



EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPARTMENT OF NUMERICAL ANALYSIS

# Pneumonia Detection with Convolutional Neural Networks and a Web-Based Software implementation

*Supervisor:*

Dr. Béla János Szekeres

Senior Lecturer

*Author:*

Bálint Molnár

Computer Science BSc

*Szombathely, 2023*

The aim of this thesis is to provide a user-friendly and efficient solution for accurately detecting pneumonia from chest X-ray images through an interactive web-based interface. The program currently operates on a demo database, allowing for the storage of patient data. Chest X-ray images uploaded to the system undergo analysis through a trained convolutional neural network model, producing a precise diagnosis that is then saved to the corresponding patient's profile. The implementation of neural networks significantly reduces the time required for various healthcare tasks, particularly in the area of patient prioritization. By quickly sorting large quantities of data into infected and non-infected categories, doctors are better equipped to make informed treatment decisions.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>User documentation</b>	<b>5</b>
2.1	Dependencies . . . . .	5
2.2	Installing Dependencies . . . . .	6
2.3	Starting the application for the first time . . . . .	6
2.4	Accessing and using the web application . . . . .	6
2.4.1	Adding new patients . . . . .	7
2.4.2	Editing existing patients . . . . .	8
2.4.3	Removing patients . . . . .	9
2.4.4	Diagnosing Patients' photos . . . . .	10
2.4.5	Removing diagnoses . . . . .	13
2.4.6	Adding treatments . . . . .	13
2.4.7	Removing treatments . . . . .	14
2.4.8	Routes of the website . . . . .	15
<b>3</b>	<b>Developer documentation</b>	<b>16</b>
3.1	Description of the problem . . . . .	16
3.2	Folder hierarchy . . . . .	17
3.3	Convolutional Neural Network . . . . .	17
3.3.1	Libraries . . . . .	17
3.3.2	Global variables . . . . .	19
3.3.3	Data . . . . .	19
3.3.4	Convolutional neural network model . . . . .	22
3.3.5	Functions . . . . .	25
3.3.6	Graphs . . . . .	26
3.4	Web application . . . . .	29
3.4.1	Libraries . . . . .	29

3.4.2	Global variables . . . . .	30
3.4.3	Application functions . . . . .	31
3.4.4	Route functions . . . . .	37
3.5	Database . . . . .	40
3.5.1	Introduction . . . . .	40
3.5.2	Tables . . . . .	40
3.5.3	Database table column descriptions . . . . .	41
3.6	Tests . . . . .	44
3.6.1	Manual Testing for CNN . . . . .	44
3.6.2	Manual Testing for Flask Routes . . . . .	45
<b>4</b>	<b>Conclusion</b>	<b>46</b>
	<b>Bibliography</b>	<b>47</b>
	<b>List of Figures</b>	<b>49</b>
	<b>List of Tables</b>	<b>50</b>

# Chapter 1

## Introduction

Growing up around healthcare workers, I witnessed firsthand the heavy workload and responsibility that comes with their profession. One of the most challenging aspects of their work is accurately diagnosing patients. The 2020 COVID pandemic highlighted the strain on healthcare workers who had to review countless chest X-ray images, taking valuable time away from treating patients. While some cases can be diagnosed with a quick glance, others require a more complex evaluation. This is where AI and neural networks can be immensely beneficial. My thesis focused on developing a software solution that uses neural networks to accurately predict pneumonia and provide a simple web-based user interface. However, the potential applications of AI in healthcare extend beyond pneumonia diagnosis. Imagine a software solution that could accurately detect various skin diseases, health issues, and internal problems from ultrasounds. In a world where we face a shortage of doctors and nurses, leveraging AI and neural networks in healthcare could reduce the workload of healthcare workers and allow them to focus on their priorities. Providing quick and accurate diagnoses would enable doctors to better prioritize patients based on their level of urgency. The use of AI and neural networks has become increasingly widespread in various fields, and I believe healthcare is one of the most promising areas.

"Simplification is one of the most difficult things to do".[1](Ive, 2014)

My thesis primarily focused on predicting pneumonia from chest X-ray images using convolutional neural network Models. To achieve this, a substantial dataset of X-ray images was required to train the models effectively. I sourced my dataset from mendeley "Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images

for Classification"[2]. These were screened for quality control by removing unreadable images and then split into training, testing and validation datasets. Additionally, the images were evaluated by three expert physicians who graded the diagnoses. The images were taken at the Guangzhou Women and Children's Medical Center in Guangzhou, and the patients were between the ages of 1 and 5 years old. This dataset formed the basis of my research, and I used it to train my convolutional neural network Models to accurately predict pneumonia from chest X-ray images. Furthermore the book called '*Dive into Deep Learning*'[3] was also a big inspiration and help during the creation of the project. All the project information, codes, and this documentation can be found on my GitHub[4] page.

# Chapter 2

## User documentation

The web application was developed using the Flask Python micro web framework and utilizes the Waitress WSGI (Web Server Gateway Interface) server, as well as the SQLite database.

### 2.1 Dependencies

**Python3** [5] - programming language used for website back-end development

**Flask** [6] - used to develop the web application

**SQLite3** [7] - used to access the database

**Werkzeug** [8] - used to secure filenames

**OpenCV** [9] - cv2 is used to read, resize, color images

**NumPy** [10] - used to convert images into arrays

**Keras** [11] - used to load the prediction models

**Waitress** [12] - used to host the application server

(*Note:* The dependency references listed here are the official documentations of the respective libraries.)

## 2.2 Installing Dependencies

1. Python has to be installed to be able to run the web application. <https://www.python.org/downloads/>
2. After installing Python, the dependencies listed above also need to be installed. Use the `'pip install <library-name>'` command in a terminal to install all of the dependencies. ex.:

```
1 $ pip install waitress
```

## 2.3 Starting the application for the first time

1. When running for the first time the database needs to be initialized with the `'init_db.py'` file. To run this file, open a terminal, navigate to the root directory of the application, and execute the following command:

```
1 $ python ./init_db.py
```

2. To host the server, open a terminal, navigate to the directory containing the `app.py` file, and use the command:

```
1 $ python ./app.py
```

After running this command, the application can be accessed at <http://localhost:50100/>

## 2.4 Accessing and using the web application

Navigate to the <http://localhost:50100/> link in a web browser of your choice and you will arrive at the homepage of the web application.



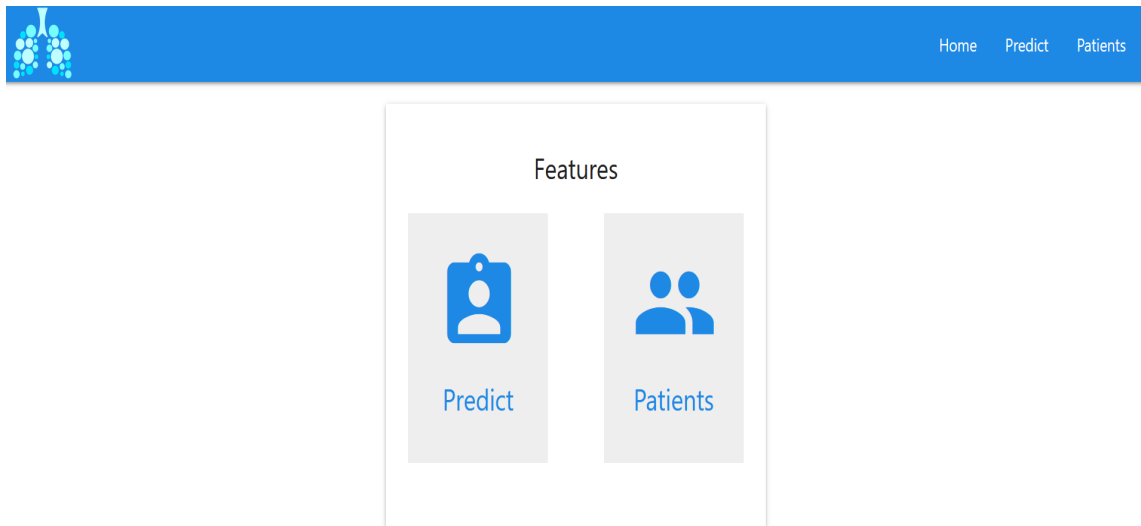


Figure 2.1: Homepage of the website

### 2.4.1 Adding new patients

Using the cards or the navigation menu at the top, navigate to the Patients tab. Once there, click on the 'Add New' button located at the bottom of the page. A modal will appear where you need to provide the necessary information about the patient and click the 'Add' button to save the new patient.

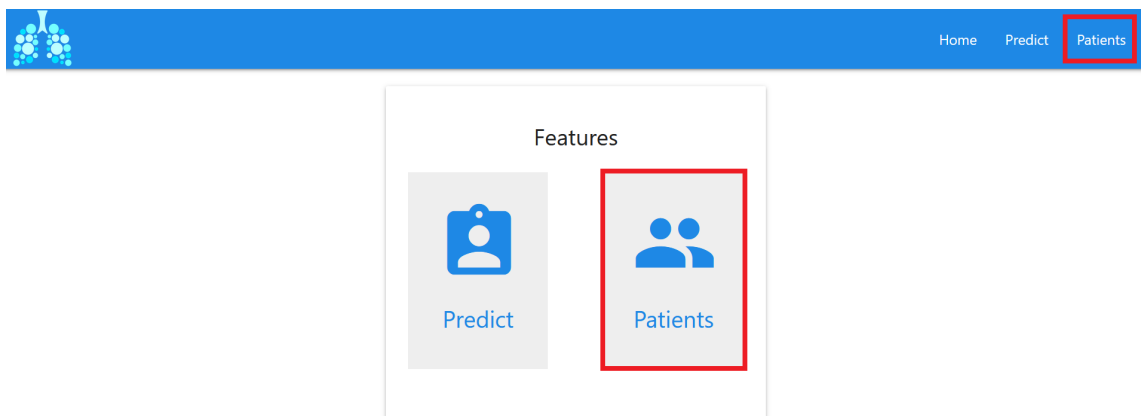


Figure 2.2: Navigation to add new patient

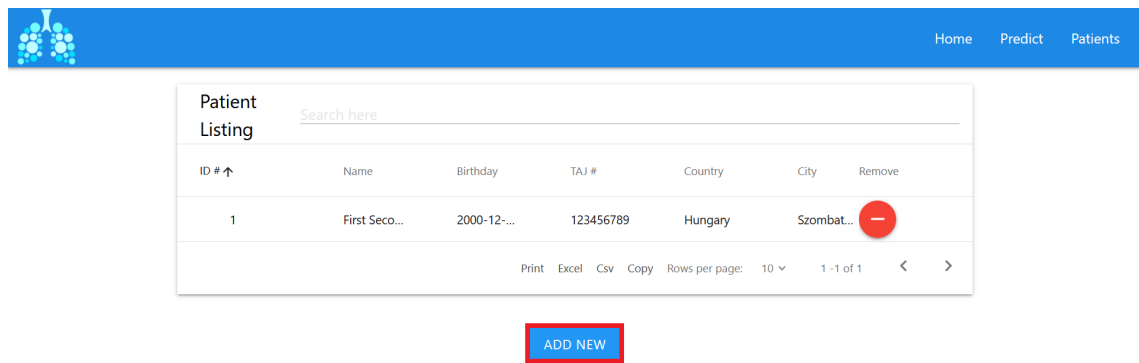


Figure 2.3: Adding new patient

The screenshot shows the 'ADD NEW' patient form. At the top, there is a header bar with the same navigation links as Figure 2.3. Below the header, there is a table with the same data as Figure 2.3. Below the table, there is a form with the following fields: First Name, Middle Name, Last Name, Birthday (mm/dd/yyyy), Email, Mobile, TAJ, Post Code, Country, City, and Street. There is a red minus icon in the Remove column. At the bottom, there is a blue button labeled 'ADD' with a red border, and a 'CLOSE' link.

Figure 2.4: Adding new patient form

### 2.4.2 Editing existing patients

To edit a patient, first find the patient's row in the table and click on it. This will take you to the edit page where you can modify the patient's information. Once you're done, click the 'Update' button to save the changes.

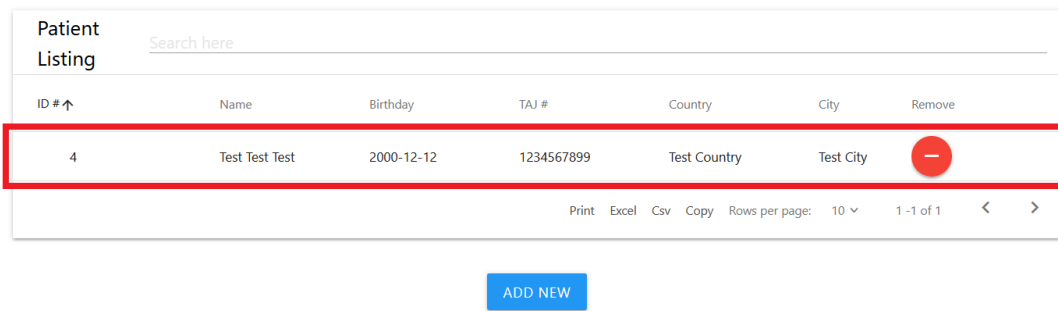


Figure 2.5: Editing existing patient

The screenshot shows a form for editing a patient. It includes input fields for Country (Test Country), City (Test City), Postal Code (TEST123), and Street (Test Street). Below these fields is a blue 'UPDATE' button, which is highlighted with a red border. Further down, there are two blue buttons: 'ADD DIAGNOSES' and 'ADD TREATMENT'. At the bottom, there is a table with columns: Diagnosis ID, Patient ID, Name, TAJ #, Results, Pictures, Diagnosis ID, Treatment ID, Treatment, Issued By, Created At, and Remove.

Figure 2.6: Editing existing patient

### 2.4.3 Removing patients

To delete a patient navigate to the patients tab, then click the red button on the patients row.

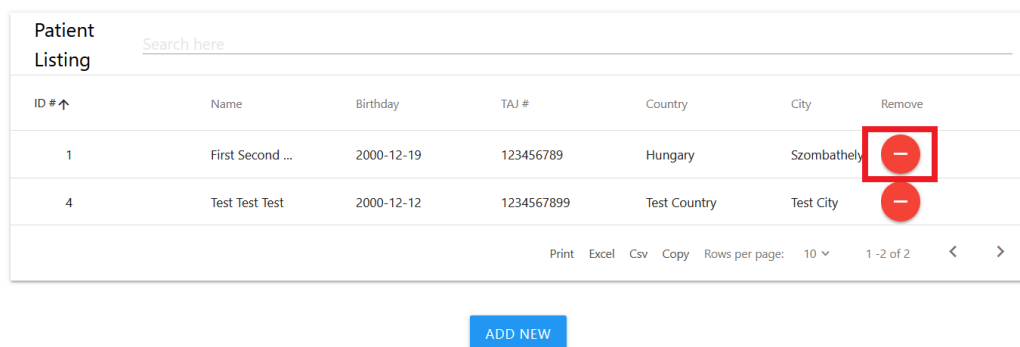


Figure 2.7: Removing patients

### 2.4.4 Diagnosing Patients' photos

To diagnose a photo from the homepage or navigation bar, click the 'Predict' button and select a patient from the table. Alternatively, you can also add diagnoses from the 'Edit Patient' page. To upload a photo, use the 'Browse' button and then click 'Submit'. Click the 'Predict' button to begin the prediction process. The diagnoses will be added to the database and displayed on the right side of the prediction page. Alternatively, you can view them under the 'Edit Patient' page.

Patient Datable <small>Search here</small>					
ID # ↑	Name	Birthday	TAJ #	Country	City
4	Test Test Test	2000-12-12	1234567899	Test Country	Test City

Print Excel Csv Copy Rows per page: 10 ▾ 1 -1 of 1 < >

Figure 2.8: Choosing patient for prediction

Country	City	Postal Code	Street
Test Country	Test City	TEST123	Test Street

UPDATE

ADD DIAGNOSES ADD TREATMENT

Diagnosis ID	Patient ID	Name	TAJ #	Results	Pictures	Diagnosis ID	Treatment ID	Treatment	Issued By	Created At	Remove
--------------	------------	------	-------	---------	----------	--------------	--------------	-----------	-----------	------------	--------

Figure 2.9: Choosing patient for prediction

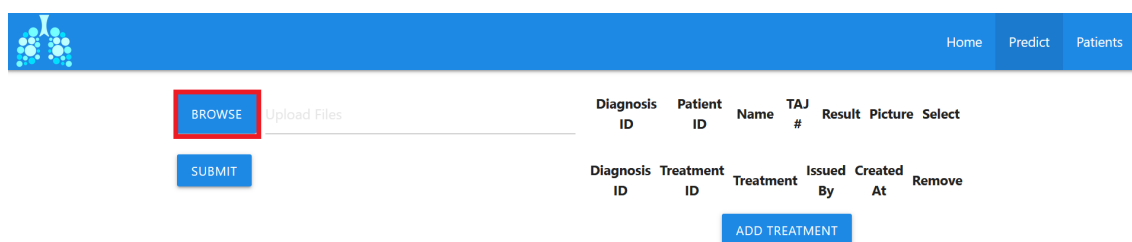


Figure 2.10: Browse button click

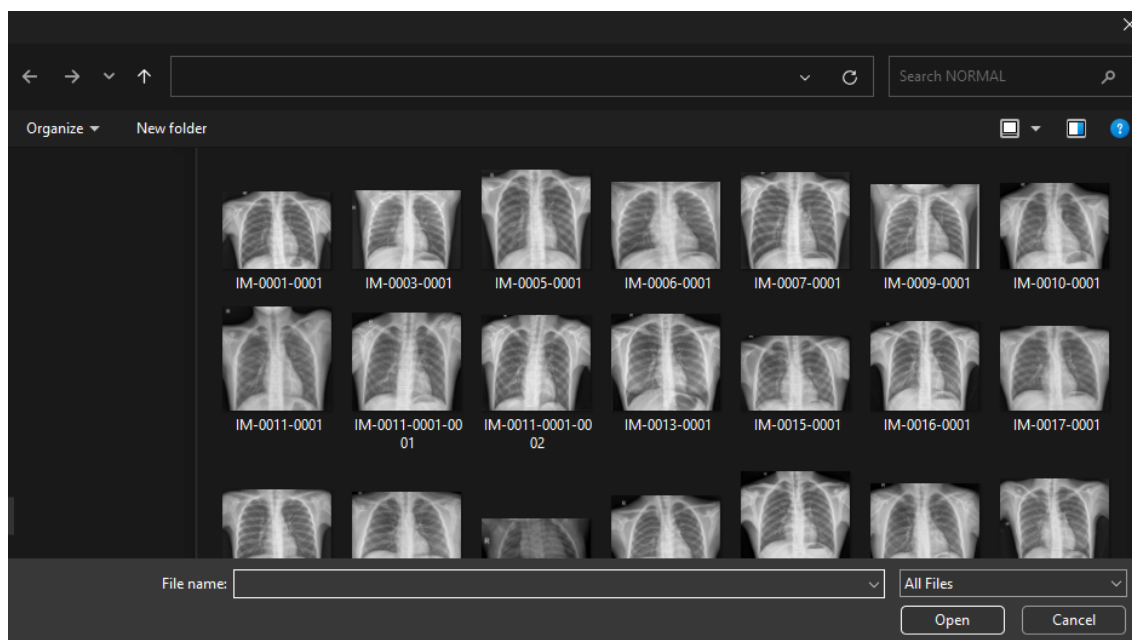


Figure 2.11: Selecting a file to upload

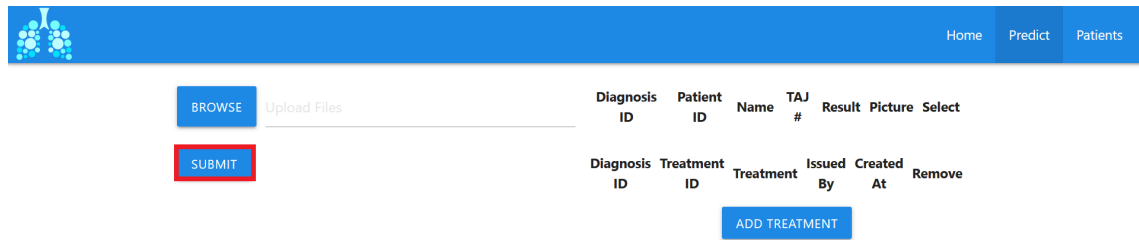


Figure 2.12: Submit button click

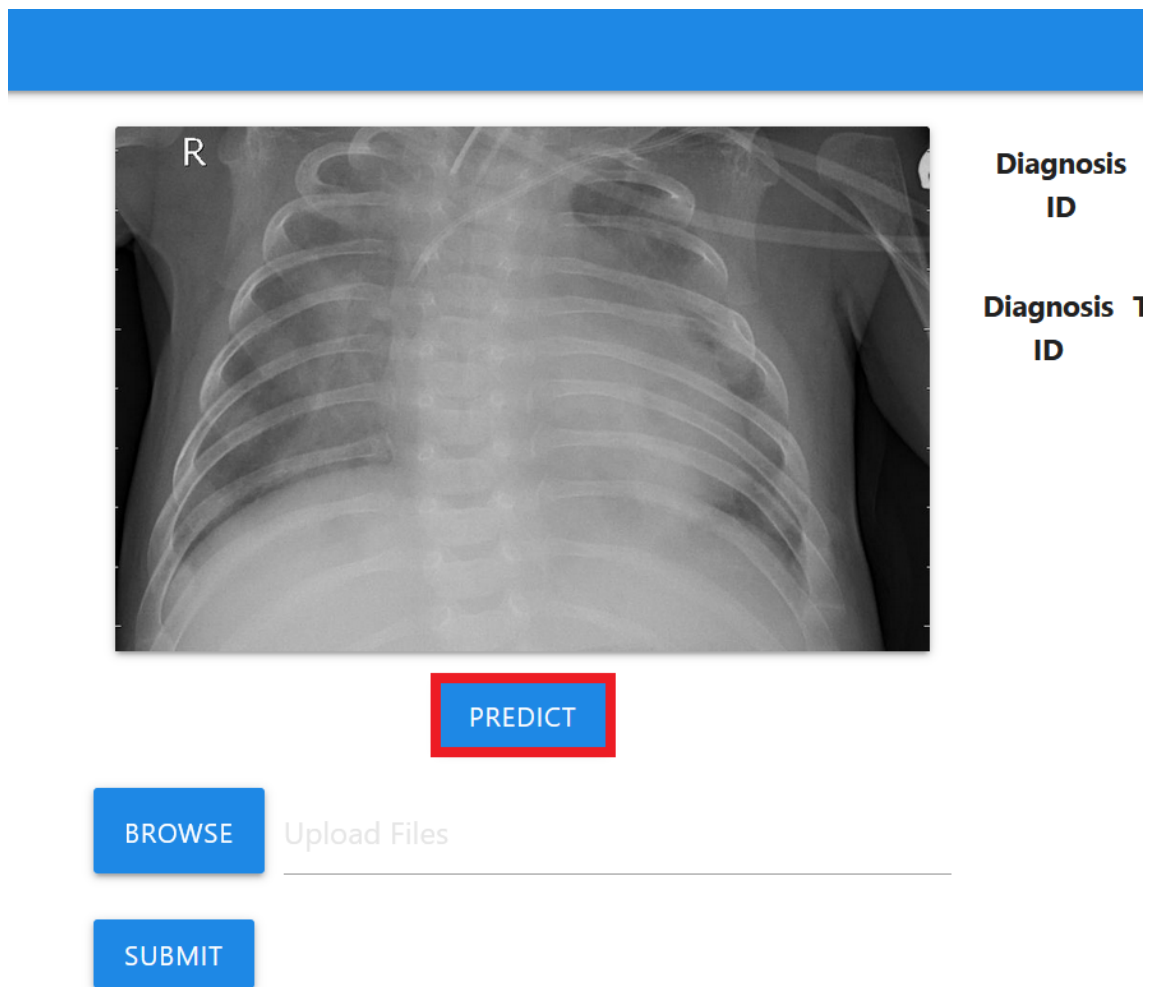


Figure 2.13: Predict button click

### 2.4.5 Removing diagnoses

To delete a diagnosis, navigate to the patient's row and select the patient. On the 'Edit Patient' page, you can remove diagnoses by clicking the red button located on the respective diagnosis's row

Test Country      Test City      TEST123      Test Street

UPDATE

ADD DIAGNOSES      ADD TREATMENT

Diagnosis ID	Patient ID	Name	TAJ #	Results	Pictures	Diagnosis ID	Treatment ID	Treatment	Issued By	Created At	Remove
49	4	Test Test Test	1234567899	P0.98	person1_virus_9.jpeg	49	2	Test	Test	2023-05-09 05:29:39	

Figure 2.14: Removing treatment and diagnoses

### 2.4.6 Adding treatments

To add a treatment, click the 'Add Treatment' button and fill out the necessary fields in the modal box. You can select which diagnoses you want to add the treatment to using the blue arrows next to each diagnosis. You can add multiple treatments to each diagnosis. Alternatively, you can do the same on the 'Edit Patient' page.

Home Predict P

PNEUMONIA Probability: 98.12 %

PREDICT

BROWSE Upload Files

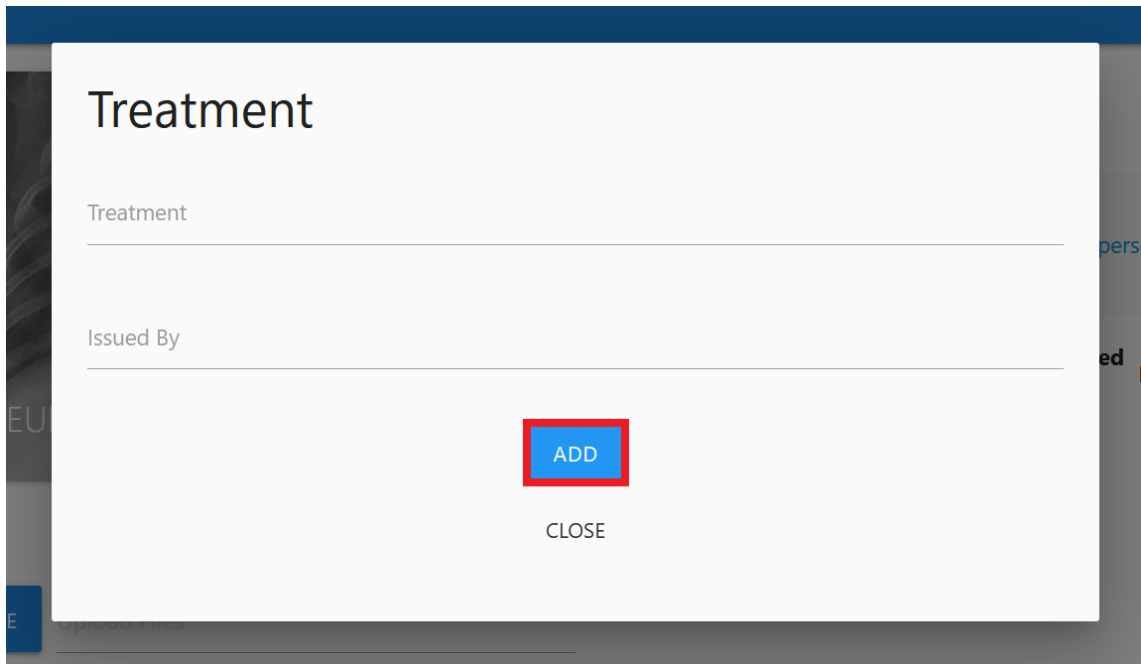
SUBMIT

Diagnosis ID	Patient ID	Name	TAJ #	Result	Picture	Select
49	4	Test Test Test	1234567899	P0.98	person1_virus_9.jpeg	

Diagnosis ID	Treatment ID	Treatment	Issued By	Created At	Remove

ADD TREATMENT

Figure 2.15: Adding treatment



Treatment

Treatment

Issued By

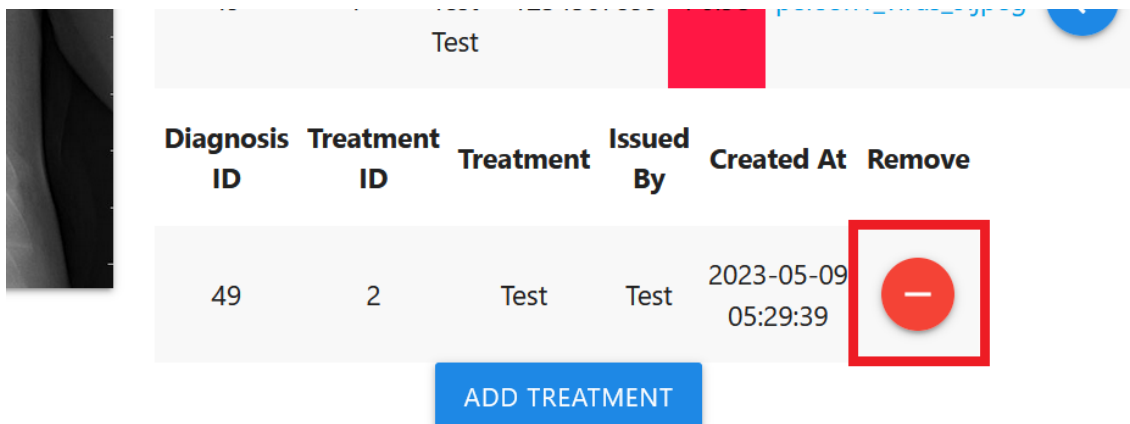
ADD


CLOSE

Figure 2.16: Adding treatment modal box

### 2.4.7 Removing treatments

To delete a recently added treatment, click the red button located on the respective treatment's row. Alternatively, navigate to the 'Patients' page, select a patient, and remove treatments by clicking the red button located on their respective rows.



Diagnosis ID	Treatment ID	Treatment	Issued By	Created At	Remove
49	2	Test	Test	2023-05-09 05:29:39	

ADD TREATMENT

Figure 2.17: Removing treatments



## 2. User documentation

Test Country

Test City

TEST123

Test Street

UPDATE

ADD DIAGNOSES

ADD TREATMENT

Diagnosis ID	Patient ID	Name	TAJ #	Results	Pictures	Diagnosis ID	Treatment ID	Treatment	Issued By	Created At	Remove
49	4	Test Test Test	1234567899	P0.98	person1_virus_9.jpeg	49	2	Test	Test	2023-05-09 05:29:39	

Figure 2.18: Removing treatment and diagnoses

### 2.4.8 Routes of the website

Route list	
Route	Description
/home	On the website homepage, you'll find clickable cards that allow for easy navigation.
/patients	On this page, you have the ability to edit existing patient data by clicking on it, or to add new patients using the appropriate option.
/choose	This website serves as a precursor to the predict function, and allows you to select a patient for diagnosis.
/predict/<int:id>	Once you've selected a patient, you'll be directed to this page where you can make a diagnosis or add treatments to existing ones.
/editPatient	On this page you are able to edit patient data, and manage treatments and diagnoses.

Table 2.1: Route table for user documentation.

# Chapter 3

## Developer documentation

### 3.1 Description of the problem

Pneumonia is a respiratory infection that affects the lungs, causing them to fill with pus and fluid which makes breathing painful and limits oxygen intake. A chest X-ray is often used to diagnose pneumonia, but test results can take several hours to a few days to confirm. In some cases, obtaining detailed information about test results may take even longer if multiple pneumonia tests are required. Early diagnosis and treatment of pneumonia is crucial to improve outcomes and save lives. Starting antibiotic therapy promptly can help to reduce the progression of the infection and prevent complications.

To address the challenge of delayed diagnosis, this application has been developed to quickly analyze X-ray images of children's chests and provide a diagnosis within seconds. The solution involved training a convolutional neural network model using two different validation datasets and creating a user friendly web interface and demo database.

The model was trained using Python3[5] using JupyterLab[13] and Google Colab. The dataset was sourced from Mendeley named "*Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification*"[2]. The teaching methodology can be found in the '*CNN\_Model.py*' file located in the root directory of the application.

## 3.2 Folder hierarchy

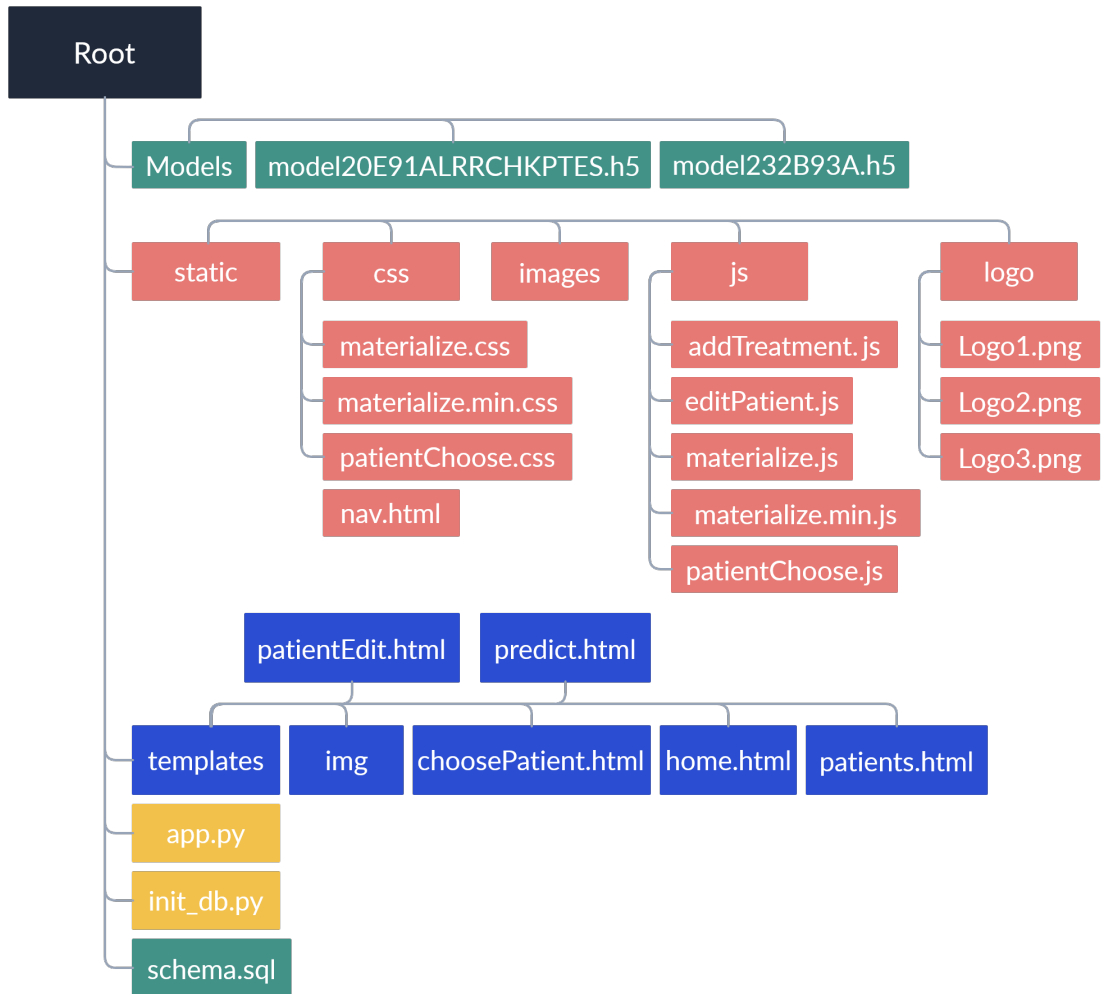


Figure 3.1: Folder structure of the program.

## 3.3 Convolutional Neural Network

### 3.3.1 Libraries

- `'import numpy as np'[10]` - Imports the NumPy library and assigns it the alias `'np'` to allow for easy access to its functions.
- `'import pandas as pd'[14]` - Imports the Pandas library and assigns it the alias `'pd'` to allow for easy access to its functions.
- `'import os'[5]` - Imports the built-in OS module, which provides a way to interact with the file system.

- `'import cv2'[9]` - Imports the OpenCV library, which is used for image processing and computer vision.
- `'import matplotlib.pyplot as plt'[15]` - Imports the PyPlot module from the Matplotlib library, which provides a way to create visualizations and graphs.
- `'import seaborn as sns'[16]` - Imports the Seaborn library, which is a visualization library based on Matplotlib.
- `'import keras'[11]` - Imports the Keras deep learning library, which provides a high-level interface for building neural networks.
- `'from keras.models import Model, Sequential'[11]` - Imports the Sequential class from the Keras models module, which is used to create a linear stack of layers in a neural network. It also imports the Model class which is used to create more complex neural network architectures.
- `'from keras.layers import Input, Dense, Conv2D, MaxPool2D, MaxPooling2D, Flatten, Dropout, BatchNormalization'[11]` - Imports several types of layers from the Keras layers module, including input layers, convolutional layers, pooling layers, and normalization layers.
- `'from keras.preprocessing.image import ImageDataGenerator'[11]` - Imports the ImageDataGenerator class from the Keras preprocessing module, which provides a way to generate batches of image data for training a neural network.
- `'from keras.callbacks import ModelCheckpoint, Callback, ReduceLROnPlateau'[11]` - Imports several callback functions from Keras, including ModelCheckpoint (to save the best model during training), Callback (a base class for creating custom callbacks), ReduceLROnPlateau used to reduce the learning rate when a metric has stopped improving.
- `'from keras.optimizers import Adam, SGD, RMSprop'[11]` - Imports several types of optimizers from the Keras optimizers module, which are used to adjust the weights of a neural network during training.

- `'from sklearn.model_selection import train_test_split'[17]` - Imports the `train_test_split` function from the Scikit-Learn library, which is used to split a dataset into training and testing subsets.
- `'from sklearn.metrics import classification_report, confusion_matrix'[17]` - Imports the `classification_report` and `confusion_matrix` functions from Scikit-Learn, which are used to evaluate the performance of a classification model.

These code snippets are used to import various libraries and functions that are needed to build and train neural networks for image classification tasks. The libraries include NumPy, Pandas, Matplotlib, Seaborn, Keras, Scikit-Learn, and OpenCV. The functions and classes are used to define the layers and architecture of the neural network, as well as to set up callbacks and optimization algorithms to improve the model's performance.

#### 3.3.2 Global variables

The file contains 5 global variables that are used in the program:

- `EPOCHS` (integer): The number of epochs used during the model training process. In this case it is 20.
- `INIT_LR` (float): The initial learning rate used in the optimizer. It is also used for learning rate reduction during the training process.
- `IMG_SIZE` (int, int): Integer tuple that stores the image sizes for resizing and model training. In this case it is set to (224, 224).
- `TITLES` [string]: An array consisting of two strings, 'PNEUMONIA' and 'NORMAL', which are used as the classes for the pictures.
- `PATH` (string): The path to the root folder of training, testing and validation data.

#### 3.3.3 Data

The dataset was sourced from mendeley "Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification"[2]. These were

screened for quality control by removing unreadable images and then split into training and testing datasets.

The images were previously evaluated by three expert physicians who labeled the diagnoses. The X-ray photos are only of children between the ages of 1 and 5 because of this the Model will only be able to identify children's lung X-rays.

#### **Data preparation**

The dataset was structured into two main folders, namely "train" and "test". Each folder contains two sub-folders named "PNEUMONIA" and "NORMAL", which contain the corresponding chest x-ray images. Additionally, a new folder named "val" was created to hold validation images. To create this validation set, 200 images were manually selected from the "train" folder.

#### **Data normalization**

Normalization of input data is a crucial preprocessing step for training convolutional neural network (CNN) models. Normalization ensures that input data is scaled and centered to the same range, making it easier for the CNN to learn the underlying patterns. Normalizing the data helps the CNN to converge faster during training, improves generalization to new unseen data, and reduces overfitting. Overfitting can occur when the network becomes too specialized to the training data, leading to poor performance on new unseen data. By normalizing the data, the network becomes less sensitive to outliers and noise, and more robust to variations in input data.

To preprocess the image data for training the model, the following normalization methods were used.

First, the `'get_data()'` function reads the images and converts them to grayscale. Then, each image is resized to the value of the global `'IMG_SIZE'` variable.

Next, the image data is converted into a NumPy array using the `'np.array()'` function. This array is then normalized by dividing each value by 255, which scales the pixel values to be between 0 and 1.

After normalization, the NumPy array is reshaped to the shape `(-1, IMG_SIZE, IMG_SIZE, 1)` using the `'reshape()'` function in NumPy. The first value of `-1` automatically determines the batch size of the data, based on the total number of

elements in the array. *'IMG\_SIZE'* is the global variable representing the size of the normalized image, and the last 1 refers to the amount of color channels in this case it means grayscale.

By performing these normalization steps, the input image data is scaled and centered to a standard range, making it easier for the neural network to learn the underlying patterns. The resulting normalized and reshaped NumPy array can be fed into the CNN's input layer for training and prediction.

#### Data augmentation

The initial dataset has a great imbalance of normal and pneumonia photos, because of this the *'ImageDataGenerator'* was used from the *'keras.preprocessing.image'* library to artificially create more data. For this the following parameters were set:

- *featurewise\_center* (boolean) -> False: sets input mean to 0 over the dataset.
- *samplewise\_center* (boolean) -> False: sets each sample mean to 0.
- *featurewise\_std\_normalization* (boolean) -> False: divides inputs by st of the dataset.
- *samplewise\_std\_normalization* (boolean) -> False: performs feature-wise standardization on the input data during model training.
- *zca\_whitening* (boolean) -> False: applies ZCA whitening transformation.
- *rotation\_range* (int) -> 20: range for random rotations.
- *zoom\_range* (float) -> 0.2: range for random zooms.
- *width\_shift\_range* (float) -> 0.2: range for horizontal random shifts.
- *height\_shift\_range* (float) -> 0.2: range for vertical random shifts.
- *horizontal\_flip* (boolean) -> True: randomly flip inputs horizontally.
- *vertical\_flip* (boolean) -> False: randomly flip inputs vertically.
- *fill\_mode* (string) -> nearest: points outside the boundaries of the input are filled according to the given mode.

These settings specify the range of possible modifications that can be applied to each image during training. To clarify, the training data was passed through the 'ImageDataGenerator' to create augmented images, and then the model was trained using these augmented images.

### **3.3.4 Convolutional neural network model**

The convolutional neural network (CNN) model has been trained to predict the presence of pneumonia in chest X-ray images. It takes in gray-scale images with a resolution of 224 by 224 pixels as input and outputs a binary classification result indicating whether the input image is normal or shows evidence of pneumonia.



## Architecture

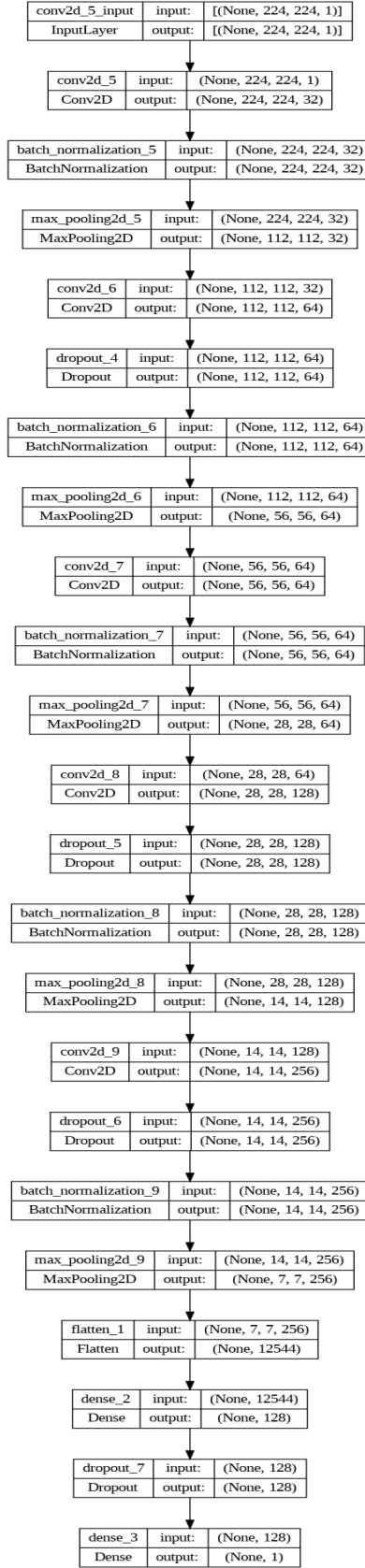


Figure 3.2: Architecture of the Convolutional neural network model' photos

The model architecture consists of multiple convolutional, max pooling, and batch normalization layers. These layers are used to extract features from the input image, reduce its spatial dimensions, and normalize.

The first layer in the model is an input layer that takes in images with one color channel (gray-scale) of size 224 by 224 pixels. This is followed by a convolutional layer with 32 filters, each one is size 3 by 3. The output of this layer is then passed through a dropout layer with a rate of 0.2 to prevent overfitting.

The next few layers follow a similar pattern: convolutional layers with increasing numbers of filters, each followed by a dropout layer and a batch normalization layer and max pooling layers to reduce the spatial dimensions of feature maps produced by the convolutional layers.

After several layers of convolutions and pooling, the final layer is a fully connected dense layer with 128 neurons, followed by another dropout layer and a final dense layer with a single neuron, representing the binary classification output.

In total, the model has over 2 million trainable parameters, which were learned during the training process to accurately classify chest X-ray images as either normal or indicating the presence of pneumonia.

#### **Loss functions**

Loss functions were used for the training process to prevent overfitting. The first is the learning rate reduction function from Keras called *'ReduceLROnPlateau'* the following parameters are set for the function:

- monitor (string) -> val\_accuracy: The metric to monitor for changes.
- patience (int) -> 2: The number of epochs with no improvement after which the learning rate will be reduced.
- verbose (int) -> 1: The amount of information to be displayed during training. In this case, it will print a message when the learning rate is reduced.
- factor (float) -> 0.3: The factor by which the learning rate will be reduced. It will be multiplied by the current learning rate.
- min\_lr=INIT\_LR (float) -> INIT\_LR: The minimum learning rate allowed. Once the learning rate reaches this minimum value, it will not be reduced further. It is set by the INIT\_LR global variable.

The second loss function is called '*ModelCheckpoint*' from the '*keras.callbacks*' library which is set up to save the best weights during the training process so they can be loaded back later.

The '*model.fit*' function is used to train the machine learning model, it is called with four arguments:

- The first argument specifies the training data. It is sent through the ImageDataGenerator that was previously set up.
- The second argument is epochs which is set to be as the global EPOCHS variable in this case 20.
- The third argument is the validation data, it is also passed through the ImageDataGenerator that was previously set up.
- The final argument is callbacks, here the previously described learning rate reduction and checkpoint is used.

#### 3.3.5 Functions

- '*get\_data*' - Takes the image data directory as an argument and has one variable which is also the return variable called data. The images are stored in two folders named PNEUMONIA and NORMAL. A for loop goes through the TITLES array and then another for loop uses the scandir function from the os library to find and go through the images. This is followed by using the imread function of the OpenCV library to read and convert the images into grayscale then the data is stored in the return array. Before returning the values, they are first converted into a NumPy array using the array function of NumPy.
- '*convolutional\_block*' - Creates a basic block for the convolutional layers. The function takes the previous layer, number of filters, kernel size, stride and dropout rate as arguments. The layer applies convolutional filters, dropout regularization, batch normalization, and max-pooling to the input data and returns the output
- '*fully\_connected\_block*' - Creates a basic block for the fully connected layer of the model. The function takes the previous layer, number of output units, and the dropout rate as arguments. It applies dense layers with rectified linear

unit (ReLU) activation and dropout regularization to the input and returns the output.

- *'create\_model'* - Defines the entire neural network model. The function takes the input image shape as argument and defines the layers accordingly. It stacks several convolutional blocks followed by a fully connected block and an output layer. The following layer consists of a single unit with sigmoid activation, which makes it suitable for the binary classification task of predicting pneumonia.
- *'compile\_model'* - Compiles the model by specifying the optimizer, loss function. and metrics to be used during training.
- *'print\_summary'* - Prints a summary of the model.

#### 3.3.6 Graphs

##### Example data after preprocessing

These images show examples of the training data after preprocessing, which involved resizing them to 224 by 224 pixels and converting them to grayscale. These steps were necessary to ensure that the input data was in a consistent format for the model. Prior to normalization, the images may have had different sizes and color channels, which could have affected the accuracy and reliability of the model's predictions.

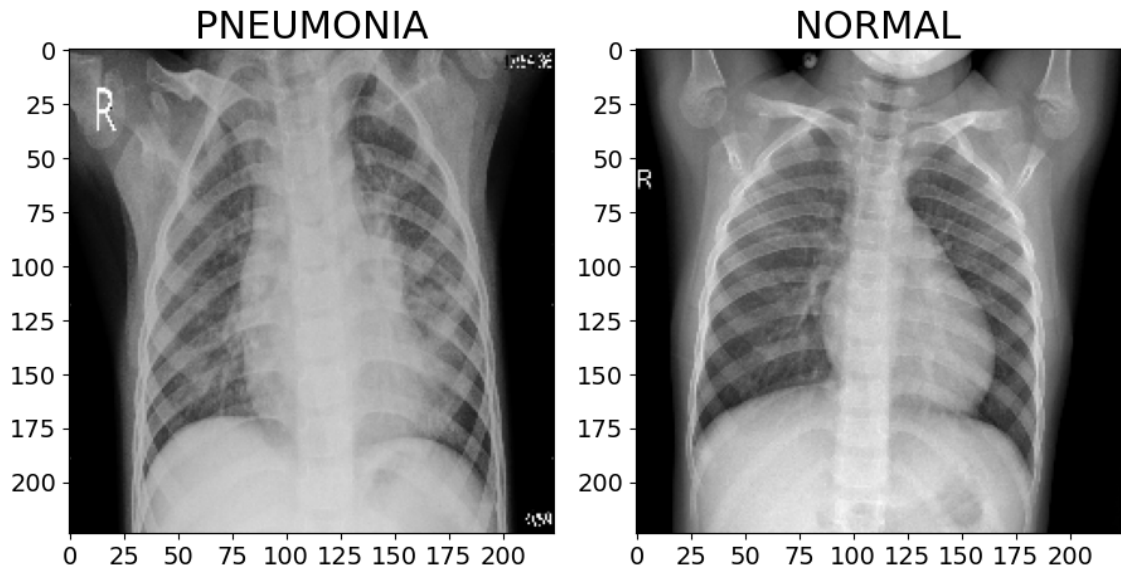


Figure 3.3: Example data after normalization

### Class imbalance

These two graphs display important information about the dataset used for training the model. The first graph shows the distribution of available data, indicating the number of images in each class. The second graph highlights the class imbalance between pneumonia and normal photos, which can affect the performance of the model. It is important to consider these factors when training and evaluating the model to ensure that it can accurately classify new images.

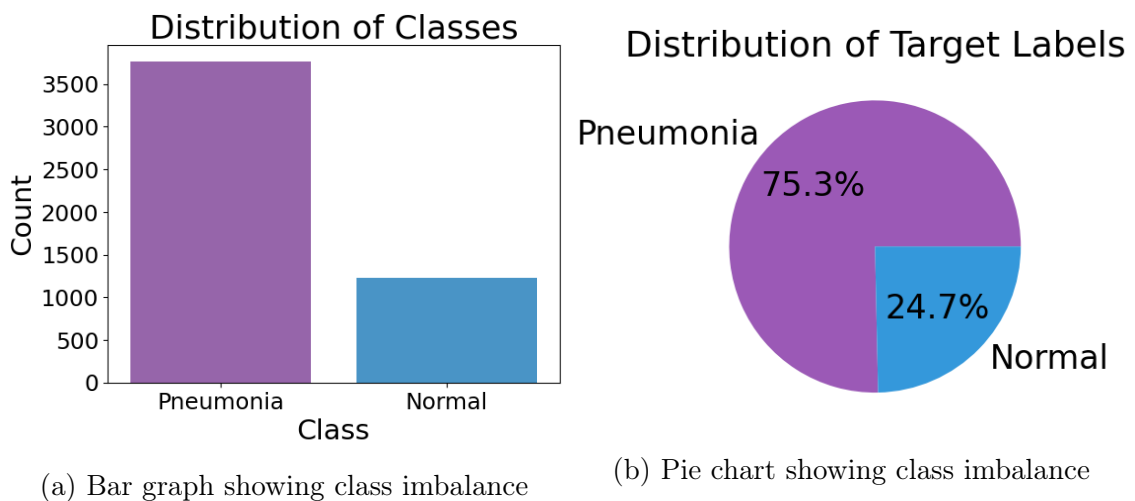


Figure 3.4: Class imbalance

## Learning curve

This graph displays the training and validation accuracies and losses during the 20 epochs of model training. The minimal difference between the accuracies and losses occurred in the 17th epoch, where the ModelCheckpoint function saved the weights of the model.

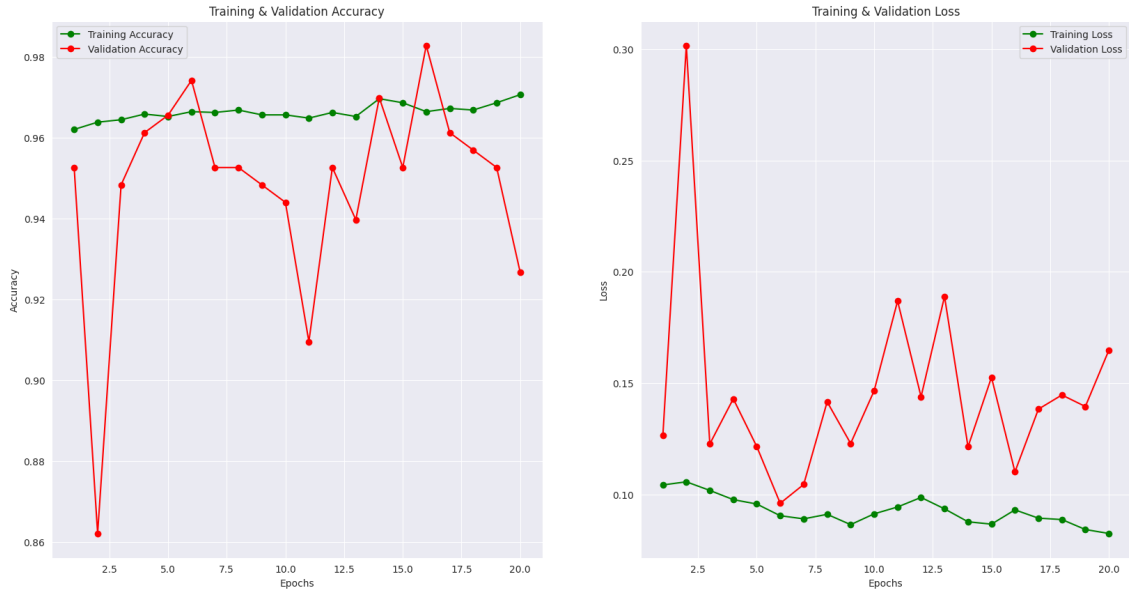


Figure 3.5: Training curve of the model

## Confusion matrix

This confusion matrix was used to evaluate the performance of the trained model. The matrix has two rows and two columns, representing the predicted and actual classes of the test dataset. Out of a total of 624 test samples, 366 were correctly classified as pneumonia (True Positives), 204 were correctly classified as normal (True Negatives), while 24 samples were incorrectly classified as normal when they were actually pneumonia (False Negatives), and 30 samples were incorrectly classified as pneumonia when they were actually normal (False Positives)

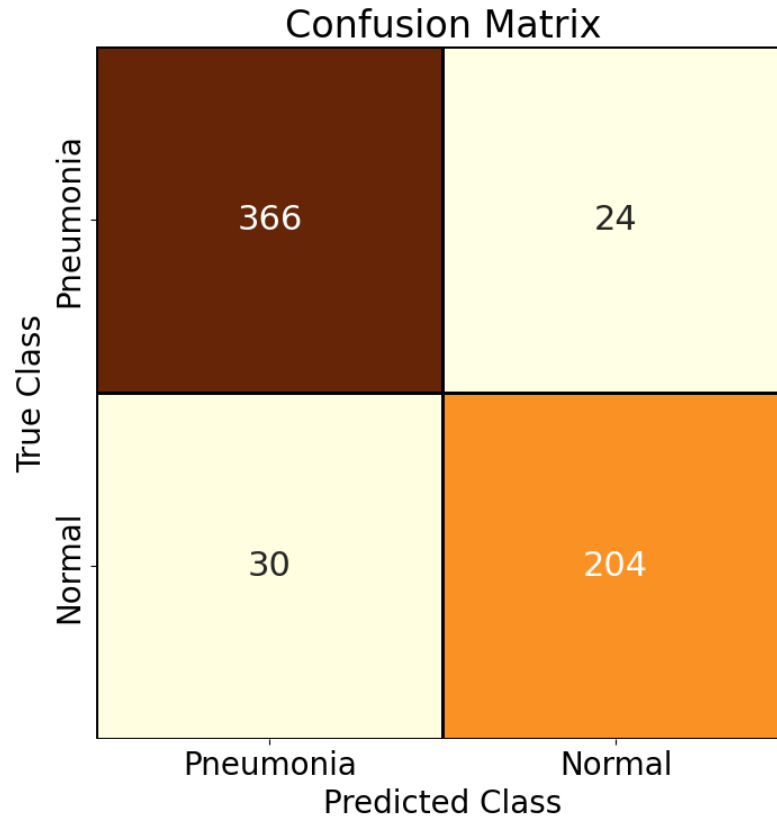


Figure 3.6: Confusion Matrix

## 3.4 Web application

The web application was developed using the *'Flask'*[6] web application framework and utilizes the Materialize CSS framework, as well as jQuery. All the required files are located in the *'Web'* folder in the root directory of the application. The *'app.py'* file contains the code needed to serve the web server and handles all the routes and functions of the application. The *'schema.sql'* file includes a demo database. Specific details about the database are listed later in the documentation. To initialize the database and add a test patient, run the *'init\_db.py'* file.

### 3.4.1 Libraries

- *'from fileinput import filename'*[5] - This library provides an easy way to iterate over multiple inputs, such as files.
- *'from flask import Flask, render\_template, request, redirect'*[6] - Flask is a lightweight web application framework for Python. It provides tools, libraries,

and patterns that allows for an easy and quick website development. The `render_template` function from flask allows for rendering templates with specific data. The `requests` object provides access to incoming requests, such as form data. The `redirect` function allows for easy redirection to URLs.

- `'import sqlite3'`[7] - SQLite3 is a built in module of Python, that provides a lightweight disk-based database, which requires no additional server processes and allows for database access through SQL.
- `'from werkzeug.utils import secure_filename'`[8] - Werkzeug provides various utilities for building Python web applications, such as secure file uploads and URL routing.
- `'import os'`[5] - OS is a built in Python module that provides a way of using operating system dependent functions, like reading or writing to the file system.
- `'import cv2'`[9] - cv2 is the Python wrapper for the OpenCV library, which is an open-source computer vision and machine learning software library.
- `'import numpy as np'`[10] - NumPy adds support for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions.
- `'from keras.models import load_model'`[11] - Keras is an open-source software library that provides a Python interface for TensorFlow[18].
- `'from waitress import serve'`[12] - Waitress is a pure-Python web server for serving WSGI applications.

#### 3.4.2 Global variables

The file includes 6 global variables:

- `'app'` - This variable creates a new instance of the Flask class, it is used throughout the application to define routes and handle requests.
- `'app.config['IMAGE_UPLOADS']'` - This variable sets a configuration option for the Flask application. It specifies a permanent location for the application to save uploaded images.



- *'model1'* - This variable is used to load the first prediction model used by the predict function.
- *'model2'* - This variable loads a secondary prediction model used as a backup or alternative.
- *'filename'* - This variable is used to store the name of uploaded files.
- *'selectedDiagnosis'* - This variable is used to store the user's currently selected diagnosis on the site.

### 3.4.3 Application functions

#### loadmodel()

The loadmodel() function uses the global variables called model1, and model2, to load two pre-trained modules using the *'load\_model()'* function from Keras.

#### prediction()

This function requires the filename global variable to not be null, that means at the run time of the function a correct file must have been uploaded. The function then creates a dictionary called *'classes'* with keys 0 and 1, representing *'Pneumonia'* and *'Normal'*

Using the OpenCV library's *'cv2.imread()'* function, the image is read in grayscale mode, converted into a NumPy array and normalized. This can be seen in the code below.

```
1  # Load the image and resize it
2  image = cv2.imread(os.path.join(app.config['IMAGE_UPLOADS'],
3                                filename), cv2.IMREAD_GRAYSCALE)
4
5  # Convert the resized image to a NumPy array and normalize it
6  image_array = np.array(image)
7  image_array = image_array[np.newaxis, ..., np.newaxis]
8  image_array = image_array / 255.0
```

Next, the function uses the *'predict'* function of the loaded *'model1'* to generate a prediction. If this prediction is between the values of 0.4 and 0.6, indicating uncertainty, the function passes the image to the second model, and takes the average

of the two predictions as a final prediction. Finally, the function returns a string containing the diagnosis and the probability of the prediction.

#### `get_db_connection()`

This function uses the `'sqlite3.connect'` function to establish a connection to the specified database. (*Note:* The database must have been created beforehand using the `'init_db.py'` file.) The function then returns the established connection.

#### `get_data()`

This function executes the SQL code shown below to retrieve specific data from the database. It is primarily used to display data on tables within some of the webpages.

```
1  SELECT pd.id, taj, first_name, middle_name, last_name, pdiag.
   result, pdiag.image_name, pdiag.id as did
2  FROM Patient_Data as pd
3  JOIN Patient_Diagnosis as pdiag on pd.id = pdiag.patient_id
4  WHERE pd.id = {record_id}'
```

The function requires a connection object and a patient ID to function properly. The SQL code uses a join between two tables, `'Patient_Data'` and `"Patient_Diagnosis"`, to retrieve data about a specific patient, including their ID, Social Security number (taj), name, diagnosis result, and the name of the image associated with their diagnosis. The function then returns the result of the executed query.

#### `get_patient_data()`

This function executes the SQL query shown below to retrieve patient data for specific webpages.

```
1  SELECT pd.id, first_name, last_name, middle_name, taj, birthday
   , country, city, post_code, street, phone, email,
   emergency_contact_email, emergency_contact_phone,
   emergency_contact_first_name, emergency_contact_middle_name,
   emergency_contact_last_name
2  FROM Patient_Data AS pd
3  INNER JOIN Patient_Address AS pa on pd.id = pa.patient_id
4  INNER JOIN Patient_Contact AS pc on pd.id = pc.patient_id
5  WHERE pd.id = {patient_id}
```

The function requires a connection object and a patient ID to be passed as arguments. The SQL code uses multiple joins between tables *'Patient\_Data'*, *'Patient\_Address'*, and *'Patient\_Contact'* to retrieve various pieces of information about a specific patient, including their ID, name, Social Security number (taj), date of birth, address, phone number, email address and emergency contact info. After executing the query, the function returns the retrieved patient data.

### **get\_treatment\_data()**

This function requires a connection object, a patient ID, and a selected diagnosis ID as input parameters. If the *'selectedDiagnosis'* parameter is not specified or its value is *'None'*, then the function queries the database to find the most recent diagnosis ID associated with the specified patient ID. The SQL code used to execute the query can be seen below. (*Note:* The *'fetchone()'* function is used from the *'SQLite3'* library to select only the first result.

```
1  SELECT id
2  FROM Patient_Diagnosis
3  WHERE patient_id = {record_id}
4  ORDER BY id DESC"
```

Once a diagnosis ID is specified, the function executes the following SQL code to retrieve treatment data associated with the diagnosis ID from the *'Patient\_Treatment'* table in the database.

```
1  SELECT *
2  FROM Patient_Treatment
3  WHERE diagnosis_id = {diagnosis_id}
```

If a diagnosis ID is found the function returns the retrieved treatment information. Otherwise, it returns *'None'*.

### **insert\_patient()**

This function is used to insert new records into the *'Patient\_Data'*, *'Patient\_Address'* and *'Patient\_Contact'* tables. It requires a connection object and task array as input parameters. The following SQL code is used in the function.

```
1  INSERT INTO Patient_Data (birthday, taj, first_name,
    middle_name, last_name)
```

```
2  VALUES (?, ?, ?, ?, ?)
3
4  INSERT INTO Patient_Contact (patient_id, phone, email)
5  VALUES (?, ?, ?)
6
7  INSERT INTO Patient_Address (patient_id, country, post_code,
8  city, street)
  VALUES (?, ?, ?, ?, ?)
```

The values to create the record are specified in the *'execute'* function as strings in the *'task'* array. The function has no return value as it's purpose is only to insert new data into the database.

### **insert\_diagnoses()**

This function is used to insert new records into the *'Patient\_Diagnosis'* table. It requires a connection object and a task array as input parameters. The *'task'* array contains the values that need to be inserted into the table. The function uses the following SQL code.

```
1  INSERT INTO Patient_Diagnosis (result, image_name, patient_id)
2  VALUES (?, ?, ?)
```

The values to create the record are specified in the *'execute'* function as strings in the *'task'* array. The function has no return value as it's purpose is only to insert new data into the database.

### **insert\_treatment()**

This function requires a connection object, task array, patient ID, and a selected diagnosis ID as arguments. The function inserts treatment data into the *'Patient\_Treatment'* table with the values specified in the task array. The function first checks if the selected diagnosis is specified. If it is not, it queries the most recent diagnosis ID for the selected patient using the following SQL code. (*Note:* The *'fetchone()'* function is used from the *'SQLite3'* library to select only the first result.

```
1  SELECT id
2  FROM Patient_Diagnosis
3  WHERE patient_id = {record_id}
```

```
4 ORDER BY id DESC"
```

If a selected diagnosis id was specified, then the function executes an SQL insert statement to insert the treatment data specified in the task array into the *'Patient\_Treatment'* table.

```
1 INSERT INTO Patient_Treatment (treatment, issued_by,  
2 diagnosis_id)  
VALUES (?, ?, ?)
```

There is no return value to this function as it's purpose is solely to insert new records into the database.

### **update\_patient\_data()**

This function is used to update data in the *'Patient\_Data'*, *'Patient\_Contact'* and *'Patient\_Address'* tables. It requires a connection object, a task array, and a patient's ID as input parameters. The *'task'* array contains the values that are to be updated into the tables. The function uses the following SQL code to execute the updates.

```
1 UPDATE Patient_Data  
2 SET birthday = '{task[10]}' , taj = '{task[9]}' , first_name = '{  
task[0]}' , middle_name = '{task[1]}' , last_name = '{task[2]}'  
WHERE id = {record_id}  
3  
4 UPDATE Patient_Contact  
5 SET phone = '{task[4]}' , email = '{task[3]}' WHERE patient_id =  
{record_id}  
6  
7 UPDATE Patient_Contact  
8 SET emergency_contact_phone = '{task[12]}' ,  
emergency_contact_email = '{task[11]}' ,  
emergency_contact_first_name = '{task[13]}' ,  
emergency_contact_middle_name = '{task[14]}' ,  
emergency_contact_last_name = '{task[15]}' WHERE patient_id  
= {record_id}  
9  
10 UPDATE Patient_Address  
11 SET country = '{task[5]}' , post_code = '{task[7]}' , city = '{task  
[6]}' , street = '{task[8]}' WHERE patient_id = '{record_id}'
```

The function does not have a return value as it's purpose is only to update specific patient's information.

#### **remove\_treatment()**

This function takes a database connection object and a treatment's ID as parameters. The function executes the following SQL query to delete the row of the given treatment.

```
1      DELETE FROM Patient_Treatment
2      WHERE id = {treatment_id}
```

There is no return value for this function.

#### **remove\_diagnoses()**

This function takes a database connection object and a diagnosis ID as arguments. It first queries all the treatments associated with the diagnosis using the following SQL code.

```
1      SELECT id
2      FROM Patient_Treatment
3      WHERE diagnosis_id = {diagnosis_id}
```

If it finds any treatments that are associated with the diagnosis ID, it iterates over the list and calls the *'remove\_treatment()'* function for each treatment. After this, the function runs the following SQL code to remove the diagnosis record.

```
1      DELETE FROM Patient_Diagnosis
2      WHERE id = {diagnosis_id}
```

There is no return value to the function.

#### **remove\_patient()**

This function takes a connection object and a patient's ID as input parameters. It first fetches all the diagnoses related to the selected patient. If there are diagnoses found, then for each diagnosis the *'remove\_diagnoses()'* function is used to remove the specific diagnosis and it's related data. After removing the diagnoses associated with the patient the function executes the following SQL code to remove all patient information.

```
1  DELETE FROM Patient_Address
2  WHERE patient_id = {patient_id}
3
4  DELETE FROM Patient_Contact
5  WHERE patient_id = {patient_id}
6
7  DELETE FROM Patient_Data
8  WHERE id = {patient_id}
```

There is no return value in this function.

### 3.4.4 Route functions

#### **home()**

This function renders the template for the *'home.html'* file using the *'render\_template'* function from the *'Flask'* library.

#### **patients()**

This function uses the *'get\_db\_connection()'* function to get a database connection. It has a POST request method that checks for POST requests. If the *'AddNew'* button on the page is pressed, it requests all the form information from the HTML form and uses the *'insert\_patient()'* function to insert a new patient into the database. At the end of the function, the *'render\_template()'* function is used from the Flask library with the following parameters: *'patients.html'*, *'data=data'*. *'data'* refers to the table data presented on the webpage. For this, the following SQL query is used:

```
1  SELECT pd.id, first_name, last_name, middle_name, taj,
      birthday, pa.country, pa.city
2  FROM Patient_Data AS pd
3  INNER JOIN Patient_Address AS pa on pd.id = pa.patient_id
```

#### **choose()**

This function uses the *'get\_db\_connection()'* function to get a database connection. It then executes an SQL query to fetch all patient data, then stores it in the data variable similarly to the *patients()*'s SQL query. Finally, the function

returns a rendered HTML template called *'choosePatient.html'* with the *'data'* variable passed as a parameter. This HTML template will be displayed on the webpage and will contain a table with patient information.

#### **predict()**

This function uses the global *'filename'* and *'selectedDiagnosis'* variables, as well as *'get\_db\_connection()'* function to get a database connection, the *'get\_data()'* function, and the *'get\_treatment\_data()'* function to get information from the database to the website. It has a POST request method that checks for POST requests coming from the server.

- If the Submit button is pressed, the function requests a file to be uploaded. If a file is successfully uploaded, it saves its filename into the global variable *filename* and saves the picture into the specified image upload folder. After this, the data variable is queried again to refresh the table present on the website.
- If the Predict button is pressed, the function uses the *'prediction()'* function to get a diagnosis from the picture. It then uses the *'insert\_diagnosis()'* function to insert the diagnosis and image information into the database. Then the *'data'* variable is queried again to refresh the table.
- If the Add button is pressed in the modal form on the website, the function first requests the given information from the HTML form and then uses the *'insert\_treatment()'* function passing the database connection object and the data received from the form. After this, the treatment data is queried again to refresh the table.
- If a treatment is being removed the function calls the *remove\_treatment()'* function passing in the database connection object and the ID of the treatment record to be removed. After removing the treatment, the function retrieves the updated treatment data using the *'get\_treatment\_data()'* function.
- If the select diagnosis button is pressed, then the function first retrieves the selected diagnosis from the HTML form using the *'request.form.get()'* function from Flask. It then uses the *'get\_treatment\_data()'* function to query



the database and retrieve the corresponding treatment data for the selected diagnosis and patient record.

After the POST call is over if the treatment data variable is not None then the function returns the *'predict.html'* template with the *'filename'*, *'data'*, *'treatment\_data'* and *'record\_id'* variables passed in as arguments. If the *'treatment\_data'* variable is of value None then it is excluded from the return statement.

#### **edit\_patient()**

This function uses the global *'selectedDiagnosis'* variable as well as the *'get\_db\_connection()'*, *'get\_patient\_data()'*, *'get\_treatment\_data()'* and *'get\_data()'* functions to get information from the database to the website. It has a POST request method that checks for POST requests coming from the server.

- If the Add button is pressed in the treatment modal form, the function first requests the given information from the HTML form. It then uses the *'insert\_treatment()'* function, passing the database connection object, treatment information, record ID, and *'selectedDiagnosis'* variables as arguments.
- If the Remove Diagnosis button is pressed, the function calls the *'remove\_diagnosis()'* function and passes the database connection object and the ID of the specified diagnosis. Afterwards, the *'get\_treatment\_data()'* and *'get\_data()'* functions are used to refresh the tables on the website.
- If the Remove Treatment button is pressed, the function calls the *'remove\_treatment()'* function, passing the connection object and the ID received from the POST call as arguments. Afterwards, the *'get\_treatment\_data()'* function is used to refresh the table on the website.
- If the Select Diagnosis button is pressed, the function updates the value of the global variable *'selectedDiagnosis'* with the ID of the currently selected diagnosis, which is obtained from the POST request. Then, the function calls the *'get\_treatment\_data()'* function to retrieve the relevant treatment data and refresh the table on the website.
- If the Update button is pressed, the function first retrieves all patient data from the HTML form. It then uses the *'update\_patient\_data()'* function, passing

all the received information and the connection object as arguments to update the patient's data in the database. Finally, the `'get_patient_data()'` function is called to refresh the text fields displaying the patient's data on the website.

After the POST call is over if the treatment data variable is not `'None'` then the function returns the `'patientEdit.html'` template with the `'data'`, `'treatment_data'`, `'patient_data'` and `'record_id'` variables passed in as arguments. If the `'treatment_data'` variable is of value `'None'` then it is excluded from the return statement.

## 3.5 Database

### 3.5.1 Introduction

The SQLite[7] database engine was used to create this database, which models a simplified hospital hierarchy. The database contains five tables that store patient data, contact information, addresses, diagnoses, and treatments.

### 3.5.2 Tables

**Table 1: (Patient\_Data)** The `'Patient_Data'` table contains basic patient information, including a primary key, birthday, Social Security number (TAJ), first name, last name, middle name, and timestamp for when the record was created.

**Table 2: (Patient\_Contact)** The `'Patient_Contact'` table holds the patient's contact information, including a primary key, a foreign key to the Patient\_Data table, email, phone, emergency contact's phone number, emergency contact's email address, emergency contact's first name, middle name, and last name.

**Table 3: (Patient\_Address)** The `'Patient_Address'` table contains patient address information, including a primary key, a foreign key to the Patient\_Data table, country, postal code, city, and street.

**Table 4: (Patient\_Diagnosis)** The `'Patient_Diagnosis'` table stores diagnostic data for patients, including a primary key, a foreign key to the Patient\_Data table, a result of diagnoses, and the name of uploaded images.

**Table 5: (Patient\_Treatment)** The 'Patient\_Treatment' table contains treatment information for diagnoses, including a primary key, a foreign key to the Patient\_Diagnosis table, treatment details, who issued the treatment, and a timestamp for when the treatment was created.

**Connections:** There is a one-to-one connection between the 'Patient\_Data' and 'Patient\_Contact' tables, and between the 'Patient\_Data' and 'Patient\_Address' tables. There is a one-to-many connection between the 'Patient\_Data' and 'Patient\_Diagnosis' tables, and between the 'Patient\_Diagnosis' and 'Patient\_Treatment' tables.

By using this demo database, we can easily store and retrieve patient data, contact information, addresses, diagnoses, and treatments which makes it ideal for testing purposes.

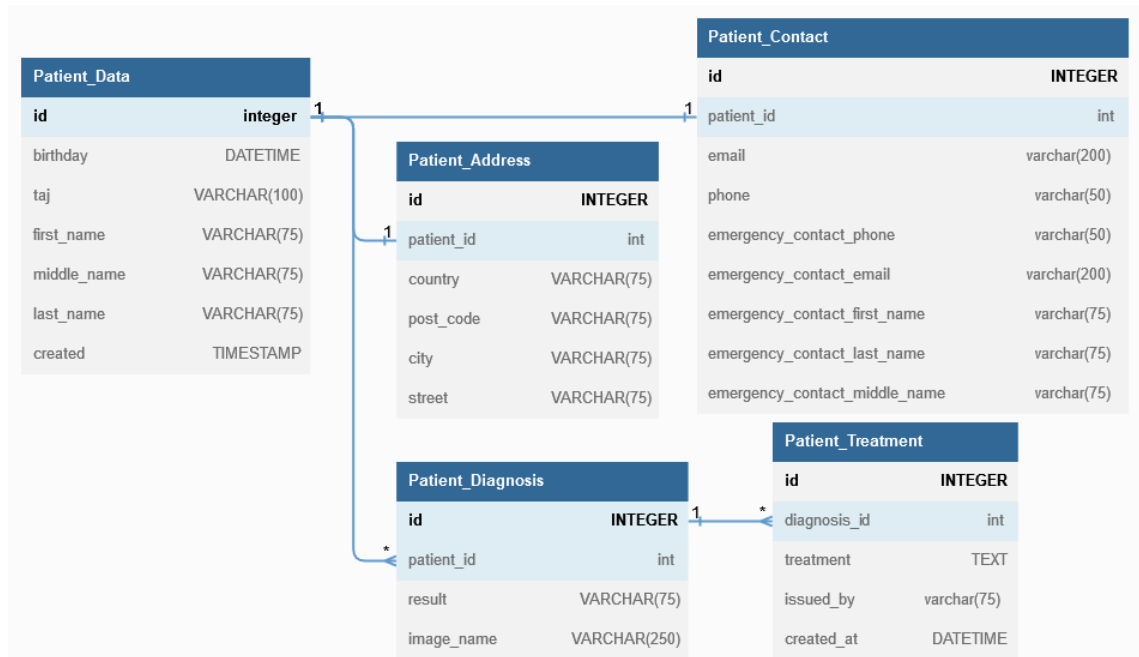


Figure 3.7: Database Relationship Diagram

### 3.5.3 Database table column descriptions

#### Patient Data

Patient_Data	
<i>Name</i>	<i>Description</i>
<i>id</i> <int>	Primary key of the table. It is set to auto-increment.
<i>birthday</i> <datetime>	Birthday registered to the patient. It cannot be null.
<i>TAJ</i> <varchar(100)>	Social Security number (TAJ) registered to the patient. It needs to be unique and cannot be null.
<i>first_name</i> <varchar(75)>	First name of the patient. It cannot be null.
<i>middle_name</i> <varchar(75)>	Middle name of the patient.
<i>last_name</i> <varchar(75)>	First name of the patient. It cannot be null.

Table 3.1: Database table column description for the Patient\_Data table.

## Patient Contact

Patient_Contact	
<i>Name</i> <datatype>	<i>Description</i>
<i>id</i> <int>	Primary key of the table. It is set to auto-increment.
<i>patient_id</i> <int>	Foreign key referencing the Patient_Data table's id.
<i>email</i> <varchar(200)>	Patient's email address.
<i>phone</i> <varchar(50)>	Patient's phone number.
<i>emergency_contact_phone</i> <varchar(50)>	Emergency contact's phone number.
<i>emergency_contact_email</i> <varchar(200)>	Emergency contact's email address.
<i>emergency_contact_first_name</i> <varchar(75)>	Emergency contact's first name.
<i>emergency_contact_middle_name</i> <varchar(75)>	Emergency contact's middle name.
<i>emergency_contact_last_name</i> <varchar(75)>	Emergency contact's last name.

<i>Name&lt;datatype&gt;</i>	<i>Description</i>
-----------------------------	--------------------

Table 3.2: Database table column description for the Patient\_Contact table.

### Patient Address

Patient_Address	
<i>Name&lt;datatype&gt;</i>	<i>Description</i>
<i>id&lt;int&gt;</i>	Primary key of the table. It is set to auto-increment.
<i>patient_id&lt;int&gt;</i>	Foreign key referencing the Patient_Data table's id.
<i>country&lt;varchar(75)&gt;</i>	Country the patient lives in.
<i>postal_code&lt;varchar(75)&gt;</i>	Postal Code of the city the patient lives in.
<i>city&lt;varchar(75)&gt;</i>	City the patient lives in.
<i>street&lt;varchar(75)&gt;</i>	Patient's street address.

Table 3.3: Database table column description for the Patient\_Address table.

### Patient Diagnosis

Patient_Diagnosis	
<i>Name&lt;datatype&gt;</i>	<i>Description</i>
<i>id&lt;int&gt;</i>	Primary key of the table. It is set to auto-increment.
<i>patient_id&lt;int&gt;</i>	Foreign key referencing the Patient_Data table's id.
<i>result&lt;varchar(75)&gt;</i>	Result of the diagnosis.
<i>image_name&lt;varchar(250)&gt;</i>	Image name used for the diagnosis.

Table 3.4: Database table column description for the Patient\_Diagnosis table.

## Patient Treatment

Patient_Treatment	
<i>Name</i> <datatype>	<i>Description</i>
<i>id</i> <int>	Primary key of the table. It is set to auto-increment.
<i>diagnosis_id</i> <int>	Foreign key referencing the Patient_Diagnosis table's id.
<i>treatment</i> <TEXT>	Treatment of the patient.
<i>issued_by</i> <TEXT>	Person who issued the patient's treatment.
<i>created_at</i> <datetime>	Timestamp of the creation of the record. The default value is set to be the current time.

Table 3.5: Database table column description for the Patient\_Treatment table.

## 3.6 Tests

### 3.6.1 Manual Testing for CNN

To ensure that the Convolutional Neural Network (CNN) model used in the application is functioning correctly and returning the correct values for uploaded chest X-ray images, the following manual tests were performed:

1. Test chest X-ray photos were gathered, including images that the model was trained on to ensure that it returns the correct results.
2. The prediction page on the web server was opened.
3. First the training photos were uploaded to ensure that the model was working correctly and returning the correct predictions.
4. Then chest X-ray images that the model was not trained on were uploaded to ensure that the model is correctly predicting unseen data.
5. Random images from the Internet were also tested that were not necessarily chest X-ray images. For these inputs the program returned a random prediction value.

It is important to note that only image files in the following formats can be uploaded to the web application: png, jpg, jpeg, tiff, bmp, and gif. Files in other formats will not be accepted.

By performing these manual tests, it was confirmed that the CNN model is functioning correctly and providing accurate results for the specific chest X-ray images that the model was trained to work on. It was also verified that the CNN model returns a random prediction for photos that are not chest X-ray images.

#### 3.6.2 Manual Testing for Flask Routes

To ensure that the Flask web application routes are working correctly and returning the expected outputs, the following manual tests were performed:

1. Each route was tested manually by navigating to the corresponding pages using the navigational bar and cards on the webpage and checking if the expected output was displayed.
2. The input validation for each route was tested by providing both valid and invalid input data to the form fields. This includes testing with empty and null inputs, inputs of incorrect data types, and inputs that exceed the maximum length that is allowed.
3. The file upload functionality of the application was tested by uploading various files of different types and sizes. Only image files in the following formats were accepted: png, jpg, jpeg, tiff, bmp, and gif. Other file types were not uploaded as the application only accepts image files.
4. Additionally, the functionality of the database was thoroughly tested by inserting test data and verifying that it is stored and retrieved correctly.

By performing these manual tests, it was confirmed that the Flask web application routes are working correctly and providing the expected outputs for various input scenarios. It was also verified that file uploads are functioning correctly and that the database is working as intended.

## Chapter 4

## Conclusion

During the creation of my thesis, I explored the use and creation of Convolutional Neural Network (CNN) models for detecting pneumonia and implemented a simple to use web interface for uploading and analyzing chest X-ray images. The created model achieved a high level of accuracy in distinguishing between normal and pneumonia cases, with an accuracy of 91%. The web application became simple and easy to use, however it is not without its limitations. Due to the lightweight database engine, the database structure and SQL queries had to be simplified, which in a real-world scenario could limit scalability. The web application could benefit from additional error handling and some design improvements on the user interface to enhance user-friendliness. All in all, my thesis highlights the potential benefits of using machine learning in healthcare to improve diagnostic accuracy and follows the increasing trend of web-based applications.



# Bibliography

- [1] Jonathan Ive. “Simplicity isn’t simple”. In: The Telegraph, 2014.
- [2] Kermany Daniel; Zhang Kang; Goldbaum Michael(2018); *Labeled Optical Coherence Tomography(OCT) and Chest X-Ray Images for Classification*. 2018. DOI: 10.17632/rscbjbr9sj.2. URL: <https://data.mendeley.com/datasets/rscbjbr9sj/2>.
- [3] Aston Zhang et al. “Dive into Deep Learning”. In: *arXiv preprint arXiv:2106.11342* (2021).
- [4] Bálint Molnár. *Pneumonia Detection with Convolutional Neural Networks and a Web-Based Software implementation*. 2023. URL: <https://github.com/Yama-Yama-Yama/Thesis.git> (visited on 05/12/2023).
- [5] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.
- [6] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.
- [7] Richard D Hipp. *SQLite*. Version 3.31.1. 2020. URL: <https://www.sqlite.org/index.html>.
- [8] Armin Ronacher. *The comprehensive WSGI web application library*. 2022. URL: <https://pypi.org/project/Werkzeug/>.
- [9] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).
- [10] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: <https://doi.org/10.1038/s41586-020-2649-2>.
- [11] François Chollet et al. *Keras*. <https://keras.io>. 2015.

- [12] Zope Foundation and Contributors. *Waitress WSGI server*. 2022. URL: <https://pypi.org/project/waitress/>.
- [13] Thomas Kluyver et al. “Jupyter Notebooks – a publishing format for reproducible computational workflows”. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87–90.
- [14] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: <https://doi.org/10.5281/zenodo.3509134>.
- [15] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.
- [16] Michael L. Waskom. “seaborn: statistical data visualization”. In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: <https://doi.org/10.21105/joss.03021>.
- [17] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [18] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/>.

# List of Figures

2.1	Homepage of the website . . . . .	7
2.2	Navigation to add new patient . . . . .	7
2.3	Adding new patient . . . . .	8
2.4	Adding new patient form . . . . .	8
2.5	Editing existing patient . . . . .	9
2.6	Editing existing patient . . . . .	9
2.7	Removing patients . . . . .	9
2.8	Choosing patient for prediction . . . . .	10
2.9	Choosing patient for prediction . . . . .	10
2.10	Browse button click . . . . .	11
2.11	Selecting a file to upload . . . . .	11
2.12	Submit button click . . . . .	12
2.13	Predict button click . . . . .	12
2.14	Removing treatment and diagnoses . . . . .	13
2.15	Adding treatment . . . . .	13
2.16	Adding treatment modal box . . . . .	14
2.17	Removing treatments . . . . .	14
2.18	Removing treatment and diagnoses . . . . .	15
3.1	Folder structure of the program. . . . .	17
3.2	Architecture of the Convolutional neural network model' photos . . .	23
3.3	Example data after normalization . . . . .	27
3.4	Class imbalance . . . . .	27
3.5	Training curve of the model . . . . .	28
3.6	Confusion Matrix . . . . .	29
3.7	Database Relationship Diagram . . . . .	41

# List of Tables

2.1	Route table for user documentation. . . . .	15
3.1	Database table column description for the Patient_Data table. . . . .	42
3.2	Database table column description for the Patient_Contact table. . . . .	43
3.3	Database table column description for the Patient_Address table. . . . .	43
3.4	Database table column description for the Patient_Diagnosis table. . . . .	43
3.5	Database table column description for the Patient_Treatment table. . . . .	44