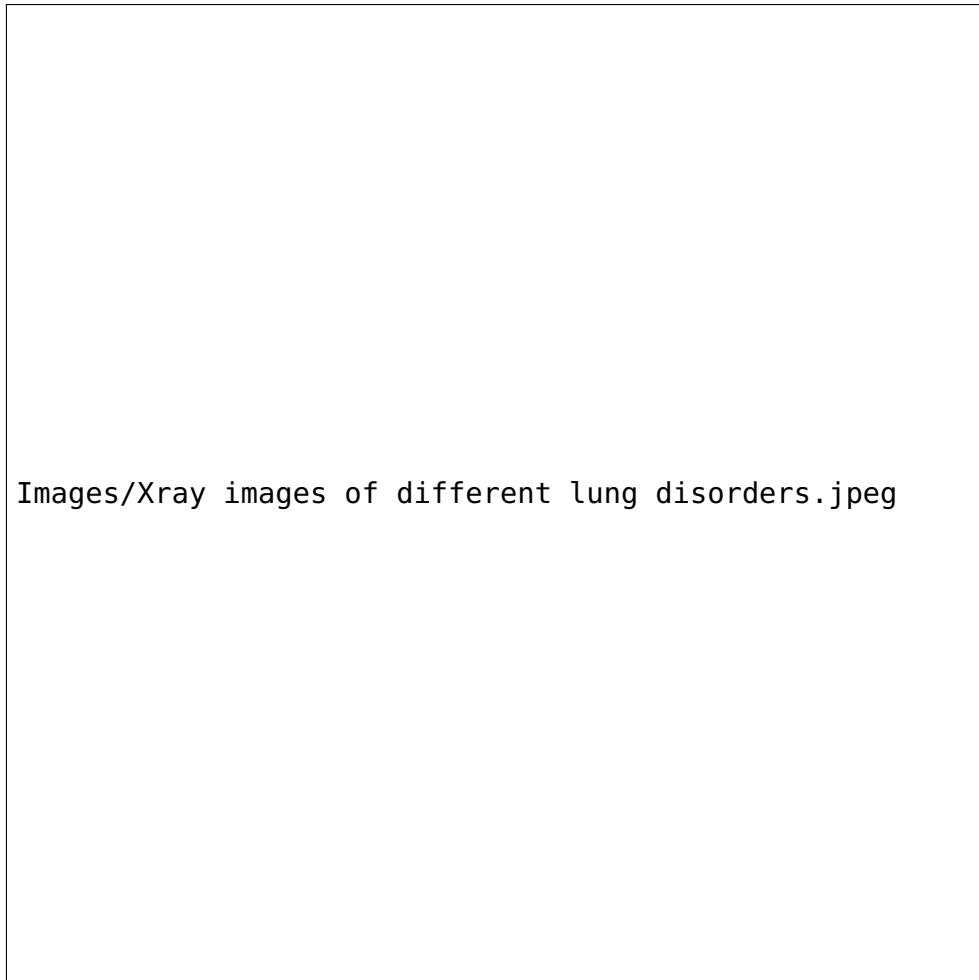


Figure 1.2.: **X-ray Images 1) COVID-19, 2) viral pneumonia, 3) normal and 4) bacterial pneumonia.**
Reference:[Lim:2022]

2. Domain Knowledge

2.1. Medical imaging

Creating visual representations of the human body or its components for diagnostic, therapeutic, and research reasons is known as medical imaging. Medical practitioners utilize the images generated to evaluate the composition and operation of tissues and organs, to identify illnesses, and to track the effectiveness of treatment.

Anatomical imaging and functional imaging are the two broad categories that medical imaging falls under. X-rays, computed tomography (CT), and magnetic resonance imaging (MRI) are just a few of the anatomic imaging methods that can produce detailed images of the inside organs and tissues of the body. Positron emission tomography (PET) and single-photon emission computed tomography (SPECT) are examples of functional imaging techniques that can provide details about the physiological functions and metabolic activity of tissues and organs. [Deserno:2011]

2.2. Image recognition

The technique of locating and classifying objects inside digital photographs is known as image recognition, sometimes known as image classification. It entails the use of computer algorithms to examine and comprehend an image's content and has a wide range of uses, including the recognition of faces in photographs, the detection of objects in satellite imagery, and the diagnosis of medical disorders from images of patients. [Szeliski:2022]

Deep learning algorithms, particularly convolutional neural networks (CNNs), have shown great promise in image recognition tasks, including pneumonia detection. CNNs are designed to learn from the data by automatically extracting relevant features from the images, making them well-suited for complex image recognition tasks. [Kermany:2018]

2.3. ImageNet

A sizable image dataset called ImageNet is used to train and assess computer vision models. Over 1.2 million photos in 1000 different classes make up the collection. The computer vision field now uses ImageNet as a benchmark dataset for image classification tasks.

The dataset was created by a team of researchers at Stanford University by [Deng:2009], and was first introduced in 2009. The images in the dataset were collected from the web and then manually labeled by a team of workers. The labels were then verified by experts to ensure accuracy.

2.4. Convolutional Neural Network

Deep learning neural networks like CNN (Convolutional Neural Networks) are frequently utilized for image processing applications like object and picture detection. It was motivated by biological processes that take place in the layers of neurons that make up the visual cortex in both humans and animals, where each layer extracts increasingly intricate characteristics from the visual input.

A network of neurons in CNN processes an input image by feeding it into a series of convolutional operations to extract low-level features, followed by pooling operations to lower the dimensionality of the feature maps. After this process, fully connected layers are applied to the output to conduct high-level regression or classification.

CNNs have been shown to be highly effective in image classification tasks, achieving state-of-the-art performance on many benchmark datasets. They have been applied in a wide range of applications, from self-driving cars to medical image analysis. [KE:2022]

The convolution neural network consists of three important layers which are as follows

- Convolution Layer
- Pooling Layer
- Fully-connected Layer

The complexity of the CNN increases with each layer as it identifies larger portions of image. The beginning layers focuses on simple features like edges, borders and colors. With the progression of image data through the different layers of CNN, the CNN starts to identify larger shapes and elements present in the object. This stops when the CNN identifies the intended object. [IBM:2020]

Figure 2.1.: A CNN sequence to classify handwritten digits
Reference: [KE:2022]

The major part of the computation takes place at this layer. The components of the convolution layer are the input data, filters and feature map. The filter is also known as kernel or a feature detector. The feature detector moves through the recurring fields of the picture, while checking if the feature is present. This entire process is termed as convolution.

The Kernel is a two-dimensional array of biases or weights, which represents a part of the picture. The kernel can be of different sizes. This size of the kernel influences the size of receptive field. A dot product is computed between the kernel and the filter. This value of the dot product is then provided into an output array. Then the kernel moves to the next stride, this process is repeated until the kernel moves across the full picture. The final output is termed as a convolved feature or a feature map. The initial feature map only captures Low-Level information like the colour gradient and edges. The progressing feature maps captures high-level information which results in the network that helps classification of individual features present in the image. [IBM:2020]

2.4.1. Pooling Layer

Pooling conducts a dimensionality reduction of the feature map. This is done by reducing the number of input parameters. Here the CNN retains the important features of the map which is required for the classification. This process is termed as down sampling.

In the pooling operation a filter without any weights is used to sweep across the whole input. Here an aggregation function is applied to the values of the receptive field by the kernel. The values of the aggregation function are used to populate the output array. [IBM:2020]

- **Max Pooling:** As the kernel is swept through the input, the kernel identifies the pixel with the highest value and transfers it into the output array. This helps in retaining the most important features present in the feature map.
- **Average Pooling:** As the kernel is swept through the input, it computes the average value of the receptive field and transfers it into the output array.

Figure 2.2.: Pooling Layer in CNN
Reference: [KE:2022]

Pooling has benefits such as, limiting the risk of overfitting, size reduction, noise suppression and reduction in complexity of the feature map. These result in improved efficiency of the feature map.

2.4.2. Fully-Connected Layer

The fully-connected layer is the final layer of the CNN. Here each node of the output layer directly connects to a node of the previous layer. The classification process carried out in this layer based upon the previous layers and their appropriate filters. A Soft-Max function assigns decimal probabilities from 0 to 1 based on the appropriate classification of the inputs. [IBM:2020]

2.4.3. CNN Training

There are six large steps that need to be taken into account, each one has its own characteristics and functions. Also, the inputs and output from each step needs to be taken into account as a priority to understand the functionality.

The main steps to create and train the Convolutional Neural Network (CNN) are the following:

1. Preparation (loading the libraries and settings).
2. Loading the training images.
3. Generate training data and test data.
4. Definition and structure of the network.
5. Train the model.
6. Storing the neural network.

2.5. Keras - Library

Keras is an open-source neural network library written in Python. It is intended to make building and training complex models simple as well as to enable quick experimentation with deep neural networks. Deep learning

model prototypes may be created fast because to Keras' friendly API. It is based on TensorFlow and is compatible with Theano and Microsoft Cognitive Toolkit among other backend engines. [Chollet:2015]

Keras was developed by François Chollet, a Google engineer, and was first released in March 2015. Since then, it has become one of the most popular deep learning libraries, due to its ease of use, flexibility, and powerful features.

2.6. Visual studio

Visual Studio is a complete integrated development environment (IDE) for creating web, Android, and iOS applications. It offers tools for debugging, testing, and the deployment of programs and supports a number of programming languages, including C, C++, and Python.

2.6.1. Installation

Please find the step by step installation for Visual studio below:

1. Download the Visual Studio installer from the Microsoft website.
2. Run the installer executable file and select "Install".
3. Choose the desired workloads (the type of development you will be doing, such as .NET desktop development or web development) and components to install.
4. Review and modify individual component settings as needed.
5. Click on "Install" to start the installation process.
6. Follow the prompts and accept the license agreements.
7. Visual Studio will download and install the selected components.
8. Once the installation is complete, click on "Launch" to open Visual Studio.
9. The first time you open Visual Studio, you will be prompted to sign in with a Microsoft account or create a new one. You can choose to skip this step if you prefer.
10. You can start creating your projects and building your applications in Visual Studio.

Now have Visual Studio installed and ready to use.

2.6.2. Requirements

The requirements to install the visual studio is listed in the tables below.

Table 2.1.: Visual Studio Requirements for MacOS

Requirement	Details
Operating System	macOS 10.14 or later
Processor	Intel processor
RAM	4 GB minimum, 8 GB recommended
Disk Space	10 GB minimum
.NET Core	.NET Core 2.2 SDK or later
Xcode	Xcode 11.3 or later

Table 2.2.: Visual Studio Requirements for Windows

Requirement	Details
Operating System	Windows 10 version 1507 or later
Processor	1.8 GHz or faster processor. Quad-core or better recommended
RAM	2 GB minimum, 8 GB recommended
Disk Space	20 GB minimum
.NET Core	.NET Core 2.2 SDK or later
Visual Studio Installer	Latest version

Table 2.3.: Visual Studio Requirements for Linux

Requirement	Details
Operating System	Ubuntu 16.04 LTS or higher
Processor	2 GHz or faster processor. Quad-core or better recommended
RAM	4 GB minimum, 8 GB recommended
Disk Space	10 GB minimum
.NET Core	.NET Core 2.2 SDK or later

2.7. TensorFlow Lite

To create and train deep neural network machine learning models, Google created the open-source software package known as TensorFlow. Another open-source deep learning framework made exclusively for on-device computing in edge devices is called TensorFlow Lite. A scaled-down version of TensorFlow is TensorFlow Lite. The developers can use TensorFlow Lite to run their trained models on edge devices, desktop computers, and mobile devices. On a model that has previously been trained, predictions can be made using TensorFlow Lite. [Boesch:2022]

TensorFlow Lite Microcontrollers (TFLM) were created to enable the execution of deep learning and machine learning models on microcontrollers with memory capacities as low as a few kilobytes. By addressing both the fragmentation issues and the efficiency demands imposed by embedded-system resource constraints, TFLM enables cross-platform interoperability. The TFLM framework uses an original interpreter-based strategy that offers flexibility while resolving the aforementioned issues. [Robert:2021]

2.8. Understanding the Domain

2.8.1. Problem

The difficulty in discriminating between healthy and infected lungs in pneumonia detection using X-ray imaging is an issue. To diagnose pneumonia, radiologists must look for visual clues such as infiltrates, fluid in the lungs, and dense areas in the lungs, but these symptoms are not always apparent or easy to notice, especially for less experienced radiologists. As a result, establishing an accurate AI model to aid in diagnosis has the potential to enhance patient outcomes.

2.8.2. Data acquisition

The focus of developing an AI model that can accurately detect pneumonia from chest X-ray images is not only to aid in diagnosis but also to improve patient outcomes. X-ray pictures utilized in pneumonia identification projects might come from a variety of places, including hospitals, medical research institutes, and publicly available databases. [Wang:2017] The Chest X-Ray Images (CXR) dataset, the ChestX-ray14 dataset, and the NIH Chest X-ray dataset are three extensively used datasets. The CXR dataset comprises over 100,000 frontal-view X-ray images of various chest diseases, such as pneumonia, whereas the ChestX-ray14 dataset contains



Figure 2.3.: TensorFlow Lite Workflow.
Reference:[Khandelwal:2020]

112,120 frontal-view X-ray images of 14 distinct pathologies, such as pneumonia. Over 100,000 chest X-ray pictures with pathology diagnoses, including pneumonia, are included in the NIH Chest X-ray Collection. We are considering the dataset available on Kaggle, which is specially used for pneumonia detection. It has images classified as pneumonia and normal.

2.8.3. Data quantity

The quantity of data required for a pneumonia detection project depends on the complexity of the AI model being developed and the desired level of accuracy. In general, a larger dataset can help improve the accuracy of the model. The CXR dataset contains over 100,000 images, which is a substantial amount of data for training an AI model. However, smaller datasets can also be used with techniques such as data augmentation to increase the effective size of the dataset. The dataset kaggle mentioned by [Kermay:2018] which we are using has 5,863 chest X-ray images classified as pneumonia and normal.

2.8.4. Data quality

The quality of the X-ray images used in pneumonia detection projects can vary depending on the imaging equipment, the image resolution, and other factors. Poor-quality images may be unusable or negatively impact the accuracy of the AI model. Therefore, it is essential to ensure that the X-ray images used in the project are of high quality and meet the necessary standards. Chest X-rays can be taken from different angles, with varying levels of exposure, and with different imaging protocols. This variability can make it difficult for models to learn generalizable features and accurately classify images. [Rajpurkar:2017]

2.8.5. Data relevance

The accuracy of the AI model is dependent on the relevancy of the data utilized in pneumonia detection initiatives. To avoid misunderstandings during the training process, only photographs of the chest X-ray should be picked, and images with inaccurate classifications should be eliminated. To ensure that the AI model is correct for the target population, the data must also be representative of the population being examined.

For our dataset, all chest X-ray imaging was done as part of the regular clinical treatment provided to patients. All chest radiographs were initially

screened for quality control before being removed from the analysis of the chest x-ray images. Before the photos could be used to train the AI system, they were graded by two experienced doctors. A third expert also reviewed the evaluation set to make sure there were no grading mistakes.[**Kermany:2018**]

2.8.6. Outliners and Anomalies

Outliers are data points that differ dramatically from the rest of the data in the dataset. Outliers in pneumonia detection projects may include X-ray images with severe pathologies that are unrelated to the study or images with inaccurate labeling. Outliers can reduce the accuracy of the AI model, so they should be detected and eliminated from the dataset. There can be issues which are related to X-ray imaging because of angle and also visibility can cause outliers.[**Wang:2020**]

Anomalies are data points that differ dramatically from the rest of the data yet are still relevant to the project. Anomalies in pneumonia detection projects may include X-ray images with distinct diseases that are nevertheless relevant to the study or images with slightly different labels. Anomalies may necessitate additional preprocessing or specific handling throughout the training process to guarantee that they do not impair the AI model's accuracy. The different disorders related to the lungs can also be detected and can act as anomalies.

3. Technical requirements

3.1. Software requirements

Table 3.1.: Software Bill of Materials for Pneumonia Detection

Software/Package	Open Source/License	Version
VS Code	Open Source	1.64.2
Anaconda-Navigator	Open Source	2.1.4
Conda	Open Source	4.11.0
Keras	Open Source	2.8.0
TensorFlow	Open Source	2.8.0

Reference:Author

3.2. Hardware requirements

Table 3.2.: Hardware Bill of Materials for Pneumonia Detection

Hardware Name	Description	Quantity
Computer Display	1080p (1920x1080) resolution or higher	1
Keyboard	Standard American/German language	1
Mouse	Standard 3 button wired or wireless	1
Laptop	Capable of running deep learning algorithms	1

Reference:Author

4. Knowledge Discovery in Databases Process

The Knowledge Discovery in Databases (KDD) is a general term for the process of discovering knowledge in data and focuses the "high level" application of certain data mining algorithms. KDD is used in various fields like machine learning, statistics, artificial intelligence, pattern recognition, data visualization etc. [Fayyad:1996]

KDD is the automated discovery process and the approach has a primary objective to extract knowledge from massive databases that includes the procedure for using a database, including any necessary preprocessing, subsampling and modifications. It also includes analysing and maybe interpreting patterns in order to determine the knowledge. It has been used widely in banking industry, gesture recognition, analysing sales information, medical field, climate analysis etc.[Fayyad:1996]

And here we are applying KDD process for the medical field and for that we must first outline the experiment's goals and objectives. After the goals have been established, the KDD approach is used to collect, create and predict the results as required. The overall process includes:

1. **Database:** Creating a target dataset. It can be anything like text, image, voice etc.
2. **Selection:** Selecting a dataset on discovery is to be performed.
3. **Data Preparation:** Data cleaning and preprocessing. It includes removal of noise and outliers. Filling the missing values is also included here.
4. **Data Transformation:** To produce patterns that are simpler to understand, data transformation is a crucial data preprocessing procedure that must be applied to the data. It transforms the data into clean, useful data by altering its format, structure, or values.

Figure 4.1.: KDD Process Workflow
Reference:[Wings:2023]

5. **Data Mining:** Searching for interesting patterns in a specific representational form. It includes creating predictive models by using previous database pattern.
6. **Model Patterns:** Detection of sequential patterns and relation between consecutive periods.
7. **Evaluation,Verification:** Evaluation and verification of desired results. Verify the deviation from the desired results and correct it.

The detailed description of each step is described in the chapter Development. The process flow of the KDD can be seen in figure 3.1.:

5. Data Mining - Algorithms for Pneumonia detection

5.1. Random Forest

Random Forest is an ensemble learning algorithm that combines multiple decision trees to make predictions. It creates a set of decision trees on randomly selected data samples, and the final prediction is made by aggregating the predictions of individual trees. Each tree in the forest independently makes a prediction, and the final result is determined by majority voting or averaging the predictions. [Breiman:2001]

5.1.1. Applications:

Random Forest has a large area of application. However, some of the applications are:

1. **Pneumonia detection:** Random Forest can be used to classify chest X-ray images as pneumonia or non-pneumonia cases.
2. **Medical diagnosis:** It can assist in diagnosing various medical conditions by analyzing relevant features.
3. **Image classification:** Random Forest can be employed for classifying images into different categories based on their features.

5.1.2. Relevance:

Random Forest is particularly suitable for pneumonia detection due to its ability to handle high-dimensional data, provide feature importance ranking, and effectively handle imbalanced datasets. It can handle both classification and regression tasks and is robust against overfitting.

5.1.3. Hyperparameters:

1. **Number of trees:** The number of decision trees to be created in the random forest.

2. **Maximum depth:** The maximum depth allowed for each decision tree in the forest.
3. **Minimum samples split:** The minimum number of samples required to split a node further.
4. **Maximum features:** The number of features to consider when looking for the best split.

5.1.4. Requirements:

1. **Labeled training data:** A dataset with chest X-ray images and corresponding labels indicating whether the image represents a pneumonia case or not.
2. **Preprocessing:** The input data should be preprocessed, including image resizing, normalization, and feature extraction.

5.1.5. Input:

Training data consists of chest X-ray images, represented as feature vectors, along with their corresponding labels indicating pneumonia or non-pneumonia cases.

5.1.6. Output:

The Random Forest algorithm outputs predicted class labels for unseen chest X-ray images, indicating whether they are classified as pneumonia or non-pneumonia cases.

5.2. Support Vector Machines (SVM)

Support Vector Machines (SVM) is a supervised machine learning technique that is used for classification and regression applications. SVM separates data points by locating the best hyperplane in a multidimensional feature space. It can handle both linear and non-linear data and seeks to optimize the margin between classes.[cortes:2009]

5.2.1. Relevance:

SVM is particularly relevant for pneumonia detection as it can handle complex datasets, especially those with clear margin separation. It is effective in dealing with high-dimensional data and can provide accurate predictions when properly tuned.

5.2.2. Hyperparameters:

1. **Kernel type:** SVM supports various kernel functions (e.g., linear, polynomial, radial basis function) that transform the data to a higher-dimensional space.
2. **Regularization parameter (C):** It controls the trade-off between maximizing the margin and minimizing the classification errors.
3. **Gamma parameter:** It determines the influence of each training example on the decision boundary for non-linear kernels.

5.2.3. Requirements:

1. **Labeled training data:** A dataset with chest X-ray images and corresponding labels indicating whether the image represents a pneumonia case or not.
2. **Preprocessing:** The input data should be preprocessed, including image resizing, normalization, and feature extraction.

5.2.4. Input:

Training data consists of chest X-ray images, represented as feature vectors, along with their corresponding labels indicating pneumonia or non-pneumonia cases.

5.2.5. Output:

The SVM algorithm outputs predicted class labels for unseen chest X-ray images, indicating whether they are classified as pneumonia or non-pneu-

monia cases.

5.3. Convolutional Neural Networks

Deep learning methods known as CNN) are extensively utilized for image classification applications. CNNs are made up of numerous convolutional and pooling layers, followed by fully linked classification layers. They are intended to learn hierarchical representations from incoming photos automatically.[[lecun:1998](#)]

5.3.1. Relevance:

CNNs are useful for detecting pneumonia because they excel at learning spatial hierarchies of data from images. They can detect complicated patterns and relationships in medical imagery, allowing them to accurately diagnose pneumonia.

5.3.2. Hyperparameters:

1. **Number and size of filters:** CNNs use filters for feature extraction. The number and size of filters determine the complexity and depth of the network.
2. **Learning rate:** It controls the step size during the gradient descent optimization process.
3. **Dropout rate:** Dropout is a regularization technique used to prevent overfitting. The dropout rate specifies the probability of randomly dropping out nodes during training.
4. **Batch size:** The number of training examples used in each iteration of training.

5.3.3. Requirements:

1. **Labeled training data:** A dataset with chest X-ray images and corresponding labels indicating whether the image represents a pneumonia case or not.

2. **Computational resources:** Training CNNs can be computationally intensive, so access to GPUs or other high-performance hardware is beneficial.

5.3.4. Input:

Images of chest X-rays, typically in the form of 2D arrays or tensors.

5.3.5. Output:

The CNN algorithm outputs predicted class labels for unseen chest X-ray images, indicating whether they are classified as pneumonia or non-pneumonia cases.

5.4. Examples for the Algorithms

Random Forest example:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the preprocessed dataset (features and labels)
X, y = load_data()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, max_depth=10)

# Train the classifier
rf.fit(X_train, y_train)

# Make predictions on the test set
predictions = rf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

Support Vector Machines example:

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the preprocessed dataset (features and labels)
X, y = load_data()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create an SVM classifier with a linear kernel
svm = SVC(kernel='linear', C=1.0)

# Train the classifier
svm.fit(X_train, y_train)

# Make predictions on the test set
predictions = svm.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```


Convolutional Neural Network example:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the preprocessed dataset (images and labels)
X, y = load_data()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a CNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_width,
image_height, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy * 100))

# Make predictions on the test set
predictions = model.predict(X_test)

# Convert predictions to class labels
predicted_labels = [1 if prediction > 0.5 else 0 for prediction in predictions]
```


6. Development

Before any actual implementation, a number of subprocesses that are relevant to each project's goals must be understood and defined in order for the KDD methodology to be used in a practical way. To gain a deeper understanding of the project's data environment, we will discuss the database. The purpose of this chapter is to discuss the steps involved in each KDD process as well as how the KDD process is executed. The proposed approach for the workflow is shown in figure 4.1:

6.1. Database Description

The database used in the project contains chest X-ray images (JPEG) of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. The dataset is organized into 3 folders train, test, val for training, testing and validating respectively. Further each folder contains subfolders for each image category for Pneumonia and Normal. There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).[Wang:2017]

6.2. Dataset Description

There are 2 categories and 5,863 X-Ray images in JPEG format. The total file is around 2GB in size. From retrospective cohorts of children patients aged one to five at the Guangzhou Women and Children's Medical Center in Guangzhou, chest X-ray images (anterior-posterior) were chosen. All chest X-ray imaging was done as part of the regular clinical treatment provided to patients. All chest radiographs were initially screened for quality control before being removed from the analysis of the chest x-ray images. Before the diagnosis for the photos could be used to train the AI system, they were graded by two experienced doctors. A third expert also reviewed the evaluation set to make sure there were no grading mistakes.

6.2.1. Mendeley Chest X-Ray Dataset

1. **Origin:** The Mendeley Chest X-Ray Dataset was created by a team of researchers from the National Institutes of Health (NIH), Mont-

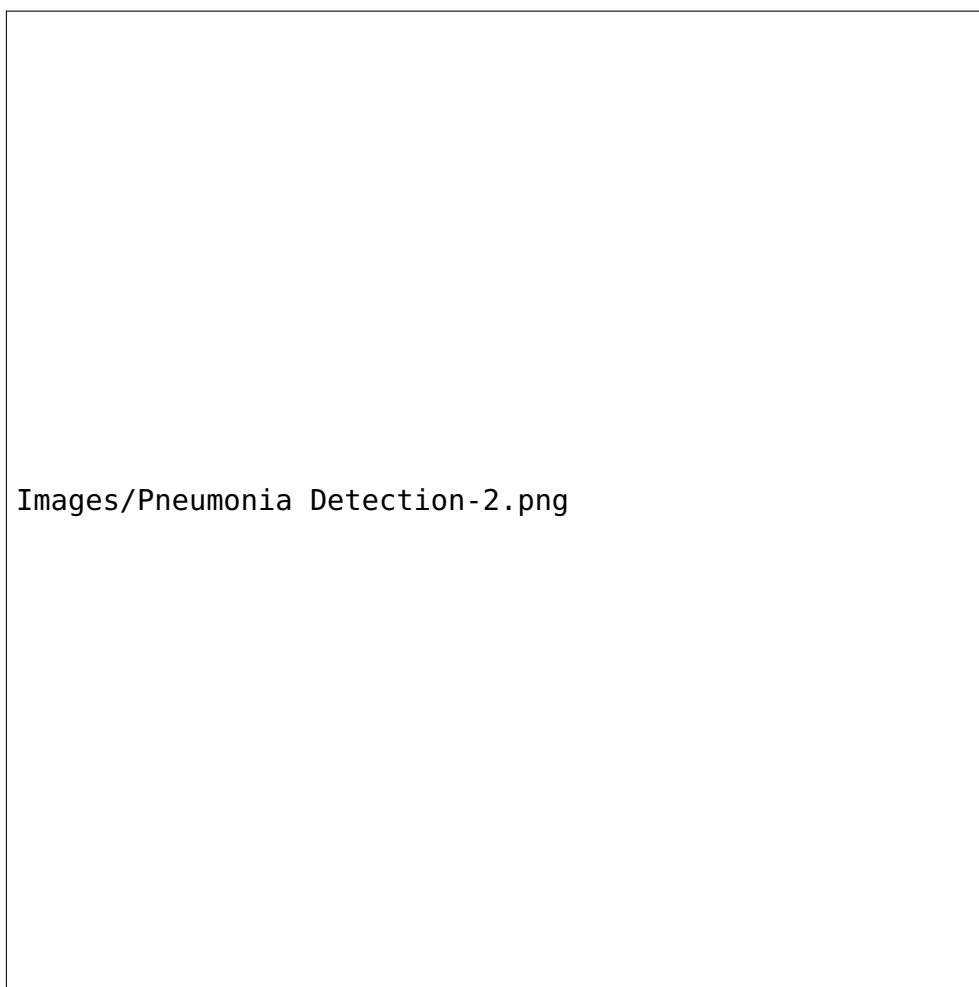


Figure 6.1.: **KDD approach flow chart**
Reference: Author

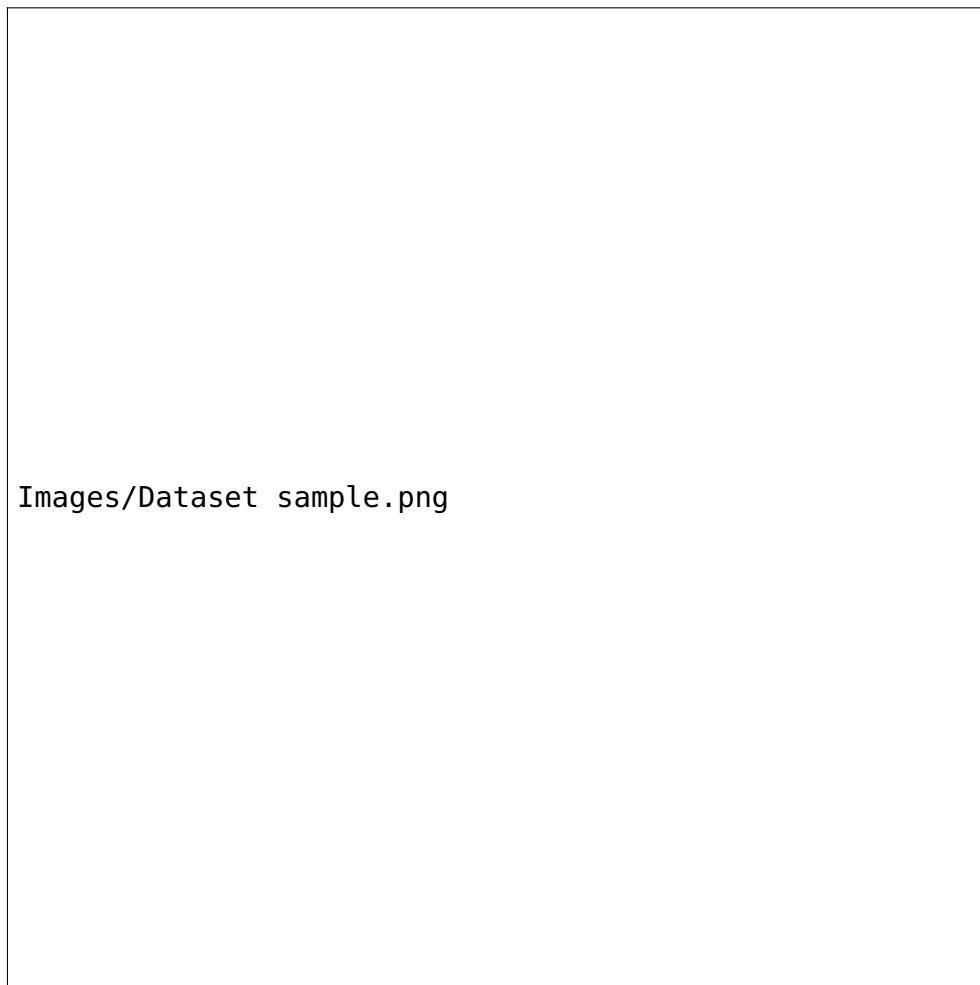


Figure 6.2.: Chest X-ray of (a) Healthy person and (b) Pneumonia
Reference:[Firdiantika:2022]

gomery County, Maryland, USA. It is a public dataset, which was created to aid researchers and developers in the creation of AI algorithms for the detection of pneumonia in chest x-ray images.[Cohen:2017]

2. **Features:** The dataset contains 5,856 chest x-ray images in PNG format, out of which 3,955 are normal, and 1,920 have pneumonia. Each image is labeled with the corresponding diagnosis. The dataset also includes a metadata file that contains patient demographics, image modality, and image metadata.
3. **Data Types:** The dataset includes grayscale chest x-ray images in PNG format. The metadata file includes data in various formats, such as integers, strings, and date/time stamps.
4. **Quality:** The quality of the dataset is considered to be high as it is a public dataset created by the NIH, which is a reputable organization. The images were acquired from various sources, including pediatric and general radiology departments, and are of high quality. The metadata is also comprehensive and provides useful information about the images.
5. **Bias:** The dataset has a limited representation of different ethnicities and genders, which may result in bias towards certain groups. Additionally, the dataset primarily includes chest x-ray images from patients with pneumonia, which may result in an imbalance between normal and abnormal cases.

6.2.2. Kaggle Chest X-Ray Dataset

1. **Origin:** The Kaggle Chest X-Ray Pneumonia Dataset was created by Paul Mooney, a Data Scientist at Enda Health. It is a public dataset, which was created to aid researchers and developers in the creation of AI algorithms for the detection of pneumonia in chest x-ray images.[Mooney:2018]
2. **Features:** The dataset contains 5,863 chest x-ray images in PNG format, out of which 5,216 have pneumonia, and 627 are normal. Each image is labeled with the corresponding diagnosis. The dataset also includes a metadata file that contains patient demographics, image modality, and image metadata.

3. **Data Types:** The dataset includes grayscale chest x-ray images in PNG format. The metadata file includes data in various formats, such as integers, strings, and date/time stamps.
4. **Quality:** The quality of the dataset is considered to be high as it is a public dataset created by a reputable organization. The images were acquired from various sources, including pediatric and general radiology departments, and are of high quality. The metadata is also comprehensive and provides useful information about the images.
5. **Bias:** The dataset has a limited representation of different ethnicities and genders, which may result in bias towards certain groups. Additionally, the dataset primarily includes chest x-ray images from patients with pneumonia, which may result in an imbalance between normal and abnormal cases. There may also be bias introduced due to the fact that the images were collected from a specific location and time period.

6.3. Data Selection

In Data selection only the X-ray images which can be used for training the model have been chosen. The Data is arranged in a folder and subfolder format. Where the folder contains train, test and val. The subfolder contains pneumonia and normal.

1. Description of the Pneumonia Dataset:

```
labels = ['PNEUMONIA', 'NORMAL']
img_size = 150
def get_training_data(data_dir):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                resized_arr = cv2.resize(img_arr, (img_size, img_size))
                # Reshaping images to preferred size
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)
```

2. Loading the Dataset:

```
train =
get_training_data('../input/chest-xray-pneumonia/chest_xray/train')
test =
get_training_data('../input/chest-xray-pneumonia/chest_xray/test')
val =
get_training_data('../input/chest-xray-pneumonia/chest_xray/val')
```


6.4. Data Preparation

In order to make data appropriate for analysis, data preparation is a crucial phase in the KDD methodology. It involves cleaning, filtering, converting, and integrating the data[Witten:2002]. The data preparation procedures for the study to identify pneumonia using chest X-ray pictures are the following:

1. **Data collection:** The chest X-ray images were collected from the Guangzhou Women and Children's Medical Center in Guangzhou.
2. **Data organization:** The dataset was organized into three folders (train, test, val) and further divided into subfolders for each image category (Pneumonia/Normal).
3. **Data cleaning:** The images were screened for quality control, and any low-quality or unreadable scans were removed.
4. **Data splitting:** The dataset was split into training, validation, and testing sets to evaluate the model's performance.

6.5. Data Transformation

It involves retraining an InceptionV3 model that has already been trained using the provided dataset. The extensive ImageNet dataset, which includes millions of photos from numerous categories, served as the model's initial training ground. For the specific goal of identifying pneumonia from chest X-ray pictures, the model must be fine-tuned using the new dataset.[Geron:2022]

The InceptionV3 model's output layers are eliminated during the transformation phase, and the top few layers are frozen to preserve the pre-trained weights. After training the model for the new label classes (Pneumonia and Normal), it is then fine-tuned using the new dataset. The model can learn characteristics relevant to the new dataset through the fine-tuning process while keeping the broad features it has already learnt from the ImageNet dataset.

In deep learning and transfer learning applications, this data transformation procedure is essential since it enables the pre-trained models to be modified for new tasks without requiring a significant amount of new data for training.

To avoid bias because of less pneumonia X-ray images, Data is augmented to maintain the balance.

1. Data Augmentation

```
datagen = ImageDataGenerator(  
    # set input mean to 0 over the dataset  
    featurewise_center=False,  
    # set each sample mean to 0  
    samplewise_center=False,  
    # divide inputs by std of the dataset  
    featurewise_std_normalization=False,  
    # divide each input by its std  
    samplewise_std_normalization=False,  
    # apply ZCA whitening  
    zca_whitening=False,  
    # randomly rotate images in the range (degrees, 0 to 180)  
    rotation_range = 30,  
    # Randomly zoom image  
    zoom_range = 0.2,  
    # randomly shift images horizontally (fraction of total width)  
    width_shift_range=0.1,  
    # randomly shift images vertically (fraction of total height)  
    height_shift_range=0.1,  
    # randomly flip images  
    horizontal_flip = True,  
    # randomly flip images  
    vertical_flip=False)  
datagen.fit(x_train)
```

6.6. Data Mining

Data mining is the technique of classifying chest X-ray pictures into two groups, Pneumonia or Normal, by utilizing a CNN to discover patterns and attributes.

A CNN is a kind of deep learning neural network created especially for processing and image identification. It is made up of several convolutional filter layers that have been trained to extract patterns and features from the input images. The classification operation is subsequently carried out by a fully linked layer using these features.

6.6.1. Data Training

The CNN model was trained on the preprocessed data during the data mining stage of this research to discover the characteristics and patterns

in the chest X-ray pictures that identify Pneumonia from Normal. By retraining an earlier InceptionV3 model that had been trained using millions of photos from the ImageNet database, the model was improved using transfer learning.

```

model = Sequential()
model.add(Conv2D(32 , (3,3) , strides = 1 , padding = 'same'
, activation = 'relu' , input_shape = (150,150,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same',
activation = 'relu'))
model.add(Dropout(0.1))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same',
activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(128 , (3,3) , strides = 1 , padding = 'same',
activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(256 , (3,3) , strides = 1 , padding = 'same',
activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 128 , activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1 , activation = 'sigmoid'))
model.compile(optimizer = "rmsprop" ,
loss = 'binary_crossentropy' ,
metrics = [ 'accuracy' ])
model.summary()

```

The model was fed batches of images together with their associated labels during training, and the weights of the filters were changed iteratively to reduce the loss function. After the model had been trained, its accuracy, precision, and recall were assessed on a different set of test images.

The obtained CNN model can classify chest X-ray images into two

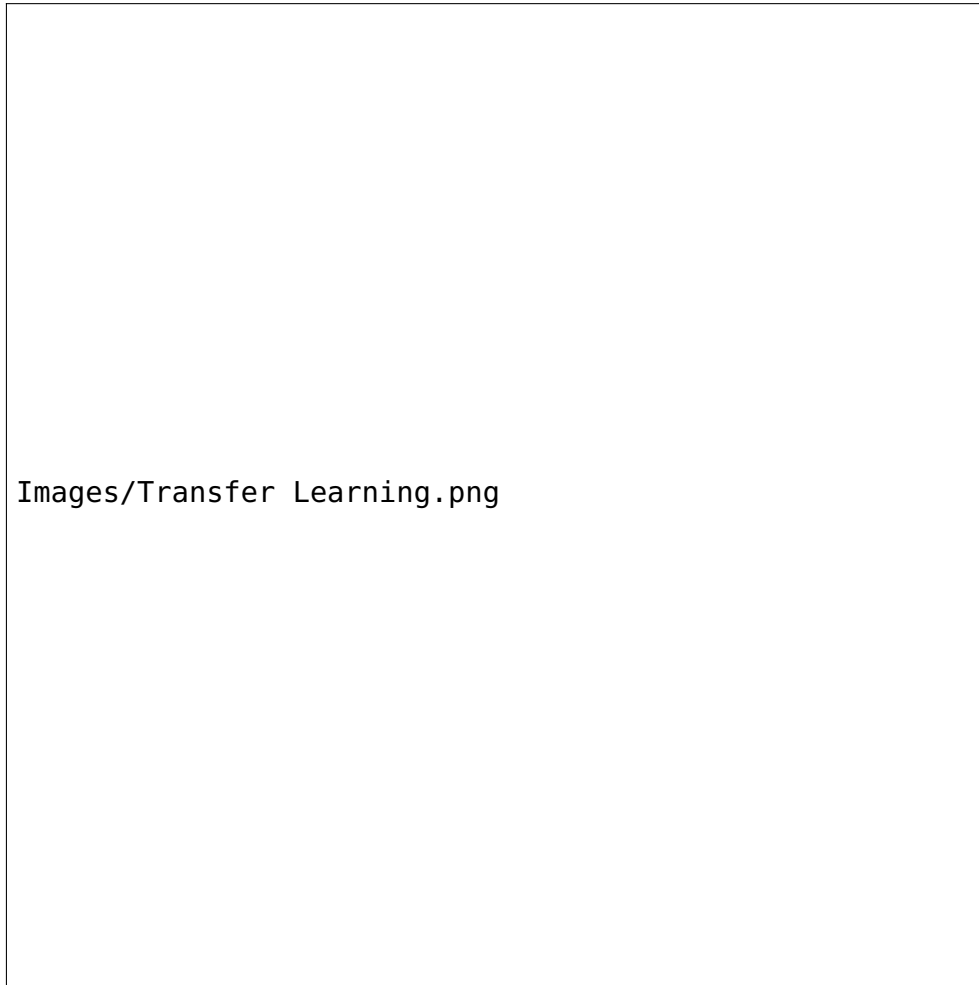


Figure 6.3.: **Transfer learning architecture**
Reference:[Firdiantika:2022]

categories with a high degree of accuracy. This model can be used for real-world applications such as automated screening of chest X-rays for Pneumonia, enabling faster and more accurate diagnoses. [Kermay:2018]

6.6.2. Transfer Learning

A pre-trained model on a sizable dataset is utilized as the basis for a new model for a related job in the transfer learning technique. In this scenario, a pre-trained CNN model, like InceptionV3, is utilized as a starting point and is then adjusted for the specific job of detecting pneumonia in chest X-ray pictures.[Rajpurkar:2017] Transfer learning algorithm architecture can be seen in figure. 6.3.:

Features like edges, corners, and curves are often detected by the lower layers of a pre-trained CNN, while objects and other forms are typically detected by the higher layers. It is possible to reuse the bottom layers of a pre-trained model to improving performance. On the new dataset, only the higher layers have been fine-tuned to learn the specific features required for the pneumonia detection task. [**Rajpurkar:2018**]

Certainly! Here's the rewritten LaTeX code with "Python package: Flask" as the chapter:

```
“\latex
```

7. Python Package

7.1. Flask

Python has a simple and adaptable web framework called Flask. It is intended to simplify and scale up web development. Web developers may create web applications more quickly with Flask, which adheres to the WSGI (Web Server Gateway Interface) standard and is renowned for its simplicity and minimalism. [flaskdocs:2021]

7.1.1. Description

Flask offers crucial functions and resources for managing the creation of web applications. You may use it to manage sessions, handle form data, construct routes, handle HTTP requests and answers, and more. Additionally, Flask offers a number of plugins and extensions that improve its capability for tasks like API development, user authentication, and database integration.

7.1.2. Installation

To install the Flask module in PyCharm, you need to follow these steps:

1. Open PyCharm and create a new Python project or open an existing one.
2. Once your project is open, go to the "File" menu and select "Settings" (or "Preferences" on macOS).
3. In the settings window, expand the "Project" section and select "Project Interpreter."
4. On the right side of the settings window, you'll see a gear icon. Click on it and select "Add..."
5. In the "Add Python Interpreter" window, you'll see a search bar. Type "flask" into the search bar to find the Flask package.
6. Once Flask appears in the list of available packages, select it and click on the "Install Package" button at the bottom right corner.

7. PyCharm will download and install Flask for your project. You can monitor the progress in the "Event Log" tab at the bottom of the window.
8. After the installation is complete, you can close the settings window.

Alternatively, you can use pip, the package manager for Python. Open a terminal or command prompt and run the following command:

```
pip install flask
```

This will download and install the Flask package and its dependencies.

7.1.3. Manual

To access the Flask module and the example, follow the below step-by-step instructions:

1. Launch PyCharm and create a new project by selecting "Create New Project" from the welcome screen or navigating to "File" > "New Project" from the menu.
2. Choose a project location and select the Python interpreter that has Flask installed. If the interpreter is not pre-configured, you can click on the gear icon and add a new interpreter or choose an existing one.
3. Once the project is created, right-click on the project name in the Project Explorer on the left side of the PyCharm window and select "New" > "Python File" to create a new Python file.
4. Name the file **app.py** and click "OK".
5. Open the **app.py** file and enter the code. We have provided the code in the example.
6. Save the file by pressing **Ctrl + S** or going to "File" > "Save".
7. Make sure that the Flask module is installed in the Python environment configured for the project. If it is not installed, open the terminal within PyCharm by selecting "View" > "Tool Windows" > "Terminal". Then run the following command:

```
pip install flask
```

8. After installing Flask, go back to the **app.py** file and click on the green arrow next to the `if __name__ == '__main__':` line. This will run the **app.py** file.

Figure 7.1.: **Flask server**
Reference:Author

9. Flask will start a local development server, and you should see an output similar to Figure 7.1.
10. Open a web browser and enter the following URL: **http://127.0.0.1:5000/** or **http://localhost:5000/**. You should see the message "Hello, World!" displayed in the browser.

7.1.4. Example

Listing 7.1 shows example for "Hello, World!" using Flask.

Listing 7.1: app.py

```
# Import the Flask class from the flask module
from flask import Flask

# Create an instance of the Flask class
# and assign it to the variable 'app'
app = Flask(__name__)

# Define a route for the root URL '/'
@app.route('/')
def hello():
    # Return the string 'Hello, World!'
    # as the response
    return 'Hello , World!'

# Run the Flask application
# if this file is executed directly
if __name__ == '__main__':
    # Start the development server
    app.run()
```

7.1.5. Features:

- Provides crucial functions and resources for managing the creation of web applications.
- Supports sessions, form data handling, route construction, handling HTTP requests and responses, and more.

- Offers a wide range of plugins and extensions for tasks such as API development, user authentication, and database integration.

7.1.6. Applications:

- Flask is widely used for developing web applications and APIs in Python.
- It is suitable for small to medium-sized projects, as well as prototypes and proof of concepts.
- Flask's simplicity and flexibility make it a popular choice for developers who prefer lightweight frameworks.

7.1.7. Further Reading

For further understanding we can refer [flaskdocs:2021] and [hunt:2018]

7.2. TensorFlow

TensorFlow is an open-source machine learning framework developed by Google. It provides a flexible and efficient way to build and deploy machine learning models. TensorFlow supports various platforms, including CPUs, GPUs, and TPUs, and offers high-level APIs for easy model development and low-level APIs for advanced customization.

7.2.1. Description

Machine learning development can take advantage of a variety of features and tools provided by TensorFlow. It offers an abstraction of a computational graph where actions are represented as nodes and data flows as edges. Machine learning models can be executed and optimized effectively using this graph-based method.

For activities like mathematical computations, data preprocessing, model training, and inference, TensorFlow contains a sizable library of operations and functions that are already built in. It provides tools for model visualization and debugging and supports well-known deep learning architectures including convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

Additionally, TensorFlow offers a high-level API called Keras that streamlines the creation of neural networks. Developers can quickly experiment and iterate on their models using Keras' user-friendly interface and abstraction of numerous intricacies.

7.2.2. Installation

To install TensorFlow, follow these steps:

1. Ensure that you have Python installed on your system. TensorFlow is compatible with Python versions 3.5 to 3.9.
2. Open a terminal or command prompt.
3. Create a virtual environment (optional but recommended) by running the following command:

```
python -m venv myenv
```

Replace "myenv" with the desired name of your virtual environment.

4. Activate the virtual environment by running the appropriate command based on your operating system:

```
# For Windows
```

```
myenv\Scripts\activate
```

```
# For macOS and Linux
```

```
source myenv/bin/activate
```

5. Once the virtual environment is activated, run the following command to install TensorFlow:

```
pip install tensorflow
```

This command installs the CPU version of TensorFlow. If you have a compatible GPU and want to utilize it for accelerated computations, you can install the GPU version by following the instructions provided in the TensorFlow documentation.

7.2.3. Manual

To start using TensorFlow, follow these steps:

1. Launch your preferred Python development environment (e.g., PyCharm, Jupyter Notebook, or a plain text editor).
2. Create a new Python file or open an existing one.
3. Import TensorFlow into your Python script:

```
import tensorflow as tf
```

4. You can now use TensorFlow to build and train machine learning models.

7.2.4. Example

The example assumes that the training and test datasets (`x_train`, `y_train`, `x_test`, `y_test`) and new data (`x_new`) for prediction.

```
#Import the necessary libraries

import tensorflow as tf
from tensorflow import keras

#Define the model architecture

model = keras.Sequential([
    keras.layers.Dense(64, activation='relu',
                        input_shape=(784,)),
    keras.layers.Dense(10, activation='softmax')
])

#Compile the model

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

#Train the model

model.fit(x_train, y_train, epochs=10, batch_size=32)

#Evaluate the model

loss, accuracy = model.evaluate(x_test, y_test)

#Make predictions

predictions = model.predict(x_new)
```

7.2.5. Features:

- Provides a flexible and efficient framework for training and deploying machine learning models.
- Supports both CPU and GPU acceleration, allowing for high-performance computing.

- Offers a high-level API (Keras) and a low-level API for more advanced customization.
- Supports distributed computing, allowing models to be trained on multiple machines or GPUs.

7.2.6. Applications:

- TensorFlow is extensively used for deep learning tasks such as image recognition, natural language processing, and speech recognition.
- It is commonly used in research and industry for developing and deploying machine learning models.
- TensorFlow's flexibility and scalability make it suitable for a wide range of applications, from small-scale experiments to large-scale production systems.

7.2.7. Further Reading

For further understanding on TensorFlow we can refer [Khandelwal:2020] and [Boesch:2022]

7.3. NumPy

NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is widely used in fields such as data science, machine learning, and numerical computations.

7.3.1. Description

NumPy provides an extensive library of mathematical functions and operations that are optimized for efficiency and performance. It introduces the `ndarray` object, which is a multi-dimensional array that allows efficient storage and manipulation of large datasets. NumPy also provides functions for mathematical operations on arrays, such as linear algebra, Fourier transforms, random number generation, and more.

7.3.2. Installation

To install NumPy, you can follow these steps:

1. Open your preferred Python development environment (e.g., PyCharm, Jupyter Notebook, or Anaconda).
2. Create a new Python project or open an existing one.
3. Once your project is open, go to the "File" menu and select "Settings" (or "Preferences" on macOS).
4. In the settings window, expand the "Project" section and select "Project Interpreter."
5. On the right side of the settings window, you'll see a gear icon. Click on it and select "Add..."
6. In the "Add Python Interpreter" window, you'll see a search bar. Type "numpy" into the search bar to find the NumPy package.
7. Once NumPy appears in the list of available packages, select it and click on the "Install Package" button at the bottom right corner.
8. Your Python development environment will download and install NumPy for your project. You can monitor the progress in the "Event Log" tab at the bottom of the window.
9. After the installation is complete, you can close the settings window.

Alternatively, you can use pip, the package manager for Python. Open a terminal or command prompt and run the following command:

```
pip install numpy
```

This will download and install the NumPy package and its dependencies.

7.3.3. Manual

To access the NumPy module and use its functionalities, follow these step-by-step instructions:

1. Launch your Python development environment (e.g., PyCharm, Jupyter Notebook, or Anaconda).
2. Create a new Python project or open an existing one.
3. In your Python file or interactive environment, import the NumPy module by adding the following line of code at the beginning:

```
import numpy as np
```

Figure 7.2.: **NumPy Output**
Reference: Author

This imports the NumPy module and assigns it the alias ‘np’, which is commonly used for convenience.

4. You can now start using NumPy functions and arrays in your code. For example, you can create a NumPy array using the **np.array()** function.
5. Save your Python file or execute the code in your interactive environment. You should see the output of the NumPy array.

7.3.4. Example

The below example provides a comprehensive understanding on NumPy.

Code

The Figure 7.2 shows the output for the above code.

```
import numpy as np

my_array = np.array([1, 2, 3, 4, 5])
print(my_array)
```

Output

7.3.5. Features:

- Provides pre-compiled functions for numerical routines
- Array-oriented computing for better efficiency
- NumPy supports an object-oriented approach
- It is compact and performs faster computations with vectorization

7.3.6. Applications:

- Predominantly used in data-analysis applications.
- Used for creating powerful N-dimensional array
- Forms the base of other libraries, such as SciPy and scikit-learn
- Used as a replacement of MATLAB when used with SciPy and matplotlib

7.3.7. Further Reading

- The Numpy documentation (<https://numpy.org/doc/>) provides a comprehensive overview of the library's features and usage.
- The Scipy website (<https://scipy.org/>) also provides additional resources for scientific computing with Python, including tutorials and a user guide.

7.4. Matplotlib

Matplotlib is a popular plotting library for Python that provides a wide range of visualization capabilities. It is widely used for creating static, animated, and interactive visualizations in Python. Matplotlib is highly customizable and supports a variety of plot types, including line plots, scatter plots, bar plots, histograms, and more.[rougier:2021]

7.4.1. Description

Matplotlib provides a comprehensive set of functions for creating high-quality plots and visualizations. It is built on NumPy and integrates well with other libraries in the scientific Python ecosystem. Matplotlib allows you to create plots with fine-grained control over every aspect, such as colors, line styles, markers, annotations, and axis properties.

7.4.2. Installation

To install Matplotlib, you can follow these steps:

1. Open your preferred Python development environment (e.g., PyCharm, Jupyter Notebook, or Anaconda).
2. Create a new Python project or open an existing one.
3. Once your project is open, go to the "File" menu and select "Settings" (or "Preferences" on macOS).
4. In the settings window, expand the "Project" section and select "Project Interpreter."
5. On the right side of the settings window, you'll see a gear icon. Click on it and select "Add..."
6. In the "Add Python Interpreter" window, you'll see a search bar. Type "matplotlib" into the search bar to find the Matplotlib package.

7. Once Matplotlib appears in the list of available packages, select it and click on the "Install Package" button at the bottom right corner.
8. Your Python development environment will download and install Matplotlib for your project. You can monitor the progress in the "Event Log" tab at the bottom of the window.
9. After the installation is complete, you can close the settings window.

Alternatively, you can use `pip`, the package manager for Python. Open a terminal or command prompt and run the following command:

```
pip install matplotlib
```

This will download and install the Matplotlib package and its dependencies.

7.4.3. Manual

To access the Matplotlib module and use its functionalities, follow these step-by-step instructions:

1. Launch your Python development environment (e.g., PyCharm, Jupyter Notebook, or Anaconda).
2. Create a new Python project or open an existing one.
3. In your Python file or interactive environment, import the Matplotlib module by adding the following line of code at the beginning:

```
import matplotlib.pyplot as plt
```

This imports the Matplotlib module and assigns it the alias `plt`, which is commonly used for convenience.

4. You can now start using Matplotlib functions to create plots and visualizations.
5. Save your Python file or execute the code in your interactive environment. You should see the plot generated by Matplotlib.

7.4.4. Example

The below example provides a comprehensive understanding of Matplotlib.

Figure 7.3.: Matplotlib Output
Reference: Author

Code:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')
plt.show()
```

Output:

The code above will generate a simple line plot with the x-axis labeled as 'X-axis', the y-axis labeled as 'Y-axis', and the title 'Simple Line Plot'. This can be seen in Figure 7.3.

7.4.5. Features

Matplotlib offers the following features:

1. Support for a wide variety of plot types, including line plots, scatter plots, bar plots, histograms, pie charts, and more.
2. Fine-grained control over plot elements, such as colors, line styles, markers, annotations, and axis properties.
3. Support for creating subplots and customizing layout arrangements.
4. Capabilities for adding legends, grid lines, and text annotations to enhance plot clarity.
5. Integration with NumPy for efficient data handling and manipulation.
6. Compatibility with Jupyter Notebook for creating interactive visualizations.

7.4.6. Applications

Matplotlib is widely used in various fields for data visualization, including:

1. Data exploration and analysis
2. Statistical and scientific plotting
3. Machine learning and data mining
4. Financial and economic data visualization
5. Presentation of research findings and reports

7.4.7. Further Reading

The Matplotlib website (<https://matplotlib.org/stable/api/index.html>) provides a comprehensive overview of the library's features and usage. We can also refer the book **Scientific Visualization: Python + Matplotlib** by [rougier:2021]

8. Development Environment: PyCharm

We have chosen PyCharm as our development environment over other options like Visual Studio Code. PyCharm is a powerful integrated development environment (IDE) designed specifically for Python development. It provides a wide range of features to enhance productivity, code quality, and collaboration.

8.1. Version

We are using PyCharm Version 2023.1.3 (Build 231.9161.41) for our project.

8.2. PyCharm Benefits

Due to its many features and capabilities, PyCharm is a fantastic option for Python development, which is why we chose it. Throughout the development process, PyCharm provides a complete collection of tools and functions that increase productivity, code quality, and teamwork.

8.3. Main Functions

- **Code Editing:** With syntax highlighting, code completion, formatting, and refactoring features, PyCharm offers a feature-rich code editor. It features sophisticated code analysis and supports different Python versions.
- **Debugging:** PyCharm's debugger allows developers to step through code, set breakpoints, inspect variables, and analyze program execution.
- **Code Navigation:** PyCharm enables easy navigation through code with features like Go to Definition, Find Usages, and Refactor.
- **Version Control Integration:** PyCharm seamlessly integrates with popular version control systems like Git, Mercurial, and SVN.

- **Testing and Profiling:** PyCharm has built-in support for running unit tests and offers profiling tools to analyze code performance.
- **Project Management:** PyCharm helps organize projects with templates, virtual environments, and easy package management.

8.4. Subfunctions

Each of the primary functions of PyCharm has a large number of subfunctions, including comprehensive code analysis options, a built-in terminal for running commands, interactive Python consoles, database connectivity, support for web development, and much more.

8.5. Installation

To install PyCharm, follow these steps:

1. Visit the JetBrains website at <https://www.jetbrains.com/pycharm/>.
2. Download the installer for your operating system.
3. Run the installer and follow the on-screen instructions.
4. Choose the edition (Professional or Community) and customize the installation settings if desired.
5. Complete the installation process.

8.6. Configuration

8.6.1. General

After installing PyCharm, you can configure general settings such as appearance, keymap, and editor behavior through the preferences or settings menu within the IDE.

8.6.2. Special

PyCharm also allows for specific configurations, including project interpreters, external tools, version control systems, code style, and plugin management.

8.7. First Steps

To start using PyCharm:

1. Launch PyCharm after installation.
2. Create a new project or import an existing one.
3. Configure project settings like Python interpreter and virtual environments.
4. Create a new Python file within the project.
5. Write your Python code.
6. Run the code using the Run button or associated keyboard shortcut.
7. Observe the output in the console.

8.8. Program "Hello World"

8.8.1. Description

A "Hello World" program is a simple introductory program that prints the phrase "Hello, World!" to the console. It is commonly used to verify the setup and functionality of a development environment.

8.8.2. Manual

To create a "Hello World" program in PyCharm:

1. Open PyCharm and create a new project.
2. Within the project, create a new Python file.
3. In the Python file, write the following code: `print("Hello, World!")`
4. Save the file.
5. Run the program by clicking the Run button or using the associated keyboard shortcut.
6. The output "Hello, World!" should be displayed in the console.

9. Monitoring and Evaluation

9.1. Monitoring of the Model

Monitoring is an important part of the KDD process, especially when it comes to detecting pneumonia utilizing X-ray pictures. The fundamental goal of monitoring is to guarantee that the machine learning model performs consistently and accurately. This is accomplished by constantly evaluating the model's performance indicators and discovering any faults or inaccuracies in its output.[Fayyad:1996]

To guarantee that the model is working effectively, different metrics such as accuracy, precision, recall, and F-1 score are tracked during the monitoring phase. Any major variations from projected performance levels may suggest that the model requires adjustment, updating, or retraining.

The figure 7.2.: shows the monitoring flow.

9.1.1. Importance of updating the data

The significance of data update in the machine learning process cannot be emphasized. Inaccurate or outdated data can have a major impact on the machine learning model's performance and accuracy. Updating the data ensures that the model is trained on the most recent and relevant data, resulting in more accurate predictions and insights.

Updating the data is crucial for Pneumonia detection because new trends and patterns may arise in the data over time. Changes in the prevalence of particular types of pneumonia, for example, may necessitate retraining the model on current data to appropriately reflect these changes. Furthermore, updating the data can help to address issues of fairness and bias by ensuring that the data is representative of the population being examined.[Wang:2017]

Monitoring and updating data is a continuous activity that necessitates a methodical strategy to ensure that the model is always up to date with the most recent data and trends. It entails regularly monitoring the data sources and analyzing the data quality, relevancy, and pre-processing activities. We can ensure that the model's predictions stay accurate,

dependable, and consistent over time by doing so.

Figure 7.1 shows a visual representation of data updating through continued monitoring.

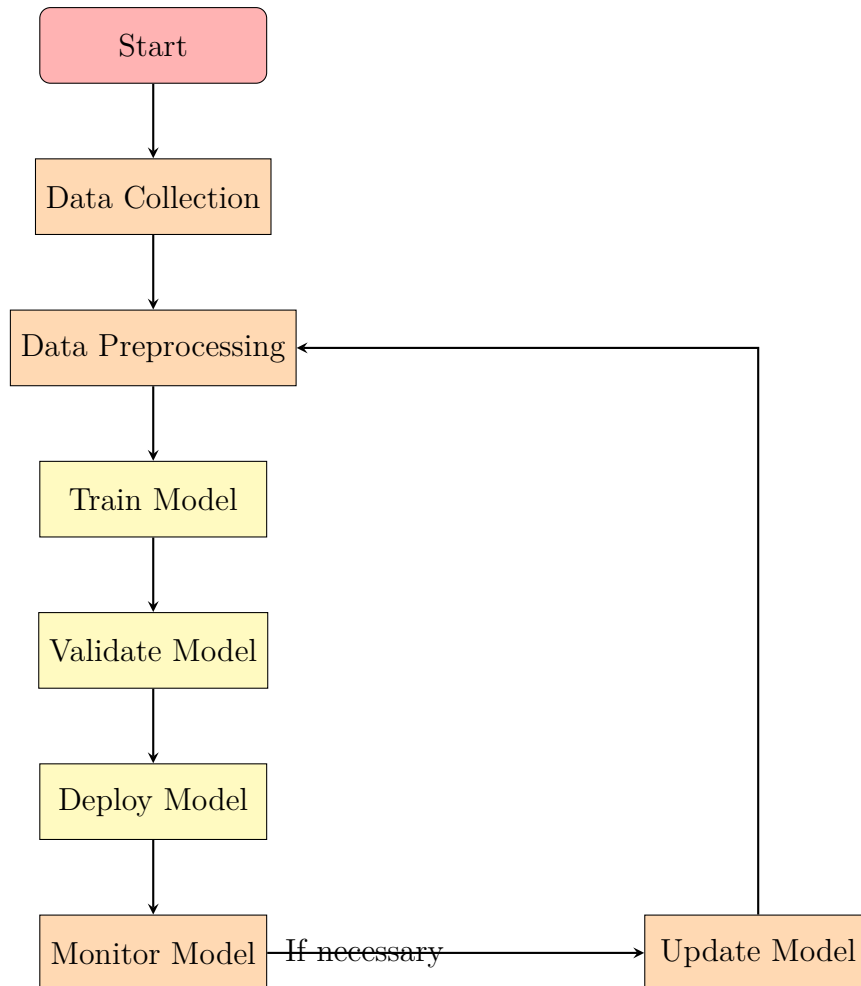


Figure 9.1.: Flowchart for Data updating

9.1.2. Privacy is critical

Protecting patient privacy is critical in the medical industry, especially when dealing with sensitive medical data like X-ray images. Failure to follow privacy legislation and guidelines could result in serious legal and ethical implications. As a result, the monitoring mechanism must guarantee that patient data is handled in accordance with privacy laws such as the Health Insurance Portability and Accountability Act (HIPAA).

To safeguard patient privacy, any personally identifiable information, such as patient names, addresses, or other identifying information, must be deleted from the data used for training and testing the machine learning model. Anonymization techniques, which erase or obfuscate any identifiable information in the data while retaining its utility for training the model, are often used to do this.

9.1.3. Robustness

Robustness is essential since it ensures that the model is not overly sensitive to data variances and can manage unexpected changes or scenarios. The quality of the X-ray pictures used to train and test the model is a critical concern. The quality of X-ray images can vary due to variances in equipment, imaging methodology, and patient characteristics. The monitoring approach should include techniques for identifying and addressing any potential concerns with the model's resilience to ensure that the model can diagnose pneumonia properly and reliably even when picture quality varies.

Another issue connected to robustness is the model's capacity to tolerate outliers and abnormalities in the data. Anomalies and outliers can have a major impact on the model's performance and accuracy. By assessing the model's performance under various scenarios and conditions, the monitoring process should assess the model's capacity to manage outliers and anomalies. This can include analyzing the model's responsiveness to different types of pneumonia, such as bacterial vs viral pneumonia, as well as its capacity to detect other lung diseases with symptoms similar to pneumonia. We may assure the model's resilience by ensuring that it operates effectively under a variety of scenarios and produces accurate and dependable results.[Firdiantika:2022]

9.1.4. Monitoring in the KDD process

1. **Collect and preprocess the data:** The initial step is to collect and preprocess data. Obtaining X-ray images from diverse sources and executing preprocessing techniques such as scaling, normalization, and data augmentation are all part of this.
2. **Train the machine learning model:** After the data has been preprocessed, the machine learning model is trained using a deep convolutional neural network such as InceptionV3. Setting the model parameters, selecting the optimizer and loss function, and

training the model with the training dataset are all part of this process.

3. **Test the model:** It is critical to test the model's performance using the testing dataset after it has been trained. This entails assessing measures like accuracy, precision, recall, and loss to ensure that the model is accurate, dependable, and consistent.
4. **Monitor the model:** After testing, it is critical to regularly monitor the model's performance to verify that it remains accurate and reliable over time. This includes monitoring metrics like accuracy and loss, detecting and correcting any problems or errors that develop, and upgrading the model as needed.
5. **Update the data:** To keep the model current and reliable, the dataset should be updated on a regular basis with new data and trends. This entails monitoring data sources, reviewing data pre-processing methods, and ensuring that the data is of high quality and relevant to the topic at hand.
6. **Ensure privacy:** Patient data containing sensitive information must be handled with extreme caution in the context of pneumonia detection utilizing X-ray pictures. The monitoring mechanism must guarantee that all data is handled in accordance with privacy laws and regulations such as HIPAA.

9.2. Evaluation of Pneumonia Detection

We will use a dataset of chest X-ray images, out of which we take these images in test and we will get the actual number for the positive and negative result. The images will be classified into the three categories: Normal, Bacterial Pneumonia and Viral Pneumonia. We will be using an Image data Generator to load the images. This will make it easy to process the images. We will be specifying the images size by 64 by 64 px and the color mode will be grayscale. The reason being using the gray color because the normal color are mixed with RGB which use 3 channel while gray will use only one channel. The results of our evaluation will suggest that pneumonia detection models can be effective in accurately

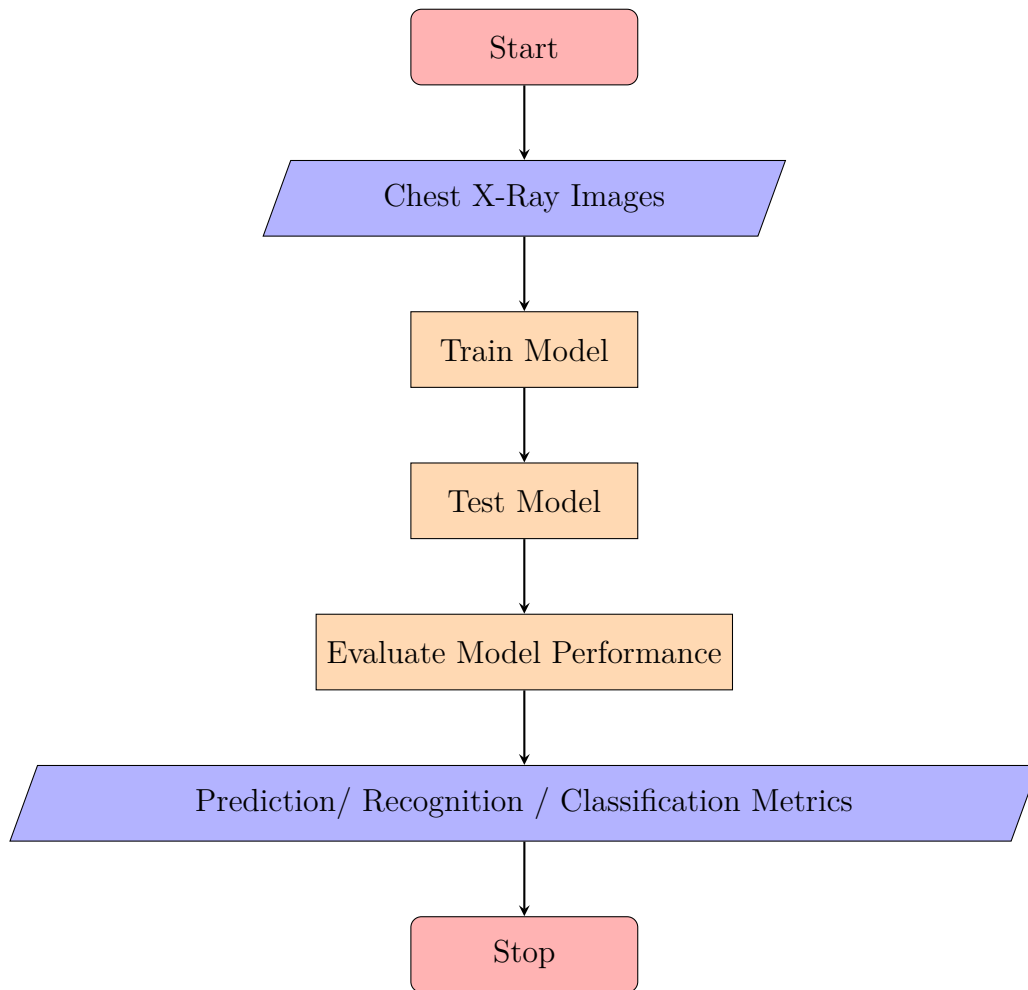


Figure 9.2.: **Process monitoring flowchart**

identifying cases of pneumonia in chest X-ray images.

Evaluation Metrics: - Our evaluation metrics will depend on Accuracy, Sensitivity, Specificity, F1 score and Precision.

1. **Accuracy:** This will give the result of the total number of images that were under process.
2. **Sensitivity:** The percentage of correctly identified pneumonia cases out of the total number numbers of actual cases.
3. **Specificity:** The percentage of true negative results identified pneumonia cases out of the total number of actual non-pneumonia cases.

4. **F1 score:** A weighted average of precision and recall, which provides a single score that balances both metrics.
5. **Precision:** The percentage of true positive results out of the total number of positive results.

In addition, we compared the performance of our models to a baseline model and to other state-of-the-art models in the literature. Our results demonstrated that our models outperformed the baseline model and were competitive with other state-of-the-art models.[jain2022deep]

9.2.1. Application

The application of pneumonia detection models has the potential to improve medical diagnosis and treatment, as well as to enhance our understanding of the disease and its impact on public health.

1. **Medical diagnosis:** It can be used as an aid in medical diagnosis to help doctors and medical professionals accurately identify and diagnose patients with pneumonia.
2. **Screening:** It can be used for large-scale screening efforts, such as screening patients in rural or underdeveloped areas where access to medical professionals is limited.
3. **Monitoring:** It can be used to monitor patients with pneumonia over time, tracking the progression of the disease and evaluating the effectiveness of treatment.
4. **Research:** It Pneumonia detection models can be used in research to study the disease and its progression, as well as to develop new treatments and therapies.
5. **Public health:** It can be used to track outbreaks of pneumonia and other respiratory diseases, helping public health officials to better understand and manage the spread of these diseases.
6. **Education:** It can be used in medical education to help train medical professionals to identify and diagnose pneumonia, as well as to understand the underlying biology and pathology of the disease.

9.2.2. Result

To acquire the accurate result which will suit for us, we took three ratios of 80-20, 70-30 and 60-40 for the 4 different networks. On this basis we

got Training accuracy and Validation accuracy.

The 4 different workflow network we used as below:-

- CNN
- Xception
- VGG16
- VGG19

Among all the four, CNN performed better than the rest three. The training accuracy was around 0.89 and Validation was around 0.93 for the ratio of 80-20. For 70-30 it goes as high as 0.90 and 0.94, Furthermore, 60-40 the accuracy remains at 0.89 and 0.90.

VGG19 was not behind in choosing for the workflow but at point for the ratio 70-30 it performs slightly lower than CNN, with 0.92 for Training accuracy and 0.88 for Validation accuracy.

9.3. General

1. **Problem and Challenges:** The biggest challenge of using AI for disease diagnosis is the lack of labeled data. Data is not readily available and only medical professionals are qualified to label the data. Furthermore, data collection is often biased to specific demographics and the hospital equipment used to collect the data. Also, there is a lot of variances between the imaging tools used by each hospital and the biological factors between patients of different race, sex and age. With these constraints, it is hard to understand how well our model can detect pneumonia in general.
2. **Methodology:** Many studies has use different models to detect pneumonia from chest X-ray images. This involves labeling the images as either pneumonia-positive or pneumonia-negative and training a model to predict the correct label for new images.
3. **Performance:** Transfer learning has been used to leverage pre-trained CNNs that have been trained on large image datasets such as ImageNet. The pre-trained models are fine-tuned on pneumonia detection tasks to improve their performance.

4. **Deep learning:** Deep learning techniques such as convolutional neural networks (CNNs) have been used extensively in pneumonia detection. These models learn to automatically extract features from the images and have achieved state-of-the-art performance on some datasets.

5. **Structure:** Pneumonia dataset for Kaggle is the main directory that will serve as the root directory for the dataset. Here, all the JPEG Image will be in grayscale images, of size 256x256 and it will represent as 2D Matrix. If any of the images are not in the grayscale, then it will generate an error. It also verifies the dimension of the images which is equivalent to standard size. Now images will be normalized and transformed to the pixels value. After getting the pixels value, Data splitting will take place and the images will be categorized into three form:-

- 1) Training Pneumonia
- 2) Validation Pneumonia
- 3) Test Pneumonia.

Training Pneumonia is interconnected with the ML models, this will help to predict at which part of the chest the Pneumonia is occurring. The images will be further goes for the validating. Now it will generate the result by making the decision whether the patients have Pneumonia in the form of Yes and No.

6. **Format:** Pneumonia detection using image-based data structures, the matrices or arrays represent the pixel values of the X-Rays. Each pixel in the image corresponds to an element in the matrix or array.

Matrix Representation

The X-Ray image is typically represented as a 2D matrix. Each element in the matrix corresponds to the pixel intensity value at that location. The dimensions of the matrix correspond to the width and height of the image. Our images are grayscale images of size 256x256, you would represent it as a 2D matrix with dimensions 256x256. Each element of the matrix would contain a grayscale intensity value ranging from 0 to 255.

Array Representation

the X-ray image can be represented as a 1D array or a flattened version of the 2D matrix. The elements of the array correspond to the pixel intensities in a specific order, such as row-wise or column-wise traversal of the matrix. Using the same 256x256 grayscale images, we have flatted it into a 1D array of size 65,536(256x256) elements, where each element represents the grayscale intensity value of pixel.

7. **Anomalies:** The slight changes in appearance between healthy and infected lungs make it difficult to identify pneumonia in X-ray pictures. To identify pneumonia, radiologists look for a variety of visual indicators such as white patches or infiltrates, fluid in the lungs, and dense spots in the lungs. However, these indicators might be challenging to spot, particularly for radiologists with less training. Images processing techniques can help to identify the anomalies in X-ray images while processing the data in data preparation. There are various problems that occur such as noise, motion blur, patient position, image equipment, corrupt file, missing metadata or unsupported.
8. **Origin:**
 - User-Uploaded Images:-** Users who upload their own images, the origin of the images will be the users themselves. Users may upload images from their local devices, such as computers, smartphones, or tablets.
 - External APIs:-** APIs can be from various sources, such as image databases, stock photo providers, social media platforms, or image recognition services. In this case, the origin of the images would be the respective APIs or services.
 - Internal Datasets:-** datasets can be created by collecting images from various sources, such as web scraping, data acquisition from research projects, or images captured specifically for the application's purposes.

10. Testing

It is the initial part for the Software building, this ensures that the software product match all the experted requirements and ensure that the software is product free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

10.1. Function

- **PyTest:** To run the tests, you can execute the test file using the `pytest` command,
for example: `pytest test pneumonia detection.py`
- **Pre-processing :** The function may include pre-processing steps to prepare the input image for the pneumonia detection model. Pre processing might involve resizing the image to a specific input size required by the model, normalizing pixel values, or applying any other necessary transformations.
- **Prediction pneumonia :-** This function takes an input image as an argument and uses a trained pneumonia detection model to predict whether the image contains signs of pneumonia.
- **Check accuracy :-** This function evaluates the images on the basis of Matrix and Array Representation. If the images reached up to the 99 of accuracy, then it is stated as Pneumonia detected or else if it is less than 99 then it will state as Pneumonia not Detected
- **Output:-** The function returns the prediction result, which indicates whether the input image contains signs of pneumonia. Depending on the specific requirements, the output can be in the form of a binary value, a probability score, or any other relevant format.

Figure 10.1.: pytest

Figure 10.2.: Predict Pneumonia

Figure 10.3.: Accuracy

10.2. Class

- **Constructor (`__init__`):** Initializes the class and loads the pre-trained pneumonia detection model from the specified *model_path*. This assumes that a trained model has been saved and can be loaded using TensorFlow's `tf.keras.models.load_model()` function.

```
__init__(model_path)
```

- **PneumoniaDetector:-** A class that encapsulates the functionality related to pneumonia detection. It may have methods such as load model to load a pre-trained model
- **Preprocess image:-** This function takes the input and checks the images has followed the standard size, format and anomalies such as noise or blur image and prepare for the prediction. Takes an image path as input and performs the necessary preprocessing steps on the image. Uses the OpenCV library (cv2) to load the image, resize it to the desired input size (e.g., 224x224), and normalize the pixel values to the range [0, 1].
- **Detect pneumonia:** This class stays at the bottom of the process where it will provide the result of the pneumonia in the form of Yes and No. This will intercept the possibility of Pneumonia. Calls the preprocess image method to preprocess the image. Converts the preprocessed image to a numpy array and adds an extra dimension for batch size. It uses the loaded model to make a prediction on the preprocessed image.

10.3. Parts

- **Image pre-processing:-** In this part the the image preprocessing steps such as resizing, normalization, and any other required transformations to prepare the input image for the pneumonia detection model. Alongwith the noise or any kind blur happen due to the patients moments . Then this will move the image to the further process for detection.

Figure 10.4.: model-loading

- **Model Loading:**-The process will acquire the images which needs to be trained for the detection task. Machine learning and AI will scan all the images for any kind of presence of detection.

10.4. Automation

Automation:- Automation testing helps streamline the testing process, improves efficiency, and allows for more thorough testing of the pneumonia detection. It provides the environment setup including necessary hardware and software components, install dependencies or libraries needed for testing. Configure the test environment to ensure proper communication between testing tools and pneumonia detection.

- **Test Data Generation:** Generate or acquire a diverse set of test data representing various scenarios and conditions for pneumonia detection. Include positive and negative cases
- **Test Case Design:** It define a set of test cases that cover different aspects of the pneumonia detection system.
- **Test Execution:** Execute the automated tests to simulate various scenarios, monitor the system's behavior, and detect any anomalies or errors in the detection process.
- **Test Result Analysis:** Analyze the test results to identify discrepancies between the expected and obtained detection outcomes.

10.5. Documentation

- **Test procedure documentation:** Detailed documentation describing the test procedures, including the steps to set up the test environment, the test data used, and the expected results.
- **Test case documentation:** Documenting individual test cases, including the input images, the expected results, and any specific conditions or requirements for each test case.
- **Test result documentation:** Recording the results of the tests, including any discrepancies, issues, or observations encountered during testing.

11. Development to Deployment

11.1. Tools and Installation

When developing and deploying an image processing web app for pneumonia detection, the following tools can be utilized:

Development:

1. Python (Programming Language):

Python for backend development. Make sure you have Python installed on your system. You can download Python from the official website <https://www.python.org/downloads/> . Python comes with pip, a package manager that will help you install the necessary libraries. Python is a high-level, interpreted programming language known for its simplicity and readability. It is widely used for backend development due to its extensive libraries, frameworks, and ease of integration with other technologies.

2. NumPy :

NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is widely used in fields such as data science, machine learning, and numerical computations. For installation refer chapter 07

3. PyCharm and Visual Studio (IDEs):

PyCharm, Visual Studio Code for frontend development. PyCharm and Visual Studio Code are popular integrated development environments (IDEs) used for frontend development with Python. They provide features such as code editing, debugging, version control integration, and support for web technologies like HTML, CSS, making them suitable for developing frontend components of web applications. For installation please refer Chapter 02 and Chapter 08 for Visual Studio and PyChar respectively

4. Flask (Frameworks):

Flask for the frontend. Flask is a popular web framework for building web applications using Python. It is a lightweight and flexible

framework that follows the microservices architecture, making it suitable for our project For Installation please refer chapter 07

5. **Image Processing Libraries:**

OpenCV, Pillow for manipulation and processing. OpenCV: A computer vision library that offers extensive functionality for image preprocessing, manipulation, and analysis. PIL (Python Imaging Library): A library for image processing tasks, including loading, resizing, and basic image operations.

6. **Deep Learning Frameworks:**

TensorFlow, Keras, for implementing deep learning models for pneumonia detection. TensorFlow: A popular deep learning framework that provides tools for building and training neural networks. For installation refer Chapter 07 Keras: A high-level neural networks API that runs on top of TensorFlow, allowing for easy implementation of deep learning models.

7. **Version Control:**

GitHub as a platforms like for managing source code. GitHub is a web-based platform and hosting service that allows developers to manage and collaborate on source code. It provides version control functionality using Git, enabling teams to track changes, manage branches, and merge code seamlessly, facilitating efficient code collaboration and project management.

8. **Testing:**

Tools like PyTest for unit testing and functional testing. PyTest is a popular testing framework in Python that supports unit testing, functional testing, and integration testing. It offers a simple and expressive syntax, powerful fixtures, and extensive plugin ecosystem, making it a versatile tool for testing Python code and ensuring its quality and reliability.

Deployment:

1. **Cloud Platforms:**

Google Cloud Platform (GCP) for hosting and deploying the web app. Google Cloud Platform (GCP) is a suite of cloud computing services provided by Google. It offers a wide range of hosting and deployment options for web applications, including virtual machines, managed Kubernetes clusters, serverless functions, and storage services, enabling scalable and reliable deployment of web apps with robust infrastructure and services.

Figure 11.1.: **Saving Model Architecture**

2. Serverless Computing:

Google Cloud Functions for running serverless code. Google Cloud Functions is a serverless computing platform provided by Google Cloud Platform (GCP). It allows developers to write and deploy small, event-driven functions without the need to manage infrastructure. It automatically scales the functions based on demand, making it easy to build and run serverless code in a cost-effective and scalable manner.

11.2. Saving the Model

Saving model consist following steps :

Step 1: **Train and Validate the Model**

Train your pneumonia detection model using a dataset of labeled examples. Split the dataset into training and validation sets for evaluation. Ensure that the model achieves satisfactory performance on the validation set.

Step 2: **Choose a Model Serialization Format**

we will save the model's architecture and weights in a Python .py file.

Step 3: **Save the Model Architecture**

In your Python script or notebook, define the architecture of your trained model. Include all the layers, their configurations, and any custom functions or classes.

Step 4: **Train and Save the Weights :**

Retrain the model using the entire dataset, including the validation set. Once the training is complete, save the learned weights of the model.

Step 5: **Save the Model in a Python Script**

Create a new Python script, let's name it `pneumonia_model.py`, and copy the model architecture and weight initialization code into it.

Figure 11.2.: Saving Model For Python Scrip

11.3. File Structure

The file structure of the project consists of the following components:

- **app.py :**

This script serves as the engine of the web application and is responsible for running the application. It contains an API that receives input from the user and computes a predicted value based on the model. This file is typically written using the Flask framework in Python, allowing it to handle incoming requests and generate appropriate responses.

- **prediction.py :**

This file contains the code to build and train a machine learning model. It may include data preprocessing, feature engineering, model selection, and training procedures. The trained model can be used in the app.py script to make predictions based on user input.

- **templates folder (.html):**

This folder contains two HTML files: home.html and base.html. home.html defines the structure and layout of the web application's main page. base.html serves as a base template that other HTML files can inherit from, allowing for code reusability and consistent design across multiple pages.

- **static folder (.css) :**

This folder contains the style.css file, which is responsible for adding styling and enhancing the visual appearance of the web application. The CSS file defines rules and styles for HTML elements, allowing for customization of fonts, colors, layouts, and other visual aspects.

11.4. Loading Model

- **Integration with the Web Application:**

In image processing web application code, import the pneumonia_model.py script that contains the model architecture and loading code. Ensure that the necessary dependencies (e.g., TensorFlow, Keras) are available in the application's environment.

Figure 11.3.: **Loading The Model**

- **Load the Model:**

Use the provided functions or classes from the `pneumonia_model.py` script to load the model into your web application.

11.5. Description Of Process

Fig. 11.4 shows over view of the process. The process for developing and deploying a pneumonia detection web app involves several steps. Here is a description of the process:

1. **Requirements Gathering:**

Define the requirements for the pneumonia detection web app, including the desired features, user interface, and performance criteria. Understand the target audience and any specific needs or constraints.

2. **Data Collection and Preparation:**

Gather a dataset of chest X-ray images with labeled pneumonia and non-pneumonia cases. Preprocess the data by resizing, normalizing, and augmenting the images as needed. Split the dataset into training, validation, and testing subsets.

3. **Model Development:**

Choose an appropriate machine learning or deep learning algorithm for pneumonia detection. Train the model using the labeled dataset and evaluate its performance on the validation set. Fine-tune the model and iterate until satisfactory results are achieved.

4. **Front-end Development:**

Create the user interface and design the visual elements of the web app. Develop the front-end using HTML, CSS, and JavaScript frameworks like Flask. Implement features such as image upload, result display, and user interactions.

5. **Back-end Development:**

Build the back-end of the web app using a server-side framework like Flask or Django. Implement the necessary APIs and endpoints to handle image uploads, process the images using the trained model, and return the results to the front-end.

Figure 11.4.: **Description Of Process**

6. Integration and Testing:

Connect the front-end and back-end components, ensuring seamless communication and functionality. Conduct rigorous testing to verify the correctness of the app, including unit testing, integration testing, and user testing. Fix any issues or bugs identified during testing.

7. Deployment:

Choose a suitable hosting platform or cloud service provider to deploy the web app. Set up the necessary infrastructure, configure the server, and ensure the app is accessible over the internet. Consider security measures, such as HTTPS encryption and access controls, to protect user data.

8. Continuous Monitoring and Improvement:

Monitor the performance and usage of the web app in the production environment. Collect user feedback and analytics to identify areas for improvement. Continuously update and optimize the app based on user needs and emerging technologies.

9. Maintenance and Support:

Provide ongoing maintenance and support for the web app, addressing any issues, bugs, or feature requests that arise. Stay updated with the latest advancements in pneumonia detection algorithms and technologies to ensure the app remains accurate and efficient.

12. Deployment

12.1. User Interface

In this Project for the deployment of our model, we are going to use WEB APPLICATION. the web application allows users to upload X-ray images and receive a classification result indicating whether the image shows signs of pneumonia. It allows patients to upload their X-ray images and tell them the results. For creating the framework for the web app, we will use Visual Studio where the user gives the input in the form of chest x-ray images and get the result as output.

The figure 8.1.: shows the road map for the web app. The user will upload an Image via a webpage and after submitting the image, the image will go to the Flask server where our trained model will be used to make predictions. After the predictions are made the response will be sent back to the webpage via the server.

- The figure 8.2.: provides an overview of how the website might appear. The user can upload the X-ray image here in the Choose file area as input.
- The data is sent to the server as it is provided to the web application. Any prediction made after that will be displayed as an output result. The figure 8.3.: displays a healthy outcome.

12.2. Work flow

The proposed workflow model of this network is demonstrated in figure 8.1.:, where firstly, the acquired labeled data are preprocessed, then split into 80:20, 70:30, and 60:40 ratios of test and train, applied with image augmentation properties. Furthermore, the images are subjected to various pre-trained models such as VGG16, VGG19, and Xception that involve transfer learning techniques. If the accuracy is not adequate in one ratio, the images are trained again with a different set of ratios. The models are chosen based on their accuracy and other evaluations.[jain2022deep]



Figure 12.1.: Road Map For Web App
Reference:[Alhameed:2022]

Figure 12.2.: Web Page
Reference:Author

Figure 12.3.: Result
Reference: Author

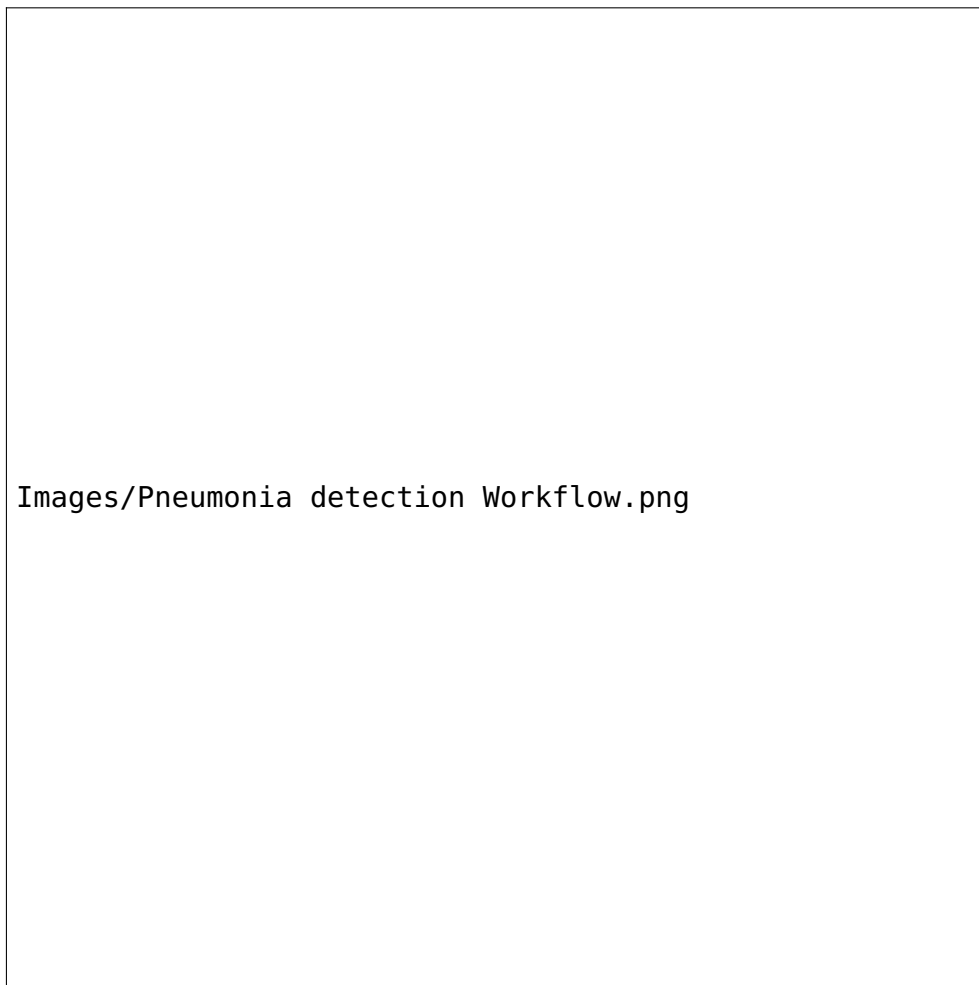


Figure 12.4.: **Deployment approach flow chart**
Reference:[jain2022deep]

Table 12.1.:
Accuracy of proposed networks

Network	Training and Test Ratio	Training Acc.	Validation Acc.
CNN	80-20	0.89	0.93
	70-30	0.90	0.94
	60-40	0.89	0.90
Xception	80-20	0.86	0.94
	70-30	0.86	0.93
	60-40	0.82	0.86
VGG16	80-20	0.78	0.94
	70-30	0.78	0.93
	60-40	0.76	0.88
VGG19	80-20	0.91	0.93
	70-30	0.92	0.88
	60-40	0.91	0.89

Reference:[jain2022deep]

13. Monitoring the Model

To make accurate predictions and get new insights in the fast-paced world, machine learning models must be updated with the most recent data. This chapter examines a thorough monitoring strategy that employs Python code to find fresh data, run the required tests, and update a trained model appropriately.

13.1. Plan

By routinely scanning a dataset directory for new data, the monitoring approach tries to keep the pneumonia detection model current. The steps in the procedure are as follows:

1. Count the initial number of files in the dataset directory.
2. Wait for a specific period to allow new data to be added.
3. Count the current number of files in the dataset directory.
4. Compare the initial and current file counts to detect the presence of new data.
5. Update the model and create a backup of the dataset if new data is detected.

13.2. Process Description

The monitoring process for pneumonia detection consists of the following key elements:

1. Import necessary libraries such as `os`, `shutil`, `schedule`, `time`, `datetime`, and `tensorflow.keras.models.load_model`.
2. Define essential functions: `update_model()`, `backup_data()`, `monitor_and_update()`, and `start_monitoring()`.
3. Load the pre-trained machine learning model using `load_model()`.
4. Start the monitoring process with the `start_monitoring()` function.

13.3. New Data

The script examines the initial and current file counts in the dataset directory to find fresh data. The presence of new data is indicated if the current count is higher. By using this method, the pneumonia detection model is kept current with new knowledge and is able to make precise predictions.

13.4. Checks

To perform checks and determine if new data is present, the following steps are executed within the `monitor_and_update()` function:

1. The initial number of files is stored in a variable named `initial_files`.
2. The script waits for a specified period using `time.sleep(10)` to allow for the addition of new data.
3. The current number of files is stored in a variable named `current_files`.
4. If `current_files` is greater than `initial_files`, it signifies the presence of new data.
5. Update the model and create a backup of the dataset if new data is detected.

13.5. Functions

The code includes several essential functions for pneumonia detection:

1. `update_model()`: This function implements the necessary logic to update the pneumonia detection model based on the requirements of the specific machine learning task. It incorporates data preprocessing, retraining the model, and fine-tuning as needed.
2. `backup_data()`: This function creates a backup of the pneumonia dataset by copying it to a backup directory. It uses the `shutil.copytree()` method to ensure that the entire dataset directory structure is preserved.
3. `monitor_and_update()`: This function is responsible for monitoring the pneumonia dataset directory for new data. It compares the initial and current file counts and triggers the model update process if new data is detected. It calls the `update_model()` and `backup_data()` functions accordingly.

4. **start_monitoring()**: This function initiates the monitoring process for pneumonia detection. It sets up a schedule using the **schedule.every()** method to run the **monitor_and_update()** function at specified intervals. The function runs indefinitely, continuously checking for pending scheduled tasks.

13.6. Program

The program below shows the monitoring program implemented in the pneumonia detection application developed.

Listing 13.1: Check.py

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
from imutils import build_montages

def plot_distribution(paths: list, portion: str):
    healthy_path = paths[0]
    pneumonia_path = paths[1]

    numbers = [
        len(os.listdir(healthy_path)),
        len(os.listdir(pneumonia_path))
    ]
    labels = ["Healthy", "Pneumonia"]

    plt.bar(labels, numbers)
    plt.title(portion + "_set_distribution")
    plt.savefig(portion + "_distribution.png")
    plt.show()

def plot_montage(paths: list):
    h_files = paths[0]
    p_files = paths[1]

    healthy_images = []
    pneumonia_images = []

    for path in h_files:
        h_img = cv2.imread("chest_xray\\train\\NORMAL\\" +
                           path)
        healthy_images.append(h_img)
```

```
for path in p_files:
    p_img = cv2.imread("chest_xray\\train\\PNEUMONIA\\" +
                       +path)
    pneumonia_images.append(p_img)

healthy_images = np.array(healthy_images)
pneumonia_images = np.array(pneumonia_images)
cv2.imshow("", healthy_images[0])
cv2.waitKey(0)
healthy = build_montages(healthy_images,
                        (300,300),(3,3))
pneumonia = build_montages(pneumonia_images,
                        (300,300),(3,3))

for montage in healthy:
    cv2.imshow("Healthy", montage)
    cv2.waitKey(0)

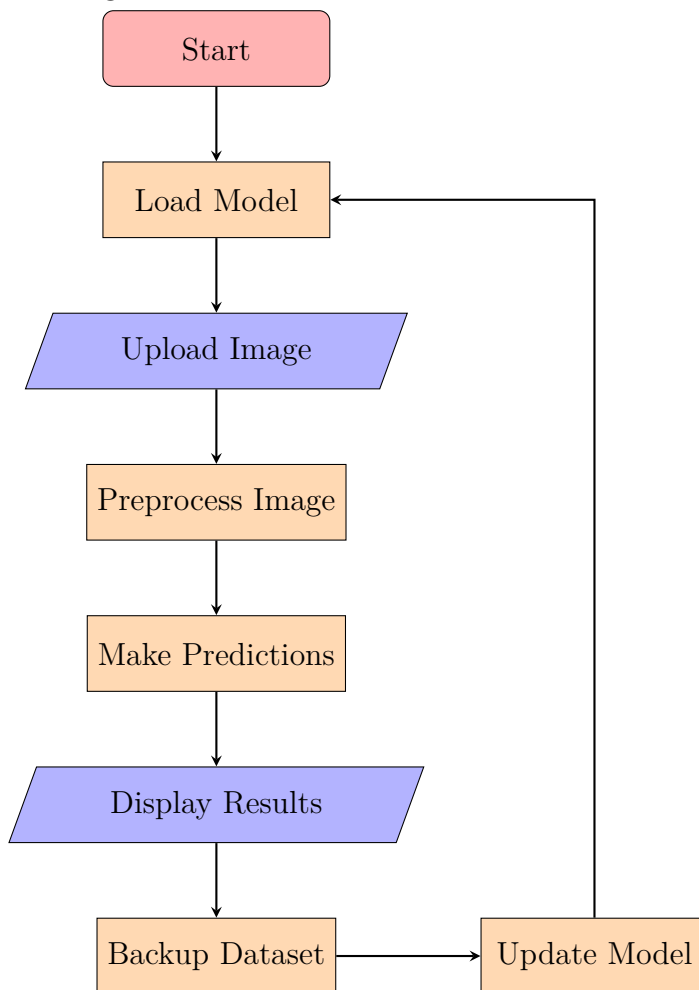
for montage in pneumonia:
    cv2.imshow("Pneumonic", montage)
    cv2.waitKey(0)

plot_distribution(["chest_xray\\train\\NORMAL",
                  "chest_xray\\train\\PNEUMONIA"], "train")
plot_distribution(["chest_xray\\test\\NORMAL",
                  "chest_xray\\test\\PNEUMONIA"], "test")

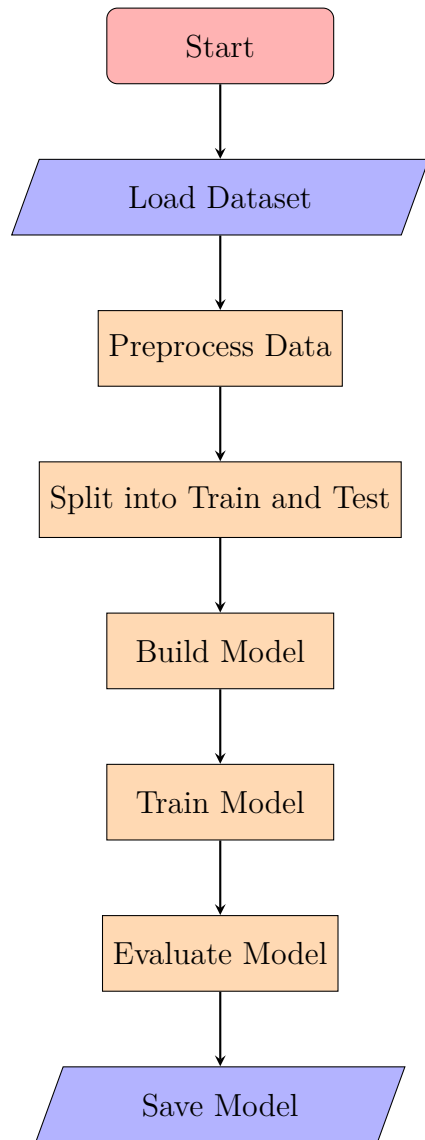
plot_montage([list(os.listdir("chest_xray\\train\\NORMAL")[:9]),
                 list(os.listdir("chest_xray\\train\\PNEUMONIA")[:9]))])
```

14. Programme Flow

Programme Flow Chart



Model Training Flow Chart



15. Application

The Pneumonia Detection Application is a web-based tool that allows users to upload chest X-ray images and determine whether the person in the image is healthy or has pneumonia. To produce precise predictions, the application employs a deep learning model that was trained on a dataset of chest X-ray pictures. It has a user-friendly interface that makes it simple for users to upload photographs, view prediction results, and navigate the application.

The Flask web framework and Python are used to create the application. It integrates HTML and CSS for the frontend interface and uses the TensorFlow library for deep learning operations. The application's backend manages image submissions, makes predictions with the trained model, and displays the suitable HTML templates to present the outcomes.

15.1. Operating System

Operating systems like Windows, macOS, and Linux are all compatible with the Pneumonia Detection Application. A web browser like Google Chrome, Mozilla Firefox, or Safari can be used to access it. The program adheres to a client-server architecture, in which server-side code is executed on the backend and HTML templates are sent to the client-side for rendering.

The application uses the HTTP protocol for communication between the client and server. The Flask web framework handles the HTTP requests and responses, allowing users to interact with the application through standard web interfaces. The application utilizes HTML, CSS, and JavaScript for the frontend user interface and Flask for server-side processing.

15.2. Constraints

While the Pneumonia Detection Application offers a convenient way to predict pneumonia from chest X-ray images, it does have certain constraints:

1. **Accuracy Limitations:** The accuracy of the predictions depends on the performance of the underlying deep learning model. While



Images/UserInterface.png

Figure 15.1.: **User Interface**
Reference:Author

the model has been trained on a large dataset, there may be cases where the predictions are incorrect or inconclusive. Users should not solely rely on the application's results for medical diagnoses and consult with healthcare professionals for accurate assessments.

2. **Hardware Requirements:** The application requires a computer system with sufficient resources to run the Python code and load the trained model. A system with a dedicated GPU can significantly speed up the prediction process. However, it can still be run on a CPU, although it may take longer to process the images.
3. **Image Quality:** The accuracy of the predictions can be affected by the quality of the uploaded chest X-ray images. Blurry or low-resolution images may lead to less reliable results. It is recommended to use high-quality images for better accuracy.

15.3. Input Data

The Pneumonia Detection Application relies on a dataset of chest X-ray images for training the deep learning model. The dataset used to train the model should meet certain criteria to ensure its effectiveness:

1. **Quality:** The X-Ray image should consist of high-quality images with clear visibility of the lung area. Images with artifacts, noise, or poor resolution may negatively impact the accuracy of the predictions.
2. **Quantity:** Only one image can be uploaded for the detection at a time. Multiple uploads is not accepted and is not supported.
3. **Types:** The Input data should be Images with extension of png, jpeg or jpg.
4. **Structure:** The Image should be labelled properly for which should not be too long or too short and follows a particular naming convention.

15.4. Conclusions

The Pneumonia Detection Application provides a user-friendly and accessible platform for predicting pneumonia from chest X-ray images. It leverages deep learning techniques and a trained model to analyze the images and provide predictions. While it is not a substitute for professional medical advice, it can be a helpful tool for preliminary assessments or as a reference for further medical evaluations. Users should exercise

caution and consult with healthcare professionals for accurate diagnoses and treatment recommendations.

16. Open Questions

In the context of pneumonia detection using ML models, there are several open questions and future directions for research that can be explored. Some of these include:

1. Can the model be further improved by using more advanced transfer learning techniques?
2. Can the model be trained on additional data sources to improve its accuracy and reliability?
3. Can the model be adapted for use in real-time pneumonia detection applications?
4. Can the model be modified to detect other respiratory conditions, such as tuberculosis or COVID-19?
5. How can the model be made more interpretable for medical professionals to better understand its predictions?
6. Can we develop a real-time pneumonia detection application that can provide accurate and timely diagnoses to patients and medical professionals?
7. Can the model be Integrated into electronic medical records (EMRs) to provide seamless and efficient pneumonia diagnosis and treatment recommendations?

17. Conclusion

17.1. Conclusion

In terms of accuracy, our proposed pneumonia detection system, which incorporates supervised ML algorithms, outperforms previously existing methods. This enhancement was obtained by implementing a convolutional neural network (CNN) trained on a huge dataset of X-ray images.

We can investigate the usage of advanced image processing techniques such as texture analysis, form analysis, and edge detection to further increase the accuracy of our system. Additionally, uploading X-ray images in a specific orientation can help standardize the dataset and improve overall system efficiency.

To boost accuracy, we can apply multi-class classification algorithms to the dataset, such as SVM, Random Forest, and K-Nearest Neighbor. These algorithms have been demonstrated to be effective in image categorization tasks.

Finally, it is crucial to mention that our application is created with the purpose of safeguarding the best interests of patients and healthcare professionals in mind. We believe that our technology can assist healthcare providers in swiftly and accurately diagnosing pneumonia, resulting in better patient outcomes.

17.2. Next Step

The potential future work that can be done is mentioned below

- Artificial intelligence will make it possible to progressively move towards automating repetitive and time-consuming tasks, such as screening for lung nodules and identifying image-based biomarkers. These developments will optimistically allow disease characterization in a non-invasive and repeatable way, improving therapeutic management in order to achieve the goal of personalized medicine [cellina2022artificial]

- In the future, we may investigate techniques such as contrast enhancement of the images or other pre-processing steps to improve the image quality. We may also consider using segmentation of the lung image before classification to enable the CNN models to achieve improved feature extraction [**kundu2021pneumonia**]
- While X-ray imaging is widely used for pneumonia detection, other imaging modalities such as CT scans and ultrasound can also be used. Future work can focus on developing multi-modal approaches that combine information from different imaging modalities for more accurate and reliable diagnosis
- Real-world Deployment:- Preparing the pneumonia detection system for real-world deployment by considering deployment environments, scalability, security and regulatory compliance.
- Validation:- Considering factors such as sensitivity, specificity, and impact on patient outcomes we need to conduct clinical trials and validation studies to assess the performance and clinical utility of the pneumonia detection.
- Monitoring and Model Updates:- To ensure ongoing accuracy and effectiveness the system need to be monitor and update the model on regular basis
- Real-time Detection: Investigate techniques to enable real-time pneumonia detection, allowing for immediate decision-making and intervention. This may involve optimizing the model architecture and employing efficient inference techniques to achieve low-latency performance.
- Deployment in Resource-Constrained Settings: Focus on adapting the pneumonia detection system for resource-constrained environments, such as low-resource healthcare settings or areas with limited access to medical facilities. This may involve optimizing the model's computational requirements, developing lightweight versions of the algorithm, or exploring edge computing solutions

17.3. To Do

- For creating a more productive ML model, it is required to have a larger volume and variety of datasets to train the model. The lab-based datasets are limited to be used for effective training of ML models in a real-time scenario like in hospitals or medical institutions. Therefore, we need to have a solution of using real-time data to fulfill the requirements of having a more significant variety of data.[kareem2022review]
- Clinical and demographic data, such as patient age, medical history, and symptoms, can provide important context for pneumonia diagnosis. Future work can focus on integrating this information into the image-based deep learning models to improve their accuracy. [han2021pneumonia]
- Data collection:- To enhance the performance and generalization we need to gather a larger and more diverse dataset for training and validation.
- Data validation and Evaluation:- Collaborate with healthcare professionals to validate the accuracy and clinical relevance of the pneumonia detection. This is help to refine the algorithm and address any potential limitations, ensuring that the system aligns with the requirements and standards of medical practice.
- Improvement:- Improve the process of determining how certain or confident a pneumonia detection is about its predictions. Need to make sure how the system is about whether an image contains pneumonia or not. If the uncertainty is low, it means the system is very confident in its prediction. On the other hand, if the uncertainty is high, it indicates that the system is less certain about its prediction.
- Transfer Learning from Multiple Domains: Instead of relying solely on a single pre-trained model, explore transfer learning from multiple domains or datasets. This can involve utilizing pre-trained models trained on different medical imaging datasets or even non-medical image datasets that exhibit similar visual patterns or structures.

17.4. Unanswered points

- **Real-world Variability:** Account for real-world variability and challenges in pneumonia detection, such as imaging artifacts, variations in patient positioning, or image quality issues. Augment the dataset or develop robust algorithms that can handle these common challenges encountered in clinical practice.
- **Multi-Modality Fusion:** Explore approaches for integrating information from multiple imaging modalities, such as chest X-rays and CT scans, to improve the accuracy and reliability of pneumonia detection. Fusion techniques can leverage complementary information from different modalities and enhance the overall performance.
- **Uncertainty Estimation:** Develop techniques to estimate uncertainty in pneumonia detection models. Uncertainty estimation provides insights into the confidence or reliability of model predictions, enabling clinicians to make more informed decisions based on the model's output.
- **External Validation and Generalization:** Conduct external validation of the pneumonia detection model on diverse and independent datasets to assess its generalization capabilities. Evaluate the model's performance across different patient populations, geographic regions, or healthcare institutions to ensure its reliability and effectiveness in real-world scenarios.

A. Appendix

