GDAŃSK UNIVERSITY OF TECHNOLOGY
FACULTY OF APPLIED PSYCHICS AND MATHEMATICS

Student's name and surname: Bartosz Rydziński
ID: 174189
Cycle of studies: undergraduate
Mode of study: Full-time studies
Field of study: Technical Physics
Specialization: Applied Computer Science

Student's name and surname: Aleksander Obuchowski
ID: 174086
Cycle of studies: undergraduate
Mode of study: Full-time studies
Field of study: Technical Physics
Specialization: Applied Computer Science

ENGINEERING DIPLOMA THESIS

Title of thesis: Analysis of X-ray scans with reports and complete blood count (CBC) tests and their applications in COVID-19 machine diagnosis based on artificial neural network

Title of thesis (in Polish): Analiza zdjęć rentgenowskich klatki piersiowej wraz z opisami i badań morfologii krwi w celu diagnozy maszynowej COVID-19 opartej na sieciach neuronowych

Supervisor: dr inż. Patryk Jasik

Date of final approval of the similarity report in Uniform Anti-plagiarism System (JSA):

Streszczenie

W niniejszej pracy badamy przepływ pracy, który może zostać wykorzystany w tworzeniu systemów diagnozy maszynowej COVID-19 bazującej na uczeniu maszynowym wykorzysującym dane o różnej modalności. Obecnie dane medyczne przechowywane są w sposób niezorganizowany, co czyni je trudnymi w analizie. Analiza tego typu może posłużyć kwalifikacji wstępnej do testów Real Time PCR, które są kosztowne, wymagają specjalistycznego sprzętu laboratoryjnego, a także zajmują nawet dobę i ich ilość jest ograniczona.

Opisujemy metody przetwarzania danych, szczególnie dużych medycznych zbiorów, oraz sposób tworzenia przyjaznej dla użytkownika strony internetowej pozwalającej na otrzymanie diagnozy maszynowej po wprowadzeniu wyników badań.

Przedstawiamy także trzy modele, które mogą być zastosowane, autonomicznie lub osobno, w diagnostyce COVID-19. Pierwszy model analizuje język naturalny zawarty w opisach stworzonych przez lekarzy, aby znaleźć i oznaczyć objawy związane z infekcją COVID-19, w szczególności związane z sercem oraz płucami. Drugi model wykorzystuje splotowe sieci neruonowe w diagnozie maszynowej opartej na zdjęciach rentgenowskich. Badamy także model klasyfikacyjny służący do analizy wyników morfologii krwi oraz metody równoważenia ilości danych, szczególnie podpróbkowanie (ang. *undersampling*) oraz nadpróbkowanie (ang. *oversampling*).

Słowa kluczowe: COVID-19, Uczenie Maszynowe, Sieci Neuronowe, Interfejs Użytkownika, Przetwarzanie Języka Naturalnego, Przetwarzanie Danych, Dane o Niezrównoważonym Rozkładzie Klas

Dziedzina nauki i techniki, zgodnie z wymogami OECD: 1.2.a

Abstract

In this thesis, we investigate the workflow that can be applied in developing COVID-19 machine learning diagnosis systems based on multimodal data. Currently, medical records are stored in unorganized manner, making it hard to analyse. Such analysis can be beneficial for prequalifying for Real Time PCR tests that take up to a day, are expensive, often limited and require specialized laboratory equipment.

We describe possible methods of data processing, especially large medical datasets as well as present a way of creating a user-friendly website that allows anyone to get a prediction based on their input. We also present three models, that can be applied, as standalone solutions or stages in machine COVID-19 diagnosis. The first model performs natural language analysis on reports, written by doctors, to extract COVID-19 infection related features, especially describing heart and lung conditions. The second model uses convolutional neural networks for COVID-19 diagnosis based on X-ray scans. We also investigate classification model for analysing complete blood count results as well as methods of data balancing, namely undersampling and oversampling.

Keywords: COVID-19, Machine Learning, Neural Networks, User Interface, Natural Language Processing, Data Processing, Imbalanced Data

Field of science and technology in accordance with OECD requirements: 1.2.a

Acknowledgements

We want to thank Anna Muraszko-Klaudel, M.D., Ph.D. for helping us with translations and explanations of phrases used in medical reports and Michał Jędrzej Stańczyk for providing us with the MTT tool as well as it's explanation and usage guide.

Contents

Chapter 1

List of important symbols and abbreviations

Unless stated otherwise, the symbols in this document are as follow:

- NLP - Natural language processing,
- NER - Named entity recognition,
- CNN - Convolutional neural networks,
- CBC - Complete blood count,
- CSV, .csv - Comma separated file,
- nan - Not a number,
- JSON - JavaScript Object Notation,
- .xlsx - Excel Microsoft Office Open XML Format Spreadsheet,
- JSX, .jsx - JavaScript XML,

- TSX, .tsx - TypeScript XML,
- WSGI - Web Server Gateway Interface,
- REST - Representational state transfer,
- HTTP - Hypertext Transfer Protocol,
- SQL - Structured Query Language,
- DOM - Document Object Model,
- pip - Preferred installer program,
- npm - Node package manager,
- CSS, .css - Cascading style sheets,
- SASS - Syntactically awesome style sheets,
- SCSS, .scss - Sassy cascading style sheets,
- UCC - University Clinical Centre,
- ABG - Arterial blood gas,
- VBG - Venous blood gas,
- ICD - International classification of diseases,
- JWT - JSON Web Token,
- URI - Uniform resource identifier,
- SSL - Secure sockets layers,
- FC - Functional component,

- RFC - Request for comments,
- DSL - domain-specific language
- MIDAS - Medical Image Data Annotation Service, ROC - Receiver operating characteristic,
- AUC - Area Under the (ROC) Curve,
- ReLU - Rectified Linear Units,
- SMOTE - Synthetic Minority Over-sampling Technique,
- DICOM - Digital Imaging and Communications in Medicine,
- PNG - Portable Network Graphics,
- COVID-19 - Coronavirus Disease 2019,
- SARS-CoV-2 - Severe Acute Respiratory Syndrome Coronavirus 2.

Chapter 2

Introduction, Objectives and Aims of thesis

2.1 Introduction

In our thesis we explore the idea of COVID-19 machine learning diagnosis based on multiple data sources and the challenges behind it. Currently, formal diagnosis of COVID-19 requires a laboratory test (RT-PCR) that requires specialized equipment and takes at least 24 hours to produce a result [1]. As the number of PCR tests is often limited, these tests have to be distributed selectively. A machine learning based solution having the ability to diagnose COVID-19 without the need for RT-PCR tests would greatly benefit the hospital infrastructure. Although such system, even if achieving outstanding accuracy proven by extensive tests, would most probably not replace PCR tests in the formal diagnosis, it could help in identifying high-risk individuals and result in better coverage of the tests.

In today's world, we have to face daunting challenge of the global pandemic, however we are also in an unprecedented moment in history, having access to the abundance

of data including medical records of patients. This data can be a powerful weapon in the fight with SARS-CoV-2 as it can be used to create diagnostic systems based on machine learning solutions that can increase the accuracy of diagnosis without the necessity of engaging additional medical personnel.

In our work we present the pathway for dealing with such large medical data depicting the necessary steps i.e. processing and sanitizing the data, building machine learning models and developing the final solution in the form of web application. More precisely we focus on the following 3 models:

1. Natural Language Processing model for parameterizing chest X-ray reports to facilitate their use in classification task,

2. Image recognition model for diagnosing COVID-19 chest X-ray images,

3. Fully connected deep network diagnosing COVID-19 based on complete blood count data.

We also present the technologies used and the motivation behind using them, challenges of data pre-processing and their solutions as well as possible future steps to take in the analysis.

2.2 Motivation

The global pandemic of SARS-CoV-2 virus has affected the whole population and every aspect of our lives. By the time of writing this thesis the global death count of coronavirus is said to be about 1.9 million with over 30 000 deaths in Poland alone. Many of the affected states lack the funding and infrastructure necessary to perform sufficient number of tests. In such cases a machine learning system being able do identify

the people having high risk of being infected with SARS-CoV-2 would serve as an aid to prioritize testing and therefore make better use of the given resources.

Our motivation also includes providing an educational resource for people dealing with similar problems. That's why we provide an in-depth descriptions of used technologies, architectures and methods used to develop the final solution.

2.3 Objective

In our work we are hoping to identify the possibilities and problems of data analysis in relation to COVID-19 machine learning prediction. Our goal is to build foundations for future systems by exploring necessary steps and possible directions in working with the data. More precisely we are hoping to answer the following research questions:

- RQ1.1 What are the necessary steps for processing large medical datasets?
- RQ2.1 Can X-ray reports written in natural language be parameterized using pattern-matching sequence tagging models?
- RQ3.1 What is the performance of popular image recognition convolutional neural networks on the task of diagnosing COVID-19 based on X-ray images?
- RQ3.2 Can the accuracy of said models be increased by training them to recognize additional types of pneumonia infections?
- RQ3.3 Can the accuracy of said models be increased by using transfer learning and utilizing larger chest X-ray datasets?
- RQ4.1 Can we diagnose COVID-19 basing on complete blood count data?

Moreover, in this thesis we take on the task of developing an end solution for the diagnosis of COVID-19 based on complete blood count and X-ray images. Our aim

is to create both the server infrastructure for deploying machine learning models and front-end part of the application providing streamline user experience. By achieving the above-mentioned goals we are hoping to build a cornerstone for machine learning based systems dealing with multimodal data aggregated from different sources.

Chapter 3

Technologies used

3.1 General information

For its broad appliance in data science, great support of data analysis and manipulation, we have decided to use Python [2] as the main programming language used throughout the creation of this project. Every part of our work had it's own virtual environment created using the virtualenv [3] module to make the deployment easy to run across different platforms. Using Python enables being more productive and allows to utilize standard library with a lot of helpful packages as well as the vast number of community supported packages that we managed using pip package manager [4]. The list of used libraries along with their versions can be found in table **3.1**.

3.2 Data analysis

Python's standard library contains several different component, that come together to make developer's life easier. These modules are written in either Python or C programming language and can help manage cross-platform compatibility or are strictly

related to Python code, like multiple children of Exception class. The most commonly used module for us was os [5] that allows to read directory contents, create operating system safe paths, create and delete folders or files, often used complementary with shutil [6], which extends these possibilities to work with multiple elements and adds another layer of security.

Data preparation requires a lot of work with comma-separated values (CSV) and pandas [7] is an obvious choice. It allows for fast data manipulation, filtering, searching, counting and clear representation, especially in Jupyter Notebook. Pandas utilizes Data Frames which are two-dimensional data structures, that can and most often are used as a collection of rows and columns. They can be initialized with any iterable data type, or directly from a file, like .csv files, which are tabular in their nature. Data Frames are a great way to work with the data they contain. Pandas naturally comes with numpy [8] but the only features of it we needed in data analysis was checking if the value was nan (not a number). Another helpful package was xlsxwriter [9] to allow reading and writing .xlsx files, which can be thought of as multiple comma-separated values files with metadata.

For our deep learning based solutions we have used Keras library, that is built on top of TensorFlow 2.0. This choice was motivated mainly by the popularity of the framework and therefore an abundance of available solutions, including pre-trained models. TensorFlow also enables easy methods of deployment of the model, through tensorflow-serving, what was specially important to us as the end goal of this project was to develop and deploy the complete application.

For natural language processing we used MTT language that is described in greater detail in section 5.2

3.3 User interface

The web application we prepared aims to be as accessible and easy-to-use as possible in order to reach most people, even if they do not use computer daily or have trouble using it. The user should be well informed what is expected to be the input, where they can find it and what to do if they can not or do not want to provide it. The results from the system should be informative and clearly represented to avoid misinterpretation.

User interface is split into two separate parts, one is the server managed by flask [10], which is a Python powered framework. It is responsible for the prediction based on user's input, authorizing users, storing usage statistics and managing users' requests.

The other part is User Interface - a web application that allows users to get predictions. The programming language used for it is TypeScript [11] - an extended version of JavaScript [12], allowing for type safety, which works great with React [13], a powerful, component-based library for building reactive web applications.

3.3.1 Server side

Flask is a web application framework that works on top of the Web Server Gateway Interface (WSGI) [14], designed to make getting started easy and keeping it scalable when the application grows more complex. It's functionalities are easily extendable with community created extensions and easy connection to various databases. Flask is also lightweight, fast in development as well as supporting multiple design patterns and leaving freedom of choice to developers.

The most important extension in our project is Flask-RESTful [15] that adds support for building REST application programming interfaces. It provides a way to handle different HTTP [16] methods, handles HTTP errors such as 405 (Method Not Allowed) out of the box and automatically translates JSON string literals to Python's dictionar-

ies, in requests as well as responses.

Our chosen relational database management system is MySQL [17] version 8.0.22 for Linux on x86_64 and we are using SQLAlchemy [18] - an object relational mapper that is wrapped by Flask-SQLAlchemy [19] to easily use it with Flask. Flask-Migrate [20] is another tool that handles database migrations on top of Alembic [21], to ensure database structure and consistency.

3.3.2 Client side

For the user interface we are using React - a JavaScript library, because of its community support and ease of use. It scales very well and is really performant because of the Virtual DOM [22], provides great developer tools that show the structures and data they contain as well as JavaScript syntax extension in the form of JSX (TSX for TypeScript). React being widely used also brings plenty of ready to use extensions or components that can be easily integrated into the application. We managed these extensions with Node package manager (npm).

The library we use to manage application routing is react-router-dom [23] and since the application is REST based, axios [24] serves as HTTP client, because of it being promise based, lightweight and configurable. We also use lodash [25] for many utility functions it provides, how fast it is and how it is not mutating data.

For styling we decided to use Syntactically Awesome Style Sheets (SASS) [26] with Sassy CSS (SCSS) [27] syntax because pre-processing allows us to save time and get the same results faster, as well as allows us to split our code better. This way of writing stylesheets allowed us to limit the scope of classes to another classes, or simply use the "&" shorthand for modifiers.

3.3.3 Other libraries

The modern approach to software development is that you should not reinvent the wheel, thus we used a lot of libraries and packages, presented in table **3.1**.

Table **3.1**: Index of used packages

Python		JavaScript/TypeScript	
Python [2]	3.8.5	TypeScript [11]	4.1.3
pip [4]	20.3.3	npm [28]	6.14.10
virtualenv [3]	20.3.0	React [13]	17.0.1
pandas [7]	1.1.4	react-router-dom [23]	5.2.0
numpy [8]	1.19.4	react-dom [29]	16.9.10
XlsxWriter [9]	1.3.7	axios [24]	0.21.1
Keras [30]	2.2.4	lodash [25]	4.17.20
TensorFlow [31]	2.2.1	node-sass [32]	4.14.1
flask [10]	1.1.12	react-file-reader [33]	1.1.4
Flask-RESTful [15]	0.3.8	react-tooltip [34]	4.2.11
SQLAlchemy [18]	1.3.22		
Flask-SQLAlchemy [19]	2.4.4		
Flask-Migrate [20]	2.5.3		
mysqlclient [35]	2.0.3		
tqdm [36]	5.2.0		
matplotlib [37]	3.3.3		
requests [38]	2.25.0		
urllib3 [39]	1.25.10		
xlrd [40]	1.2.0		

Chapter 4

Data preparation

4.1 Data overview

The data we received from University Clinical Centre (UCC) was split into 8 categories, because the amount of data would make it hard to operate on – each category has two files connected to it: one containing data and the other – headers, amounting to a total of 16 CSV files, which sum up to 2.1 gigabytes. Although comma is a standard delimiter for values and a new line symbol ("\n") separates records, special indicators were used to avoid misinterpretation caused by symbols existing in text – "<==>" for cells and "<NEW_RECORD>\n" for rows.

4.2 Data categories and sample size

Below, the files we received from UCC are listed in following sections, along with their sizes, amount of rows and descriptions of their contents. Every file contains a column named patient_id, which can be used as primary key to match data of one person from multiple files. Data distribution amongst files is shown in figure 4.1.

4.2.1 Laboratory results

The file contains medical tests data, one parameter in each row, which are bound by id of the test. The type of test we are focusing on is blood test as it is cheap, informative and often performed making it a great candidate for screening tests for COVID-19. Since this category contains the most data and covers a lot of laboratory work we decided to split it into the parts listed in table **4.1**. The mentioned files contain the tests that were deemed the most vital for the purpose of analysis, based on consultations with UCC. All other tests were placed into the "Other" category.

Table **4.1**: Data categories along with their files and sizes.

Contents	File name	Size [MB]	No of rows
Laboratory results	WynikiLaboratoryjne.csv	2048	12618775
Measurement of vital parameters	Pomiar-parametrów-życiowych.csv	486.3	435504
Complete Blood Count	Morfologia	362	3352387
Arterial Blood Gas (ABG)	Równowaga-kwasowo-zasadowa-krew-tętnicza-POCT.csv	89.1	606911
Invasive Medicine Centre's medical operation data	CMI—Operacja-Pacjenta.csv	65.8	553591
Venous Blood Gas (VBG)	Równowaga-kwasowo-zasadowa-krew-żylna-POCT.csv	36.6	258144
General urine test	Badanie-ogólne-moczu.csv	35.7	323298
Fluid balance	Bilans-Płynów.csv	35.5	355014
Creatinine blood test	Kreatynina-we-krwi.csv	25	192588
Electrolyte test	Elektrolity.csv	20.7	216224
Complete Blood Count with reticulocytes	Morfologia-z-retikulocytami.csv	17.5	142344
Measurement of vital parameters in newborns	Pomiar-parametrów-życiowych-Noworodka	16.9	133005
Other	Other.csv	276	2130225

4.2.2 Descriptive results

This file stores indexes of medical tests such as CT scans and X-ray or ultrasound examinations along with date and reports provided by doctors, written in natural language, that relate to scans via id of the test. This file is 45.3 MB, containing 49843 rows.

4.2.3 Diagnoses

This file contains one diagnosis per row, identified by International Statistical Classification of Diseases and Related Health Problems code, name, type and kind. The usage of ICD allows us to quickly determine which patients had issues with respiratory system, as they are described using Chapter X Diseases of the respiratory system [41], which contains the blocks presented in table **4.2**. This file is 39.7 MB and contains 282692 rows.

Table **4.2**: Diseases of the respiratory system.

Codes range	Description
J00-J06	Acute upper respiratory infections
J09-J18	Influenza and pneumonia
J20-J22	Other acute lower respiratory infections
J30-J39	Other diseases of upper respiratory tract
J40-J47	Chronic lower respiratory diseases
J60-J70	Lung diseases due to external agents
J80-J84	Other respiratory diseases principally affecting the interstitium
J85-J86	Suppurative and necrotic conditions of lower respiratory tract
J90-J94	Other diseases of pleura
J95-J99	Other diseases of the respiratory system

4.2.4 Categories of medical care

This file describes types of medical care based on patient's health condition, especially mobility and their capabilities of fulfilling basic needs on their own. These states

are categorized as shown in the table **4.3**. It is 386.7 kB containing 4131 rows.

Table **4.3**: Medical care categories.

Category	Type of care
I	minimal care
II	moderate care
III	increased care
IV	intensive care

4.2.5 Other categories

The sections above cover data that we focused the most on and that required more detailed descriptions; however, below we list other files, that were vital to the analysis, especially COVID-19 tests.

- COVID-19 tests (TestResult.csv – 2.0 MB, 34814 rows);

  containing the information about all COVID-19 tests that the patient had gone through with result and their date. The result can be positive, negative or ambiguous.

- Patient data (Pacjent.csv – 1.2 MB, 25441 rows);

  matching patient's id with date of birth and gender.

- Hospital stays (Pobyty.csv – 286.3 kB, 1322 rows);

  containing data of patient's hospital stays along with the hospital ward and location of discharge.

- Procedures (Procedury.csv – 9.6kB, 105 rows);

  storing data about procedures applied in the course of treatment, most importantly records the usage of life-support machine or oxygen therapy that are often used in curing COVID-19.

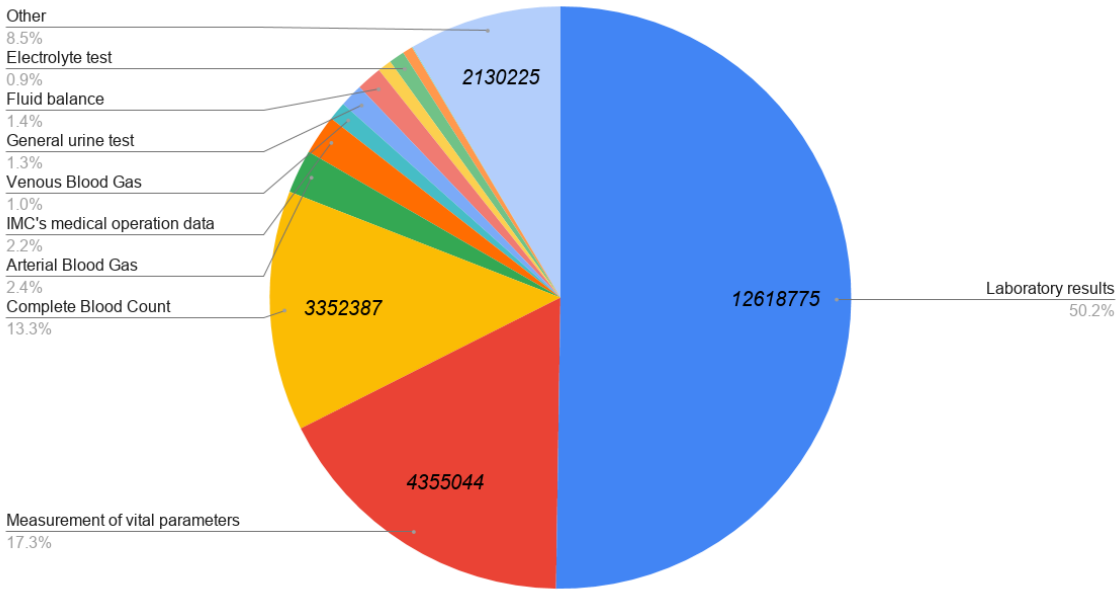


Figure 4.1: Distribution of data in files.

4.3 Data cleaning

Although pandas [7] is a very powerful tool, we quickly reached it's limitations. Firstly, we could not use the much faster C parsing engine, as it does not support multi-character delimiters, so we had to use the Python engine, which; however, does not support line terminators other than "\n", secondly we had a lot of trouble reading these files as the fields with long text weren't properly escaped, so every "\n" or " " " could have caused an exception.

In order to make it possible to read these files using pandas, we had to manipulate the files using Python's standard library. The first step was to replace separators (<==>) with a character that cannot be found anywhere in the given data. Because of how vast it is, containing values, units and natural language, we could not go for any character from Latin alphabet, polish diacritics, numbers or even non-alphanumeric symbols found on the keyboard, thus we decided to go with something foreign and used the Han character – 字.

Secondly, we had to make sure rows are properly delimited and in order to do that, we had to remove all new line characters from the file, then replace "<NEW_RECORD>" with "\n". Even that was not enough, as it turned out that some of the quotes were not closed, which stopped pandas from reading it correctly, so to cover for that, every " " " was replaced with " ' ", which is documented in listing 4.1.

Listing 4.1: Files sanitizing script

```
import pandas as pd
import os

files = os.listdir('original')

for file in files:
    with open(f'original/{file}', 'r') as f:
        org = f.read()

    org = org.replace('\n', '')
    org = '\n'.join(org.split('<NEW_RECORD>'))

    contents = org.replace('"', "'")

    with open(file, 'w') as f:
        f.write(contents)
```

Smaller files were not causing any more trouble, but the laboratory results file, because of its size had to be read in chunks (luckily, pandas has this option built in) as reading it all took a few minutes and caused our computers to freeze.

Chapter 5

Natural language data analysis in X-ray reports using MTT

5.1 Dataset and Problem Statement

After excluding bedside chest scans, our dataset consisted of 2043 chest X-ray reports written in Polish. As those reports were written in natural language, they could not be used as features in the classification tasks.

In this chapter, we take on the problem of featurizing the texts - converting them to parameterized form, where features, such as focal lesions (*"Zmiany ogniskowe"*), can either be present or absent in the text. This can later be used as a base for creating many-hot encoded representation of the texts that can be used as an input to deep neural networks.

Given the lack of pre-trained medical natural language processing models for Polish language and the cost of annotating the reports for training sequence tagging models, we decided to develop rule-based model for feature extraction. For developing this model, we used the MTT language.

5.2 MTT Language

MTT is a pattern-matching domain-specific language (DSL) aimed at building bottom-up sequence-labeling systems. It comes with an interactive browser-based editor/debugger, and an optimising compiler with Scala [42] and Python back-ends. The language was loosely inspired by AWK [43] and Alex Shinn's match macro [44, 45] for Algorithmic Language Scheme.

MTT was developed in SentiOne [46] in 2019 out of practical need for an easy, fast and portable solution for use and reuse inside larger systems. The successful applications of MTT include a set of brand crises detection models for user-generated content, advertisement/sponsored content detector, greetings removal module, and NER for names of brands, companies, cosmetics as well as banking products. It also helps with information discovery on large unannotated corpora, thus largely replacing earlier use of UNIX grep [47].

The generated stand-alone Python modules and Scala objects have surprisingly good performance and very little memory footprint, making them suitable for use within larger systems (most notably web scrapers).

MTT script consists of a sequence of commands, conceptually executed in their order of appearance (in practice the compiler postpones some computations until they are necessary).Each command either expresses which text fragments should get annotated, or which annotations (labels) made so far should be removed. Commands introducing new labels specify tokens pattern to be matched; commands removing labels are either unconditional (remove all occurrences of a given label) or conditional (remove all the labels inside, containing, or precisely fitting some other label).
Patterns are similar to regular expressions with three differences:

- they operate on entire tokens, not characters,

- they describe only finite languages (*i.e.* there are no Kleene star [48] equivalents),
- they allow bottom-up construction and working on annotations from other systems by introducing references to previously annotated fragments.

A sample script in MTT language is presented in listing 5.1

Listing 5.1: MTT script example

```
LABEL "snow" "is" ("white"|"yellow") AS @"color_statement"
LABEL ("Alfred"|"Frank") "says" @"color_statement" AS @"indirect_speech"
UNLABEL @"color_statement" IN @"indirect_speech"
```

A system described with the script in listing 5.1, given any tokenized text:

1. annotates spans of the form *"snow is white"* and *"snow is yellow"* with label `"color_statement"`, then
2. annotates with label `"indirect_speech"` all spans starting with *"Alfred says"* or *"Frank says"* immediately followed by any span annotated with label `"color_statement"`, finally
3. removes labels from all the spans which happen to be inside spans annotated with label `"indirect_speech"`.

E.g. the text *"Although Aflred says snow is white, this snow is yellow."* would get two annotations: *"Alfred says snow is white"* with `"indirect_speech"` and *"snow is yellow"* with `"color_statement"`.

We chose MTT because of the following advantages:

- the web-based environment allows training new users, as well as development and maintenance of scripts – a feature most other solutions lack (PCRE [49], Stanford's tokenregex [50], nltk chunker [51] *etc.*),

- reduced expressibility of patterns allows efficient compilation on one hand, while on the other it requires users to describe utterances in more detailed, easy to understand form (unlike regular expressions [49], ATNs [52], or "unify and evaluate" formalisms like Spejd [53]),

- the implementation does not penalize working on ambiguous semantic forms (more precisely: on utterances expressing these) – neither in script size, nor in performance (unlike traditional parser generators of LALR [54] and PEG [55] kind in the former, extended regular expressions in the latter case),

- due to imperative component of commands, the problem of prioritising rules or patterns (as in CPSL/Yape [56, 57], Spejd [53] *etc.*) was eliminated completely,

- the simple "tokens+labels" format allows for easy integration with other scripts, as well as foreign tokenizers, taggers, classifiers *etc.*,

- the primitive construction of compiler's back-end allows adding new target languages easily.

5.3 Models

Using MTT, we developed 2 basic models for chest X-ray reports. First model's role was to capture heart related anomalies and the second model captured lung related anomalies.

5.3.1 Heart

During development of the heart model, we identified that 2 most dominant heart features were related to its size. The heart could be either described as enlarged or

normal. Third most dominant heart feature was excessively horizontal position of the heart. The heart could also be described as "difficult to assess". Given the initial analysis, we have developed the rule based model written in MTT language to capture those features. The model is shown in the listing 5.2. The last 3 lines were written to capture the occurrences of the word "heart" and its variants that did not belong to the above mentioned features, enabling us to calculate precision of the model with respect to this word. The number of tags captured and the model analysis is shown in the section 5.4.1.

Listing 5.2: Heart model

```
LABEL (a"serce" | a"sylwetka" a"serca"| a"sylwetka" a"sercowo" a"-" a"naczyniowa") AS @"SERCE"
LABEL  @"SERCE" (a"i" _ |) (a"rtg"|) ((a"wielkoscia"|) a"w" a"normie" | ar"niepowiekszon(e|a)"
|ar"nieposzerzon(e|a)"| a"w" (_|) ar"granic(y|ach)"  (_|) (ar"norm(y|)"|a"n")) AS @"SERCE W NORMIE"
LABEL a"prawidlowa" @"SERCE" AS @"SERCE W NORMIE"
LABEL @"SERCE" (_|) (ar"powiekszon(e|a)"|a"wieksze") AS @"SERCE POWIĘKSZONE"
LABEL @"SERCE" (_|) a"podparte" a"na" a"przeponie" AS @"SERCE PODPARTE NA PRZEPONIE"
LABEL (_|) ar"powiekszon(e|a)" @"SERCE" AS  @"SERCE POWIĘKSZONE"
LABEL @"SERCE" (a"nie"| ar"trudn(e|a)") a"do" a"oceny" AS @"SERCE NIE DO OCENY"
LABEL a"rtg" AS @"RTG"
UNLABEL @"SERCE" IN @"SERCE W NORMIE"
UNLABEL @"SERCE" IN @"SERCE POWIĘKSZONE"
UNLABEL @"SERCE" IN @"SERCE PODPARTE NA PRZEPONIE"
UNLABEL @"SERCE" IN @"SERCE NIE DO OCENY"
```

5.3.2 Lungs

The lung model was much more complicated to develop as there were a lot of lung related anomalies to capture in the X-ray reports. We decided to limit the model to capturing the following features:

- erration of lungs e.g "normal erration of the lung",
- effusion e.g. "the right pleural effusion",
- pneumonic consolidation, e.g. "lungs with no pneumonic consolidation",
- opacities, e.g. "no parenchymal densities", "lung parenthymal without densities",

- focal lesions,
- pneumonic consolidation,
- reticular opacities,
- pneumotorax,
- decompresion of lungs,
- lung interstitial,
- emphysematous lungs.

Parameters related to COVID-19 and their relation have been shown in table **5.1**.

Table **5.1**: Lung parameters and their relation to COVID-19

Parameter	Relation to COVID-19
decreased general aeration of the lung	can indicate COVID-19
reticular opacities	can indicate COVID-19
lung interstitial	can indicate COVID-19
pneumonic consolidation	can indicate to COVID-19
pleural effultion	indicates infection other than COVID-19
lungs with no pulmonary opacities	excludes COVID-19
lungs with no pneumonic consolidation	excludes COVID-19
emphysematous lung	excludes COVID-19
lung parenthymal without densities	excludes COVID-19
no parenchymal densities	excludes COVID-19
pneumonic opacity	excludes COVID-19
hilar enlargement	excludes COVID-19

The model is shown in the listing 5.3. Although it is not shown in the listing, similarly to the heart model, the last 16 lines were written to capture the occurrences of the word "lungs" and its variants that did not belong to the above mentioned features, enabling us to calculate precision of the model with respect to this word. The number of tags captured and the model analysis is shown in the section 5.4.2.

Listing 5.3: Lungs model

```
LABEL (a"pluca"|a"pluco"| a"pola" a"plucne") AS @"PLUCA"
LABEL ar"lewe(go|)" @"PLUCA" AS @"LP"
LABEL a"prawe(go|)" @"PLUCA" AS @"PP"
LABEL @"PLUCA" ar"prawe(go|)" AS @"PP"
LABEL @"PLUCA" ar"lewe(go|)" AS @"LP"
LABEL @"PP" _ <0,2> (a"prawidlowo"|) a"powietrzne" AS @"PP_PRAWIDŁOWO_POWIETRZNE"
LABEL @"LP" _ <0,2> (a"prawidlowo"|) a"powietrzne" AS @"LP_PRAWIDŁOWO_POWIETRZNE"
LABEL @"PLUCA" (a"rozprezone" ","|) (a"prawidlowo"|) a"powietrzne" AS @"PLUCA PRAWIDLOWO POWIETRZNE"
UNLABEL @"PLUCA PRAWIDLOWO POWIETRZNE" IN @"LP_PRAWIDŁOWO_POWIETRZNE"
UNLABEL @"PLUCA PRAWIDLOWO POWIETRZNE" IN @"PP_PRAWIDŁOWO_POWIETRZNE"
LABEL (a"gorsza"|a"zmniejszona") a"powietrznosc" @"PLUCA" AS @"ZMIEJSZONA POWIETRZNOSC"
LABEL @"PLUCA" a"bez" a"zgeszczen" a"miazszowych" AS @"BEZ_ZAGESZCZEN_MIASZOWYCH"
LABEL ar"lew(a|ej)" (ar"jam(a|ie)"|a"j"|a"j.") (a"."|) (a"opl."|a"opl"|ar"oplucnow(a|ej)") (a"."|) AS
@"LEWA JAMA"
LABEL a"jama" (_|) a"lewa" AS @"LEWA JAMA"
LABEL ar"praw(a|ej)" (ar"jam(a|ie)"|a"j"|a"j.") (a"."|) (a"opl."|a"opl"|ar"oplucnow(a|ej)") (a"."|) AS
@"PRAWA JAMA"
LABEL a"jama" (_|) a"prawa" AS @"PRAWA JAMA"
LABEL ar"plyn(u|)" a"w" @"LEWA JAMA" AS @"PLYN LEWA JAMA"
LABEL ar"plyn(u|)" a"w" @"PRAWA JAMA" AS @"PLYN PRAWA JAMA"
LABEL a"jamy" (a"oplucnowe"|) AS @"JAMY"
LABEL @"JAMY" a"wolne" "od" a"plynu" AS @"JAMY WOLNE OD PLYNU"
LABEL @"LEWA JAMA" a"wolna" AS @"LEWA JAMA WOLNA"
LABEL @"PRAWA JAMA" a"wolna" AS @"PRAWA JAMA WOLNA"
LABEL a"plyn" a"w" a"obu" a"jamach" AS @"PLYN OBIE JAMY"
LABEL @"JAMY" a"wolne" "od" (_|) (a"i"|) a"odmy" AS @"BRAK ODMY"
LABEL a"odmy" (_|) <0,2> a"nie" (a"wykazano"|a"widac") AS @"BRAK ODMY"
LABEL @"PLUCA" (_|) <0,5> (a"bez"|a"i") (_|) a"zmian" a"ogniskowych" AS @"BEZ ZMIAN OGNISKOWYCH"
LABEL @"PLUCA" (_|) <0,2> a"bez" (_|) (a"zageszczen"| a"zgeszczen") AS @"BEZ ZAGESZCZEN"
LABEL  (ar"zage?zczen"| ar"zge?zczen") a"nie" a"widac" AS @"BEZ ZAGESZCZEN"
LABEL @"PLUCA" (_|) <0,5> (a"bez"|a"i") a"cech" a"zastoju" AS @"BEZ CECH ZASTOJU"
LABEL @"PLUCA" (_|) <0,2> a"bez" a"zmian" a"zapalnych" AS @"BEZ ZMIAN ZAPALNYCH"
LABEL a"zmiany" a"zapalne" AS @"ZMIANY ZAPALNE"
LABEL a"zgeszczenia" a"zapalne" AS @"ZGĘSZCZENIA ZAPLANE"
LABEL @"PLUCA" a"bez" (_|) a"zmian" a"miazszowych" AS @"BEZ ZMIAN MIAZSZOWNYCH"
LABEL @"PLUCA" (a"rozedmowe" | a"z" a"cechami" a"rozedmy") AS @"ROZEDMOWE"
LABEL @"MIĄŻSZ" a"plucny" a"bez" (_|) (a"zageszczen"|a"zgeszczen") AS @"MIĄŻSZ BEZ ZAGĘSZCZEŃ"
LABEL (ar"siateczkow(e|o)"|a"siateczkowate") AS @"SIATECZKOWATE"
LABEL ar"zrebo(wych|wymi|wym)" AS @"ZRĘB"
LABEL a"zastoj" AS @"ZASTOJ"
LABEL  @"PLUCA" a"rozprezone"  AS @"PLUCA ROZPREZONE"
LABEL a"cien" a"srodpiersia" a"gornego" AS @"CIEN ŚRÓDPIERSIA GÓRNEGO"
LABEL @"CIEN ŚRÓDPIERSIA GÓRNEGO" (_| a"i" _ |) (a"powiekszony" | a"poszerzony" |"szerszy") AS @"CIEN
ŚRÓDPIERSIA GÓRNEGO POWIĘKSZONY"
LABEL (_|) (a"powiekszony" | a"poszerzony" | a"szerszy") @"CIEN ŚRÓDPIERSIA GÓRNEGO"  AS @"CIEN
ŚRÓDPIERSIA GÓRNEGO POWIĘKSZONY"
LABEL a"aorta" (a"piersiowa"|) AS @"AORTA"
LABEL @"AORTA" (ar"(z|)miazdzycowa(na|)" | ar"z(e|)" (_|) ar"blaszk(ami|a)" ar"miazdzycow(ymi|a)" |
a"zmieniona" a"miazdzycowo") AS @"AORTA MIAŻDZYCOWA"
LABEL @"AORTA" ((_|) a"sklerotyczna" (a"w" a"luku" |) | a"z" a"cechami" a"sklerotyzacji") AS @"AORTA
MIAŻDZYCOWA"
LABEL a"sklerotyczna" (a"w" a"luku" |) @"AORTA" AS @"AORTA MIAŻDZYCOWA"
LABEL @"AORTA" (_|) (a"szersza" | a"powiększona" | a"poszerzona" | a"o" a"cechach" a"poszerzenia")
(a"w" a"luku"|) AS @"AORTA POSZERZONA"
LABEL (a"szersza" | a"powiększona" | a"poszerzona") @"AORTA" AS @"AORTA POSZERZONA"
LABEL @"AORTA" a"wydluzona" AS @"AORTA WYDŁUŻONA"
LABEL @"AORTA" (a"kreta" | a"o" a"kretym" a"przebiegu") AS @"AORTA KRETA"
LABEL @"AORTA" a"zstepujaca" AS @"AORTA ZSTEPUJACA"
LABEL a"wneki" ar"pluc(ne|)" AS @"WNEKI PŁUCNE"
LABEL @"WNEKI PŁUCNE"   (_|) (ar"szersz(e|a)" | ar"powiększon(e|a)" | ar"poszerzon(e|a)"|
ar"szeroki(e|a)") (_ <0,5> a"i" |) AS @"WNEKI PŁUCNE POWIĘKSZONE"
LABEL @"WNEKI PŁUCNE" (_|) (ar"nie(powiększone|poszerzone)" | a"w" a"normie") AS @"WNEKI PŁUCNE
NIEPOWIĘKSZONE"
LABEL @"WNEKI PŁUCNE" (_|) <0,2> (a"prawdopodnie"|) a"naczyniowe" AS @"WNEKI PŁUCNE NACZYNIOWE"
```

5.4 Results

5.4.1 Heart

The results of the heart model are shown in the table **5.2**. As expected, the most dominant feature was "Normal heart size". Overall the model was able to accurately extract 89% of features related to heart resulting in precision of 89% with respect to this word.

Table **5.2**: Results of the heart MTT model

Tag (pl)	Tag (en)	Count
Serce w normie	Normal heart size	1136
Serce powiększone	the heart is enlarged	300
Serce nie do oceny	heart borders difficult to distinguish	67
Serce podparte na przeponie	excessively horizontal position of the heart	47
Serce (other)	heart (other)	184

5.4.2 Lungs

The results of the lungs model are shown in the table **5.3**. The model was able to accurately extract 60% of features related to "lungs" resulting in precision of 60% with respect to this word. This score, appearing to be low, is however a direct result of the reports' complexity and number of rare anomalies, rather than variations in the natural language. The model also correctly labeled 92% of the features related to "aorta" and "82%" of the features related to "Wnęki płucne".

Table **5.3**: Results of the lungs MTT model

Tag (pl)	Tag (en)
Płuca prawidłowo powietrzne	normal aeration of lungs
Lewe płuco prawidłowo powietrzne	left lung normal
Prawe płuco prawidłowo powietrzne	right lung normal
Płuca bez cech zastoju	lungs with no pulmonary congestion
Płuca bez zmian ogniskowych	lungs with no focal lesions
Płuca bez zmian zapalnych	lungs with no pneumonic consolidation Płuca bez zmian mi
34	
Płuca bez zagęszczeń miąższowych	lungs with no pulmonary opacities
Płuca rozedmowe	emphysematous lungs
Płuca rozprężone	decompressed lungs
Zmniejszona powietrzność płuca	decreased general aeration of the lung
Płuca (inne)	lungs (other)
Aorta wydłużona	elongated aorta
Aorta poszerzona	dilated aorta
Aorta zstępująca	descending aorta
Aorta kręta	tortuous aorta
Aorta miażdżycowa	atherosclerotic aorta
Aorta (inne)	aorta (other)
Wnęki płucne powiększone	dilated pulmonary hilum
Wnęki płucne naczyniowe	vascular pulmonary hilum
Wnęki płucne niepowiększone	normal hila
Wnęki płucne (inne)	hilum (other)
Płyn w prawej jamie opłucnej	the right pleural effusion
Płyn w lewej jamie opłucnowej	the left pleural effusion
Płyn w obu jamach opłucnych	bilateral pleural effusion
Lewa jama opłucnowa wolna	clear left pleural cavity
Prawa jama opłucnowa wolna	clear right pleural cavity
Jamy opłucnowe wolne od płynu	both pleural cavities clear
Brak odmy	no pneumotorax
Zręb	lung interstitial
Zmiany zapalne	pneumonic consolidation
Miąższ bez zagęszczeń	lung parenthymal without densities
Zagęszczenia zapalne	pneumonic opacity
Zmiany siateczkowe	reticular opacities

Chapter 6

COVID-19 classification based on X-ray images

6.1 Introduction

In this chapter we take on the problem of diagnosing pneumonia cases including COVID-19 viral infections based on X-ray chest scans. Currently, formal diagnosis of COVID-19 requires a laboratory test (RT-PCR) that requires specialized equipment and takes at least 24 hours to produce a result [1]. The number of PCR tests is also often limited and those tests have to be distributed selectively; however, X-ray scans can be performed on the spot and the scanners are common equipment of most clinics. A machine learning system being able to accurately predict pneumonia infection related to COVID-19 could serve as a pre-diagnostic system to label the cases with high probability of COVID-19 infection, resulting in better distribution of limited PCR tests. Moreover, X-ray imaging is often performed for reasons unrelated to COVID-19. Hospitals equipped with such systems could also utilize machine learning algorithms to diagnose COVID-19 in addition to the procedures related to the primary reason of

hospital admission. As those systems, by their definition require no human assistance this could be done with little to none additional time costs for the clinics staff. In our work we describe a number of popular neural networks architectures used for image recognition tasks and evaluate their performance in COVID-19 diagnosis based on chest X-ray images. We also test the transfer learning and multi task learning capabilities of the networks and their effect on the system's accuracy.

6.2 Dataset

In our experiments we use Curated Dataset for COVID-19 [58] that is a collection of 15 publicly available datasets. This dataset contains positive and negative COVID-19 cases as well as samples from other bacterial and viral pneumonia infection. This allows us to test the multi-task learning abilities of the network i.e. if introducing additional pneumonia classes to the classifier will improve its preference on the end task of recognizing COVID-19 cases. The distribution of classes in the dataset is shown in the table **6.4**.

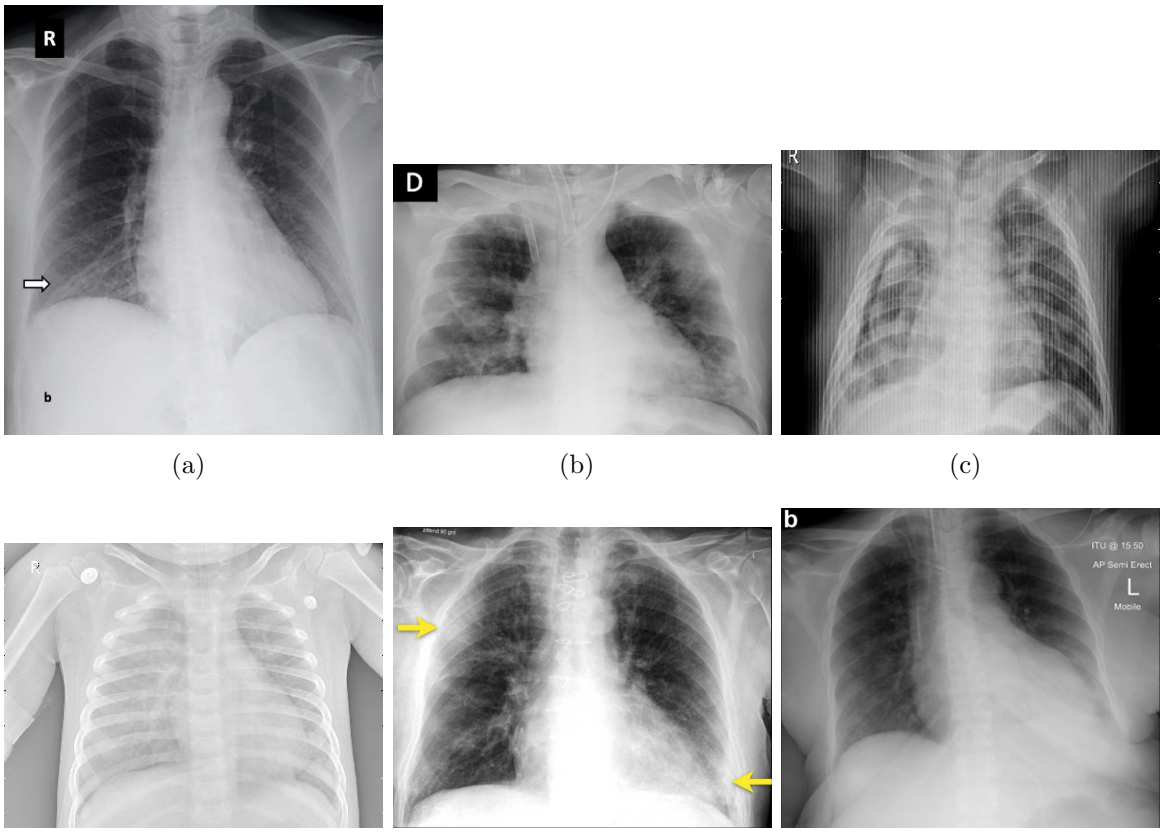
Table **6.1**: Class distribution over chest X-ray dataset

Class	Count
Normal	3270
Pneumonia-Bacterial	3001
Pneumonia-Viral	1656
COVID-19	1281
Total	9208

Sample images from the dataset are shown in the figure 6.1. The standard for storing medical images is DICOM, unfortunately the images in these datasets are in PNG format instead, so there is a variation in the contrast of images (6.1 (d)) depending on the preset configuration. This carries the risk of sub-optimal network performance as different images follow different grey-scale-level distribution. Moreover, a substantial

amount of images is annotated with markers such as arrows (6.1 (a, e)) of different colors. This again is a flaw of using PNG format as in DICOM format, such annotations can be stored in different layers, but in PNG there is no simple way to distinguish the image and annotation. Moreover, the annotations are placed across different parts of images, so they can not be removed without affecting the structure of the image. Other artefacts also include labels representing side of the body (letter "R" on images 6.1 (a, c)) and information from the device (6.1 (f)). The images also come in different sizes and crops what can also be a challenge in processing them.

In our experiments we set aside 10% of the data and saved it for testing purposes while the other 90% was used in training and development of the networks.



(a) (b) (c)

(d) (e) (f)

Figure 6.1: Sample images from the chest X-ray dataset presenting various flaws, see descriptions in the text.

6.3 Convolutional Neural Networks

In traditional (fully-connected) networks, every neuron of the input layer is connected to every neuron of the output layer. Each of these connections has its own weights that are adjusted through back-propagation of the model's error. The network's role is to learn these weights to distinguish important features and their combinations, resulting in solving the given task. The learning is done by observing samples and labels in the training dataset and learning the data transformations necessary to solve the task. The model is then tested on the new data, which it has not seen before, to test its generalization abilities and the fitness of the transformations it has learned.

Convolutional Neural Networks (CNNs) [59] are a type of neural networks specialized in processing data that has a known grid-like topology [60] e.g. X-ray images. As opposed to the fully-connected networks, CNNs rely on the local-connectivity (or sparse-connectivity [60]) principle, meaning that certain neurons in the output layer are only connected to certain neurons in the input layer inside their perception fields. This can be seen as the network "looking" only at certain parts of the image at a time. This modification introduces several improvements over fully connected layers, such as reduced number of parameters, which both reduces the memory requirements of the model and improves its statistical efficiency [60]. CNNs use convolution (or often cross-correlation) operation to process the data. This operation can be seen as applying filters to the image to extract important features from it. Example of such filter applied to an X-ray image is shown in the figure 6.2. The difference between classical image processing and CNNs is that the filter's (kernel's) parameters are not pre-defined and fixed but are instead learned, like the weights in traditional neural networks. Cross-correlation (convolution without flipping the kernel) is shown in the

equation 6.1.

$$S(i,j) = (K \times I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{6.1}$$

The values of the input data (I) in the range of receptive field (of size m × n) are multiplied by the values of the kernel (K) and then summed together. Illustrated example of the convolution operation is shown in the figure 6.3.

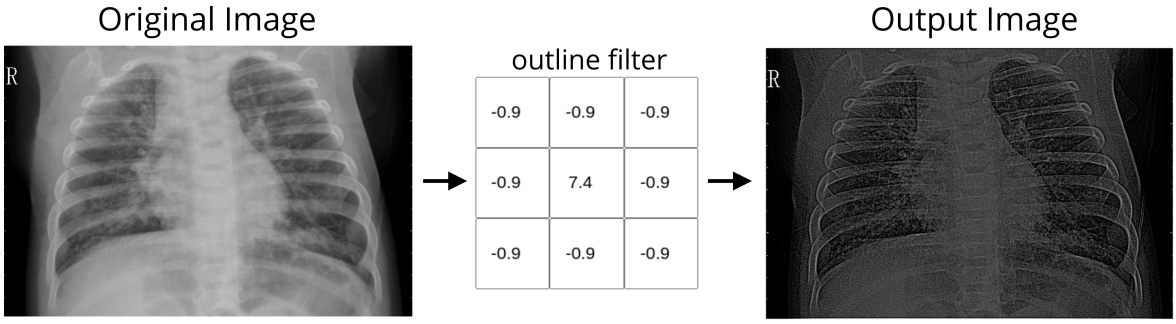


Figure 6.2: Outline filter applied to the X-ray chest image. The pixels in the output image are brighter if their corresponding pixels in the input image are brighter and pixels (7.4) next to them are darker (-0.9) - hence the detection of pixels that represent the edges.

The operation can be seen as a weighted sum of the input data values inside the perception field. In case of X-ray images these input values represent the amount of variation in X-ray exposure between points within the image [61] or, in case of the processed image, simply the pixel values in range (0-255) of the gray-scale image. The weights are the parameters of the kernel (filter) that are adjusted through back-propagation during the training of the network. The network then learns to select weights that result in detection of important features (such as certain shapes and patterns) that are useful in processing the image. In case of classification, the convolutional layers act as feature extractors that aim to extract important information from the image based on which the classification is performed. Given this objective, the network should promote weights that extract features that are most relevant for the classification (such

as presence of infection inside lungs).

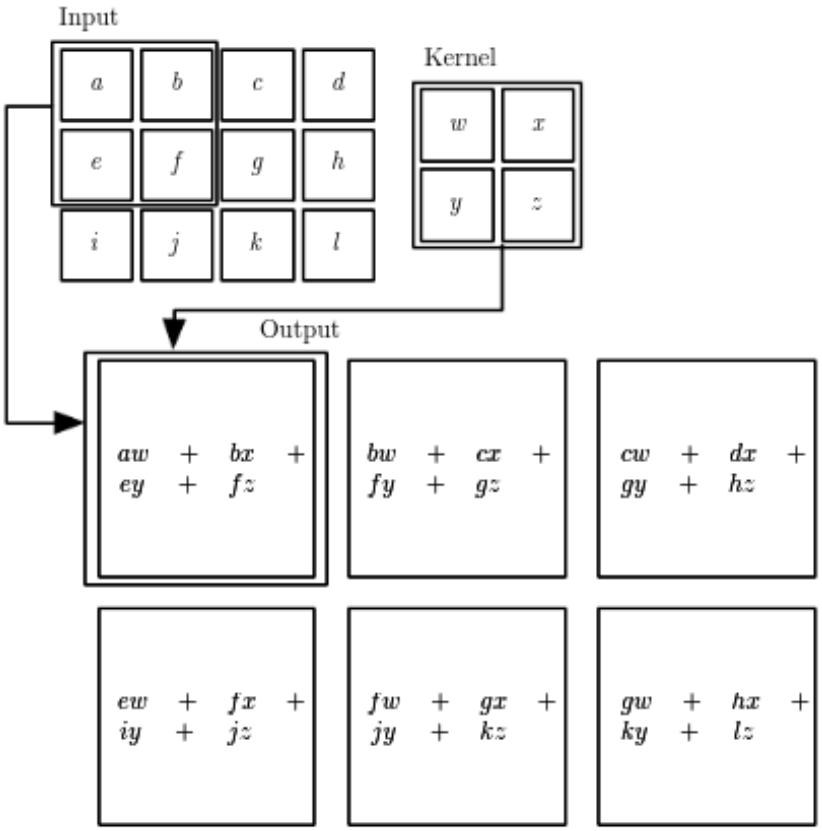


Figure 6.3: An example of 2D convolution [60]. The output data is a result of the weighted sum of the input data (a, b, c..., l) inside the perception field (in this case 2×2). The kernel represents the weights (w, x, y, z) that are learned during training of the network.

One convolutional layer usually has multiple kernels that process the input data. Output of each kernel is then treated as a separate channel analysing different feature. For example, one kernel can be responsible for detecting horizontal lines, while the other can be responsible for detecting vertical ones. The channels are grouped together, resulting in creation of a 3D tensor of shape (image length, image width, number of channels). Kernels in next layer then take data from all channels as input.

CNNs are typically constructed in such fashion that the spacial dimensions of the

image decrease in the later layers while the number of channels increases. This is motivated by the fact that deeper layers should analyse more general and abstract features, hence the decreased spacial size representing more global approach. The increased number of channels represents increased number of features analysed by the network. In the classification task, the convolutional part of the network works as a feature extractor that converts the input image into maps of features. Those maps are then converted into 1D representation by flattening layers or mechanisms such as global average pooling. This representation can then be used by the classifier part of the network made of fully-connected layers that map the encoded image into its respective class. This architecture is visualized in the figure 6.4.

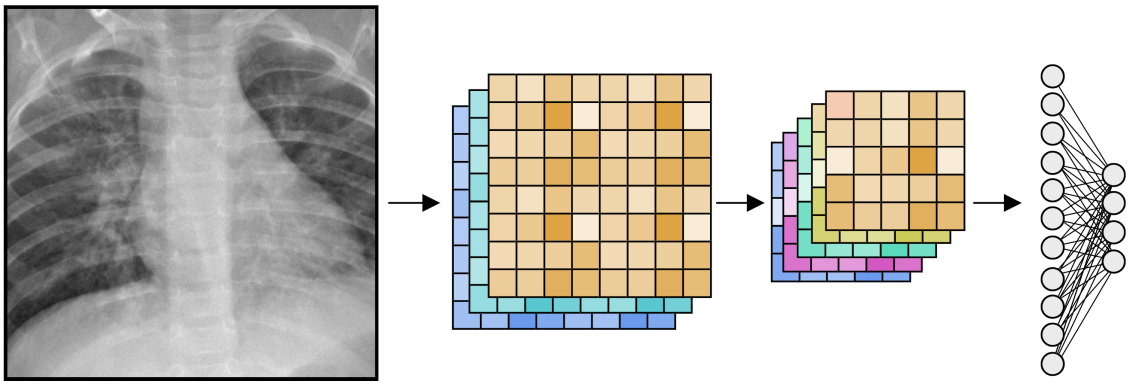


Figure 6.4: Visualization of typical CNN architecture.

In our work we have tested various neural network architectures described in next section.

6.4 Architectures

6.4.1 VGG 16

VGG16 network was introduced in paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" [62]. The authors introduced a structural design of the network using only 3×3 convolutions (the smallest size to capture notion of left/right up/down and center [62]). The usage of small kernel size allowed the network to have increased depth in comparison with its predecessors. Previous approaches used larger receptive fields (such as 11×11 or 7×7) that amounted to greater number of weights. Using multiple 3×3 convolutions instead of single 7×7 one resulted in decreased number of parameters and acted as regularization scheme where the convolutions block was forced to decompose into 3×3 convolution while having the receptive field similar to 7×7 convolution. This also enabled the authors to build deeper network without increasing the number of parameters, what was an important task given the previous findings that deeper networks perform better on image recognition task [63] but the large number of training parameters can result in network outfitting.

Used filters preserve the spatial dimensions of the image by utilizing strides fixed to one pixel and adjusting the padding. The image size is reduced through the network by 2×2 max-pooling layers. Those pooling layers decrease the image size by a factor of 2, while the number of filters is increased twice between blocks. This results in fixed number of parameters throughout the network while also following the pattern of increased generalization and abstraction through the network.

VGG16 was able to achieve state-of-the art performance image recognition tasks while also showing promising results in the medical domain.

In our experiments we use the feature-extractor part of the network, shown in figure 6.5 while incorporating our own classification layers that are shared among different

models and described in greater detail in the section 6.5.

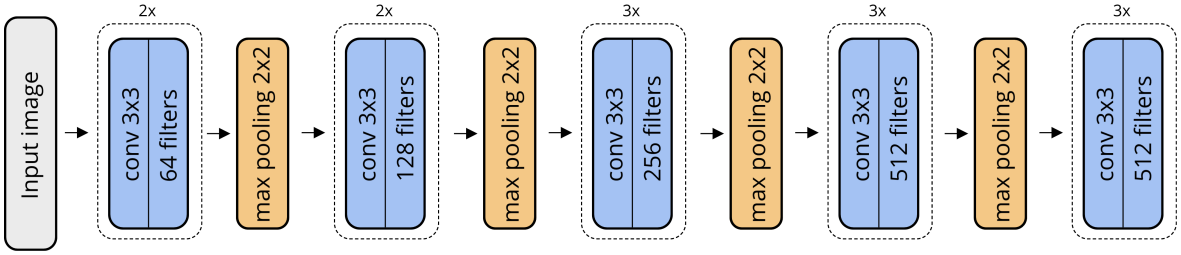


Figure 6.5: VGG16

6.4.2 Inception V3

Inception V3 network was introduced in the paper "Rethinking the Inception Architecture for Computer Vision" [64].

It was built upon the principle of Inception Modules (shown in the figure 6.6). Those modules consisted of multiple filters with different receptive fields analysing the same input data in parallel. This is motivated by the fact that filters of smaller size analyze local features in the image while ones with greater spatial size analyze more global dependencies. Output of those filters is then concatenated into one feature map representing both global and local features of the analysed image.

Inception V3 architecture introduced additional novelties to the inception architecture such as factorizing larger convolutions into multiple 3×3 convolutions similarly to VGG architecture (shown in Inception module 1 in the figure 6.6) or spatial factorization into asymmetric convolutions (visible in Inception module 2 and 3 in the figure 6.6). Asymmetric convolutions factorization consisted of replacing one $n \times n$ convolution by two convolutions of sizes $1 \times n$ and $n \times 1$. This factorization reduces the number of parameters needed (e.g. by 33% in the 3×3 convolutions) while preserving the general receptive field of the block. The whole architecture of Inception V3 network is shown in the figure 6.6.

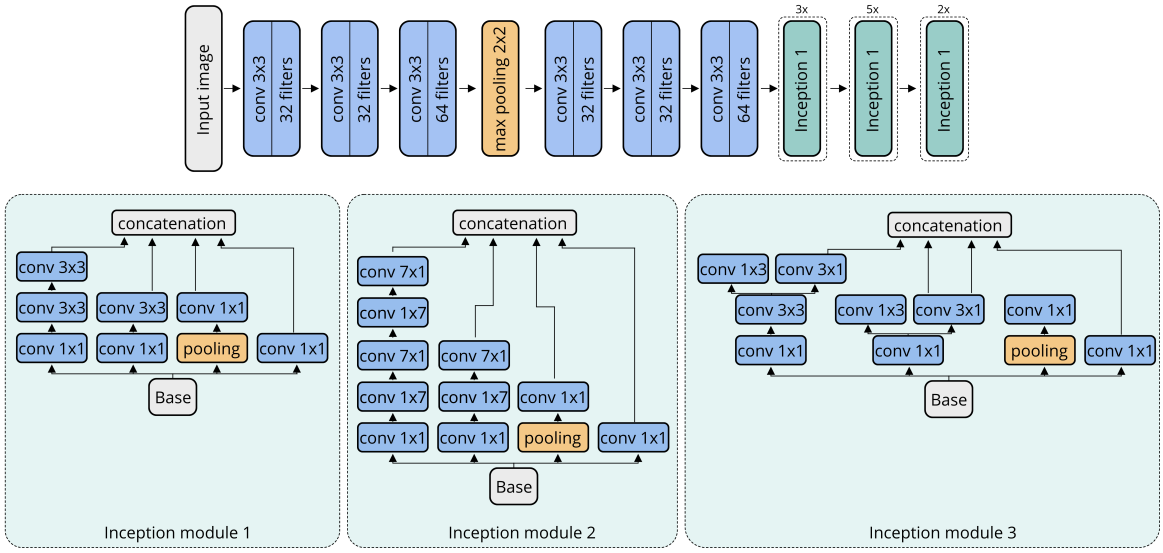


Figure 6.6: Inception V3

6.4.3 ResNetV2

ResNetV2 was introduced in paper "Identity Mappings in Deep Residual Networks" [65]. It was based on the concept of residual connections (or identity shortcut connection) introduced earlier in the Inception architecture [66]. Residual connections are shown in the residual module in the figure 6.7. Those connections skip one or more layers performing identity mapping of the input instead resulting in the input being unchanged. This unchanged input is then added to the output of the convectional block. Identity shortcut connections were developed as a mean to solve the degradation problem in deep neural networks. Deeper neural networks should produce no higher training error than shallow ones. This degradation problem - lowered training accuracy in deeper neural network is a sign of optimization difficulties as solvers at the time were unable to find optimal solutions for larger networks. The introduced residual mapping is easier to optimize than the original, referenced mapping and should therefore enable the deeper networks to produce at least as good results as its shallow counterparts.

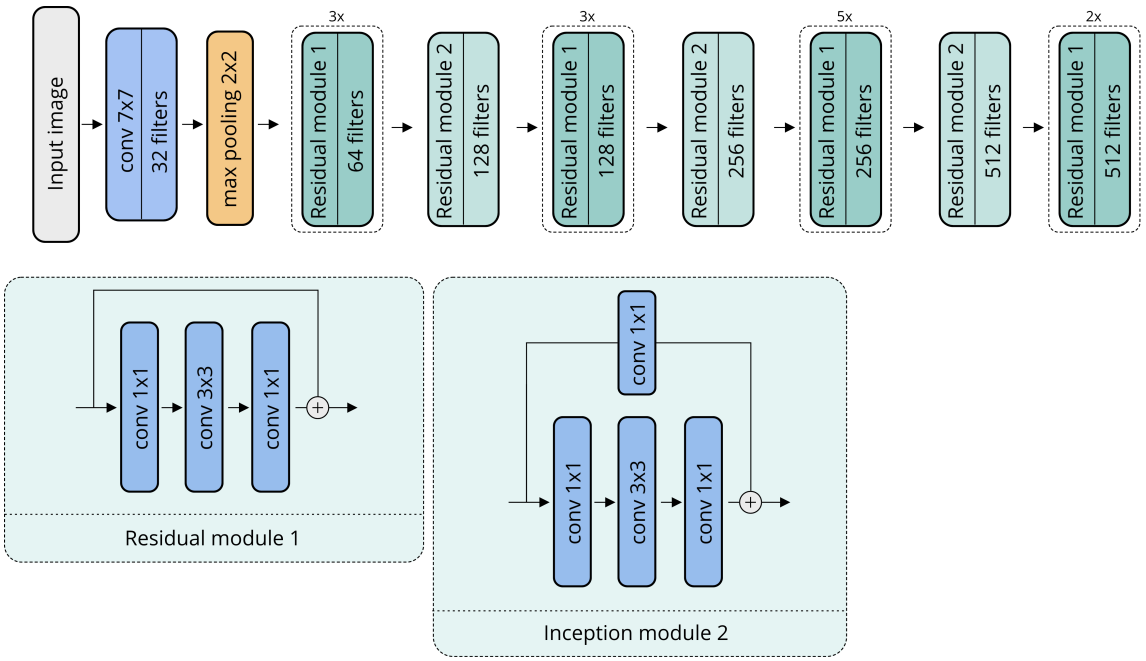


Figure 6.7: ResNetV2 architecture

6.4.4 DenseNet

DenseNet was introduced in the paper "Densely Connected Convolutional Networks" [67]. This network is built upon the principle of connecting all layers inside the dense block (shown in the figure 6.8) directly to each other. Each layer obtains additional input from all previous layers. This is done by concatenating the feature maps rather than summing them as in ResNet. This is done to ensure maximum information flow between the layers.

The usage of dense connections results in lower number of parameters required as well as improved flow of information and gradients through the network. This enables the network to be larger without overfitting by the effect of implicit deep supervision [68]. Moreover, dense connections also make the network easier to train, especially on small datasets, as dense connections act as regularization method.

The whole architecture of Inception V3 network is shown in the figure 6.8

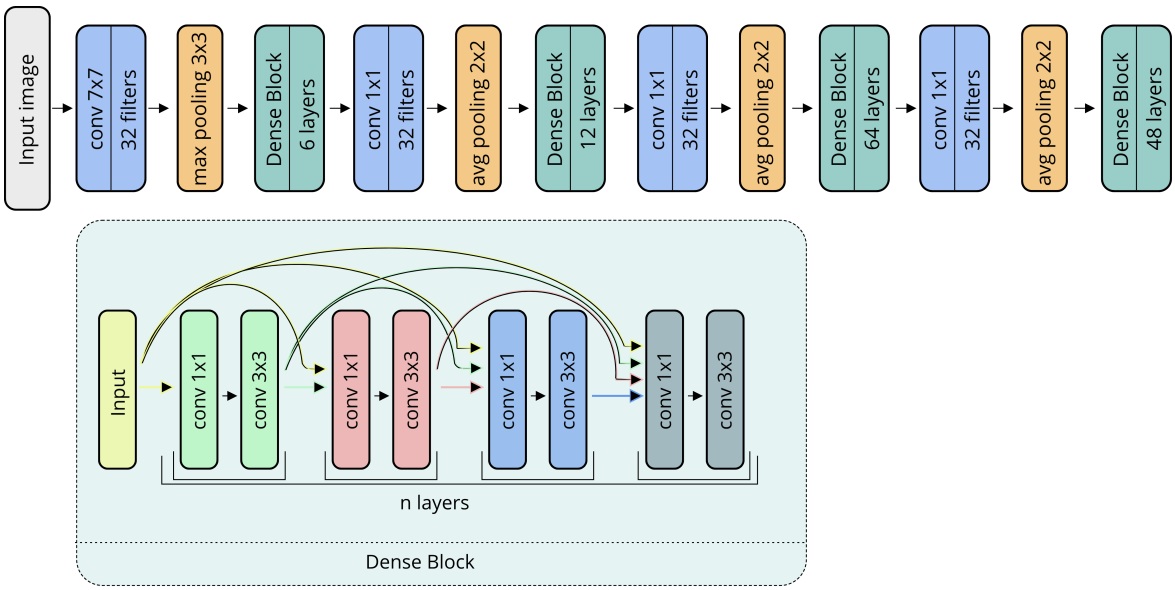


Figure 6.8: DenseNet architecture

6.4.5 CheXNet and transfer learning

CheXNet is a neural network based on DenseNet architecture developed for pneumonia classification. It is trained using the ChestX-ray14 dataset containing 112,120 frontal-view X-ray images divided into 14 thoracic pathology labels. In our work we use pre-trained feature extractor part of the network and test its performance on our end task.

Typically deep convolutional networks are pre-trained on a larger dataset before fine-tuning them for the target task to facilitate transfer learning. The convolutional part of the networks called feature-extractor is then shared between tasks while the classification layers are task-specific. The whole network is firstly trained on the source task, then the classification layers are stripped and replaced with new ones for the target task. This process is illustrated in the figure 6.9. Transfer learning holds several advantages over training the network from scratch. Usage of larger dataset in the source task makes it easier to train on the limited amount of data in the target task as the network learns generalized data representations in the pre-training process. More-

over, pre-training is able to limit the overftiing of networks and improve their general accuracy.

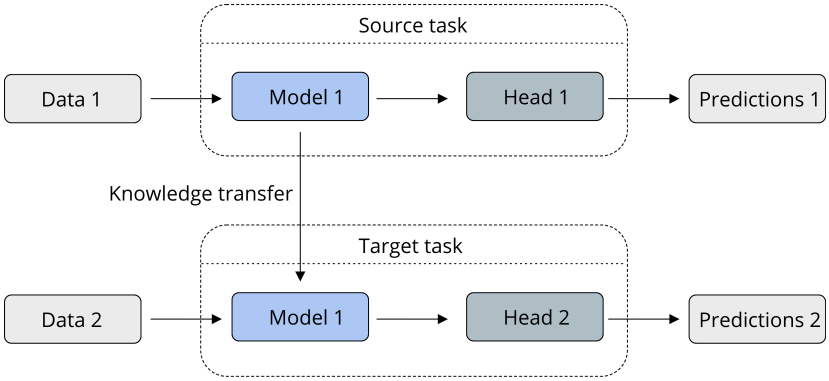


Figure 6.9: Transfer Learning

Most widely used source dataset for the pre-training task is ImageNet [69]. However, this dataset contains a set of natural images that is vastly different from medical domains such as X-ray images. The main difference is that natural images are colorful and encoded in RBG format while medical images such as X-ray or CT scans are artificially constructed based on physical measurements and are therefore encoded in gray scale format. Moreover, there is also substantial difference between the types of objects present on the natural images and medical scans. Those differences make the transfer task more challenging and may therefore result in sub-optimal knowledge transfer between tasks and lower classification accuracy of the end solution. Network pre-trained on a larger source dataset of the same modality as the target task should improve that knowledge transfer process. Motivated by this fact in our experiments we use both versions of DenseNet networks - the one pre-trained on ImageNet (labeled DenseNet) and the one pre-trained ChestX-ray14 dataset (labeled CheXNet)

6.4.6 Classification layers

In our experiments we have tested different feature-extractor models based on the above-mentioned architectures. Each of those architectures was followed by the same classification layers shown in the figure 6.10. The output of feature-extractor part of the network was pooled using global average pooling method to convert the three-dimensional into a one-dimensional feature vector suitable for the classification task. This layer was followed by a dropout [70] layer with the dropout coefficient of 0.2, and then batch normalization layer. The dropout is used to minimalize the co-adaptation of neurons and to prevent it from over-fitting. Next, we used 2 hidden fully-connected layers of 2096 neurons, and with ReLU activation function, each also followed by a dropout layer. In those layers we have also used Elastic Net regularization [71]. Next, we used the final classification fully-connected layer that was responsible for the final prediction. In this layer we used softmax activation functions as each of images belonged to exactly one class.

For optimization of the networks we have used RMSprop algorithm and categorical cross-entropy was selected as the loss function.

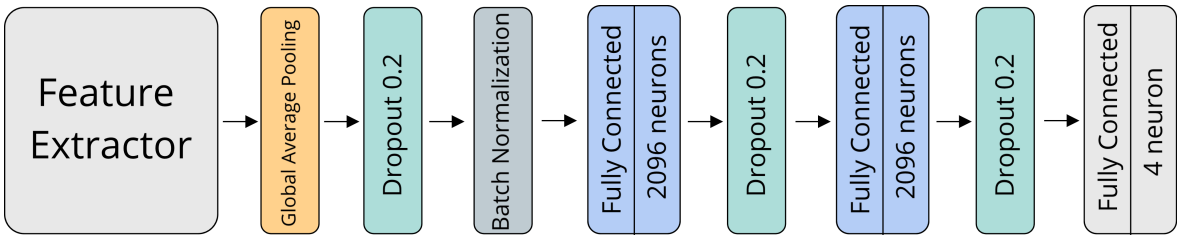


Figure 6.10: Classification model

6.5 Experiments And Results

In each one of the experiments the best epoch and parameters were chosen based on the validation loss over 10-fold train data split. The models' performance was then tested on the test dataset not used during the development process. In all the feature extractors except for CheXNet the weights were initialized based on the ImageNet pre-training. In the experiments we allowed fine tuning of the feature-extractor part of the network as our initial experiments have shown freezing the layers results in sub-optimal performance of the classifiers.

6.5.1 Three types of pneumonia classification

In this set of experiments we tested the above mentioned architectures on the 4-class prediction task (3 types of pneumonia infection and normal lungs). The results are shown in the table **6.2**. The best architecture turned out to be the one based on

Table **6.2**: Models comparison on 4-class prediction

Architecture	F1
DenseNet	0.748
Inception V3	0.772
ResNetV2	0.785
VGG16	0.856

VGG16 feature extractor, achieving 86 % F1 score in the 4-class prediction challenge. This result is most likely motivated by the fact that VGG-16 is relatively shallow compared to other tested architectures, therefore it is easier to fine-tune for the new data domain.

6.5.2 Transfer Learning

In this experiment we compared the performance of Dense Net network pre-trained on the ImageNet dataset and the one pre-tranied on chest X-ray data (labeled CheXNet) on the 4-class prediction task. The goal of the experiment was to test if pre-traning the network on larger dataset form the same domain i. e. chest X-ray results in better performance over pre-training on general dataset. Results are shown in the table **6.3**. As can be seen, CheXNet architecture achieved 0.83 F1 score what results in more

Table **6.3**: Results

Architecture	F1
DenseNet (ImageNet)	0.748
CheXNet	0.826

than 8 percentage point improvement over DenseNet. This proves that larger chest X-ray datasets can be leveraged to achieve optimal solutions in COVID-19 prediction challenge.

6.5.3 Mulit-task Learning

In this set of experiments we have tested if training the networks in 4-class prediction task improves their performance on the binary classification task of recognizing positive and negative COVID-19 cases. The training was done similarly to the experiments mentioned in sections above, however during the testing phase the networks output was pooled into binary "COVID-19", "Not COVID-19" classes. The results appear to be inconclusive with regard to the effect of providing additional labels in the training procedure. Although in some cases the effect is slightly positive (VGG16, CheXNet) in the majority of the cases (Inception V3, ResNet, DenseNet) training the network on only two classes improves the result by up to 6 percentage points.

Table **6.4**: Models comparison on COVID-19 vs Not COVID-19 prediction

Architecture	F1 (4 class training)	F1 (2 class traning)
Inception V3	0.906	**0.968**
DenseNet	0.923	0.93
CheXNet	0.935	0.93
ResNetV2	0.925	0.956
VGG16	0.965	0.961

6.5.4 Final Solution

For the final solution, based on the experiments, we chose Inception V3 model trained on binary prediction task. The results of the model are shown in tables **6.5** and **6.6**. Table **6.5** presents the confusion matrix and table **6.6** presents detailed metrics of the model.

Table **6.5**: Confusion matrix of the final model

		True diagnosis		
		Positive	Negative	Total
Model output	Positive	123	8	128
	Negative	5	785	190
	Total	128	793	

Table **6.6**: Results of the final model

Metric	Score
f1	0.968
precision	0.976
recall	0.960

Although the results look very promising we have to remain sceptical, mainly because of the quality of data used for training and evaluation (see section 6.2). In our future work we are planning to test the resulting models on real-life data stored in DICOM format without any artefacts.

Chapter 7

COVID-19 classification based on complete blood count (CBC)

7.1 Introduction

Complete blood count (CBC) is a standard test frequently performed during the admission to hospital or regular health exams. In this chapter we explore the possibility of using CBC data in prediction of COVID-19 viral infection. Such model, having high enough accuracy would enable hospitals and clinics to accurately label patients having high risk of COVID-19 infection. This is especially important as the number of COVID-19 tests is limited and they often need to be distributed selectively. CBC exams are, however, much cheaper and are already a standard procedure. Therefore machine learning model based on CBC data could serve as pre-selection mechanism for more expensive and specific tests.

7.2 Pre-processing

From the initial dataset of 30364 records we have selected patients who had COVID-19 test done in the span of one day from the CBC test. This reduced the dataset to 19559 samples with 19013 negative COVID-19 results and 546 positive results (table **7.1**).

Table **7.1**: Class distribution over dataset

Diagnosis	Count
Negative	19013
Positive	546

As the dataset was largely imbalanced, we used data balancing techniques described in the following section. Before feeding the data to the classification algorithm we have also normalized the data, accordingly to the distribution of each feature. Next, based on the correlation coefficients (shown in table **7.2**), we have discarded highly correlated features as a mean to clean the data and reduce its dimensionality.

Table **7.2**: Correlation between CBC features (for values greater than 0.5)

feature 1	feature 2	correlation coefficient
Hemoglobin	Hematocrit	0.972
% of Neutrophils	% of Lymphocytes	0.947
Red blood cell count	Hematocrit	0.919
Hemoglobin	Red blood cell count	0.888
White Blood Cell Count	Neutrophil Count	0.546
Hemoglobin	Red Cell Distribution Width	0.506

This lead to creation of the final dataset conditioning 13 CBC features shown in table **7.3**.

Table **7.3**: CBC features used in classification

Attributes
Red blood cell count
Mean Corpuscular Hemoglobin
Mean Corpuscular Hemoglobin Concentration
Red Cell Distribution Width
Platelet Count
Mean Platelet Volume
White Blood Cell Count
Immunoglobulins
% of Lymphocytes
% of Monocytes
% of Eosinophils
% of Basocytes
% of Erythrocytoblasts

7.3 Data balancing(re-sampling)

Imbalanced dataset can often lead to biased predictions as the model can obtain high accuracy being biased towards the majority class. This is often called the accuracy paradox [72]. To combat this problem in our work we have used re-sampling techniques to balance the datasets. In the initial experiments we have tested a number of under-sampling and oversampling techniques. Based on the empirical experiments we decided to use AllKNN alghoritm for undersampling and SMOTE algorithm for oversampling. Those algorithms are described in the following sections.

7.3.1 Undersampling – AllKNN

Data undersampling is a set of techniques used to reduce the number of examples from majority class in imbalanced datasets. In our work we use AllKNN alghoritm [73] that removes all examples from the dataset that were classified incorrectly.

7.3.2 Oversampling – SMOTE

Data oversampling is a set of techniques used to increase the distribution of data from minority class. This can also be viewed as an augmentation method. Classic oversampling methods are based on duplicating the data. Those methods help to balance the dataset, however they do not add any new information to the model. SMOTE algorithm [74] creates new synthetic data utilizing a k-nearest neighbor algorithm. The synthetic data created by choosing one of the k nearest neighbors at random and connecting them to form a line segment in the feature space. New data is generated as a convex combination of the two chosen instances a and b [74]. This method causes the classifier to build larger decision regions that contain nearby minority class points [74].

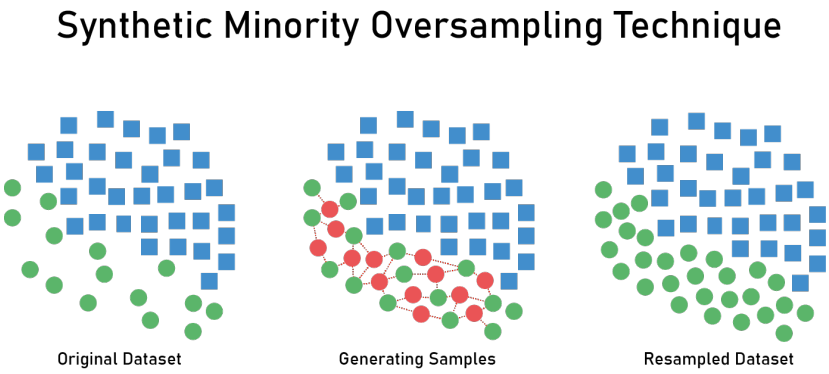


Figure 7.1: SMOTE data generation visualization [75]

7.4 Classification model

For the classification we have used a neural network architecture shown in the figure 7.2. It consisted of 2 hidden layers with 200 neurons and ReLU activation function as well as a final classification layer with one neuron and sigmoid activation function representing the positive or negative result of COVID-19 test. In every layer we have

used elastic net regularization [71]. Each of the hidden layers was also followed by the dropout [70] layer with coefficient equal to 0.5. Such aggressive dropout was used to prevent the network from overfitting. As the task is binary in nature we have used binary-crossentropy as the loss function. We have also used Adam algorithm [76] for the optimizer.

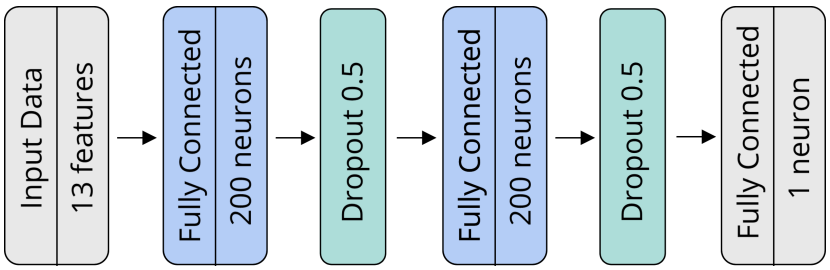


Figure 7.2: Classification model

7.5 Experiments and results

For our experiments we randomly selected 20% of the data preserving the original distribution for the test dataset and used the rest for training the network. As a primary metric we chose AUC based on Receiver Operator Characteristic (ROC) as it is not biased towards the majority or minority class [77] making it a suitable measurement of the classifier's performance. Additionally, for the evaluation we have also tested the models accuracy. Results of our best model are shown in the table **7.4**.

Table **7.4**: Results

Accuracy	97 %
AUC	74 %

The results show our model was able achieve 97 % accuracy on the classification task while still achieving relatively high (given the class imbalance and number of samples) AUC score.

Chapter 8

User interface/application

8.1 Communication

The server communicates with the User Interface using Representational State Transfer (REST) that uses well-defined HTTP requests and responses, in which the data is transferred in lightweight, human-readable JavaScript Object Notation (JSON) [78]. Such objects can have multiple name(key)/value pairs, utilize commas to separate data, curly braces to store objects and square brackets to store arrays. It often is represented as a string literal, which is similar to what is shown in listing 8.1.

Listing 8.1: Example JSON object.

```
{
    "student": {
        "name": "John Smith",
        "subjects": ["physics", "history"]
    }
}
```

Although it is native to JavaScript, multiple programming languages implement ways to work with JSON - it is easily translatable to Python's native dictionary, for example with the usage of json package contained in the standard library.

REST utilises well defined Hypertext Transfer Protocol (HTTP) for transmitting data. We are only using some of its methods, namely GET for retrieving data and POST for submitting data. This approach also enables us to use HTTP response codes, which are grouped in five classes, shown in table **8.1**.

Table **8.1**: HTTP response codes classes.

Class	Codes
Informational responses	100–199
Successful responses	200–299
Redirects	300–399
Client errors	400–499
Server errors	500–599

The most important and widely used codes are presented in table **8.2**:

Table **8.2**: Most used HTTP codes

Response	Code
OK	200
Created	201
Bad Request	400
Unauthorized	401
Forbidden	403
Internal Server Error	500

8.2 Server

We decided to organize separate parts of logic in separate directories, as listed below:

- blueprints - stores all blueprints (resources) of the application,

Listing 8.2: Example attribute representation in Python code

```
attribute = {
    'en': {
        'code': 'RBC',
        'name': 'Red blood cell count'
    },
    'maximum_norm': 4.8,
    'minimum_norm': 3.8,
    'name': 'Red blood cell count',
    'pl': {
        'code': 'RBC',
        'name': 'Krwinki czerwone'
    },
    'unit': '10<sup>12</sup>/l'
}
```

- db_models - stores all database mappings,
- ml_models - stores machine learning models we load at the application start.

8.2.1 Routing

Our server provides two endpoints, one is the authorization route for logging in (POST) and the other implements two HTTP methods, which thanks to flask-restful are just two methods defined in one class that inherits from "flask_restful.Resource". These methods are:

- GET - used to inform the front-end about CBC parameters that it needs to get from the user, which are stored as a list of dictionaries, where one attribute is represented as shown in listing 8.2,

  This data format allows for easy translation to JSON, especially since there are only 13 attributes needed.

- POST - gets parameters from the user who submitted the form, then parses

and translates them to numpy array that can be used to get the prediction. Saves usage statistics in database and responds with the results in a form of a dictionary that gets translates to JSON, depicted in listing 8.3.

Listing 8.3: Example server response with prediction results

```
{
    "covid_probability":"0.8710010165053534",
    "healthy_probability":"0.419899898349464",
    "prediction":"covid"
}
```

8.2.2 Database structure

Our database consists of two tables, relating to each other by user_id. One stores users and the other predictions, as shown in figure 8.1.

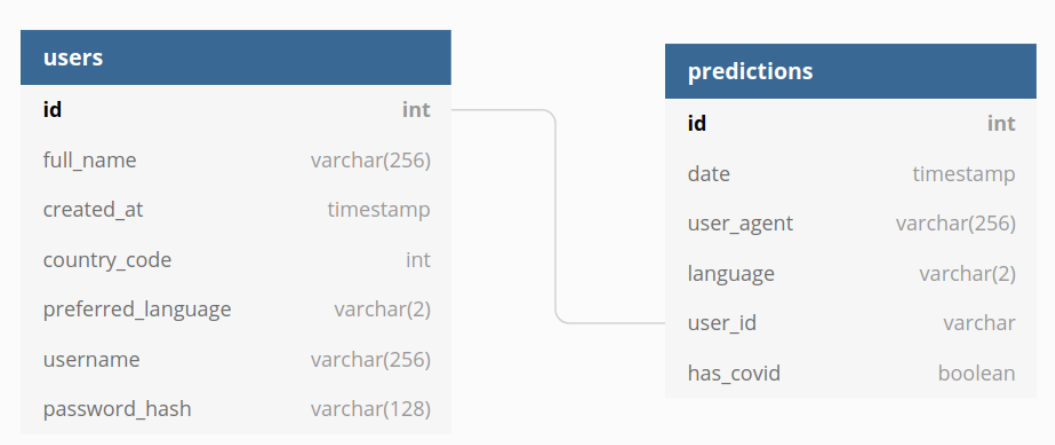


Figure 8.1: Database scheme

It can be easily extended with flask-migrate, but it is beyond the scope of our work.

8.2.3 Security

For identifying user's identity, we used JSON Web Tokens (JWT) [79] that serve as identifiers of the user's data and have a form of secret keys, that can only be decoded with enough information about them. Every request other than the one to log in must be done using a proper and fresh token, which ensures user's identity and allows to serve user related data if needed. As shown in figure 8.1, the users table contains a password hash that is being generated by bcrypt with 12 rounds, which determines the complexity of the salt.

The main disadvantage of the token approach is that the token can be stolen in a man in the middle attack, but such attack is hard to perform as the hacker would need access to the user's network traffic, which is most commonly secured by Secure Socket Layers (SSL). Tokens also have a really short lifespan and can be blacklisted (logged out), so the risk is minimal. Even if someone could impersonate an user, the only way sensitive data flows is to the server and not the other way around. Moreover, we do not store any medical records as they are used only for training that is performed on a separate, internal environment.

8.2.4 Configuration

We mostly rely on flask's out of the box settings, so we only needed to tweak some parameters, especially for security reasons, which we did using a special "Config" class. The most important thing was configuring JWT, applying a secret key and enabling tokens blacklisting. SQLAlchemy requires a database URI and along that we provided a boolean stating if we wanted to track modifications. We also set logging levels in there to generate a substantial amount of logs, so we can react fast in the event of any unforeseen problems.

8.3 Front-end

We decided to organize separate parts of front-end logic/viewing in separate directories, as shown below:

- public – stores all public data, such as "manifest.json" and "favicon.ico",
- assets – stores all images used in the application,
- components – stores all components, each in their own, separate ".tsx" file,
- styles – stores all components' stylesheets, each in their own separate ".scss" file that is bundled in "index.scss" using "@import" rule, to be made into "index.css" by the SASS pre-processor,
- tests – stores everything related to tests,
- types – stores type definitions.

8.3.1 Routing

The application has three pages that can be viewed.

- Login page that allows to log in with user's identifier and password shown in figure 8.2.

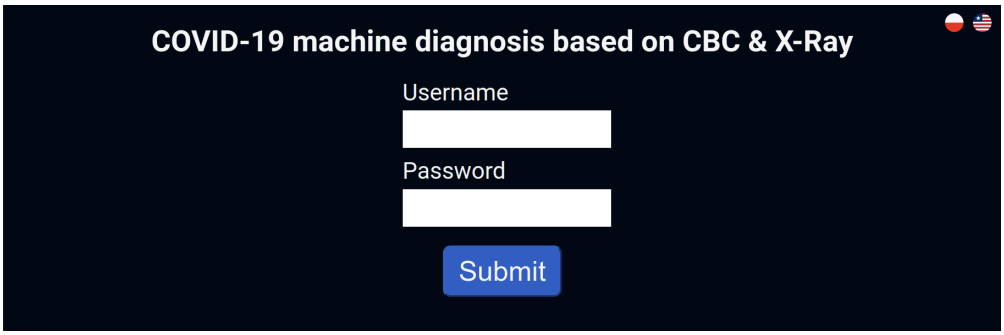


Figure 8.2: Login page form

- Prediction page that allows to get a prediction either with CBC results or X-Ray scan, shown in figures 8.3 and 8.4 accordingly.

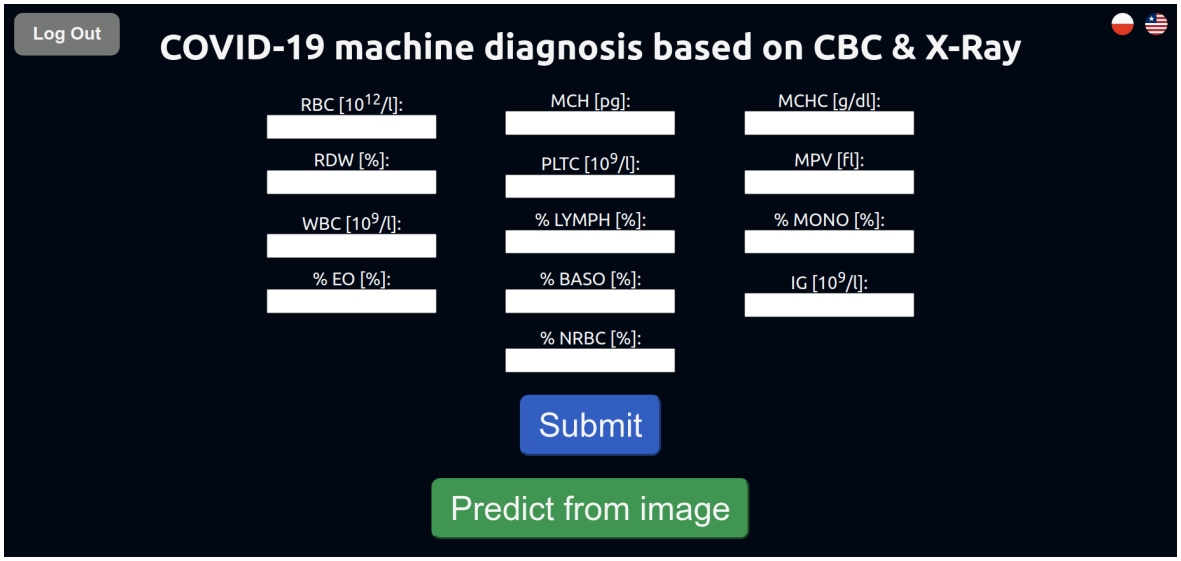


Figure 8.3: Complete blood count prediction form

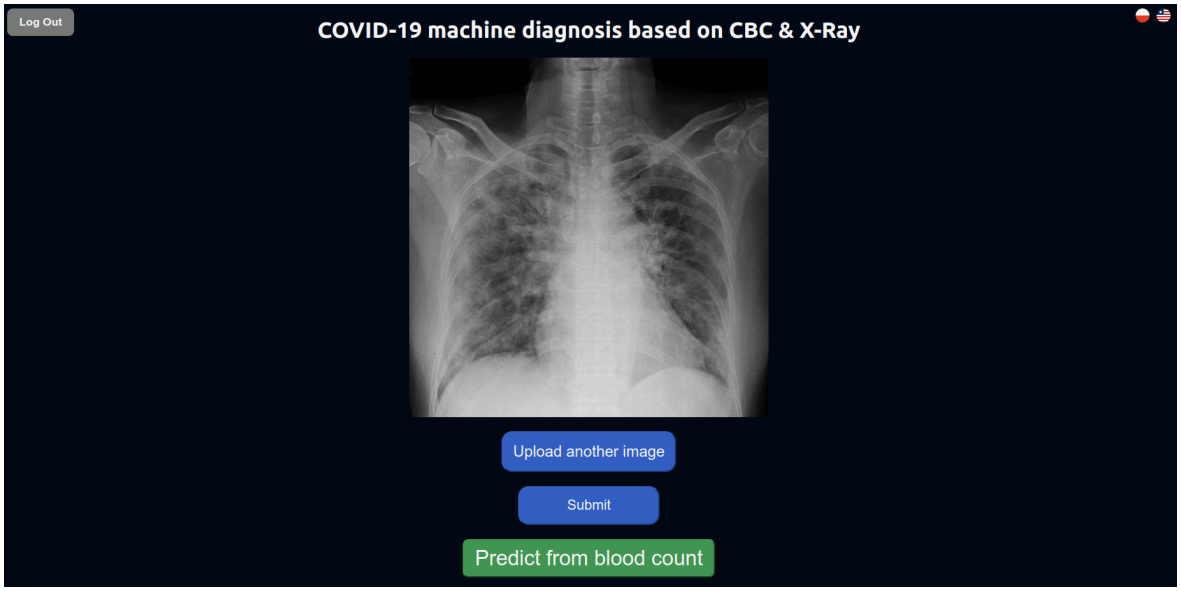


Figure 8.4: Image based prediction page

- Results page that informs about prediction results, presented in figure 8.5.

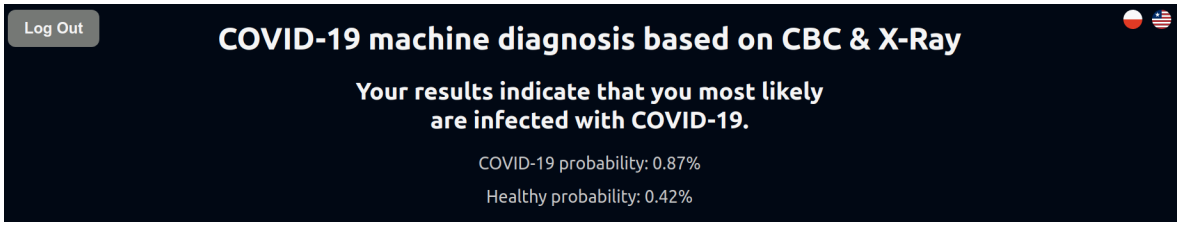


Figure 8.5: Results page

8.3.2 React Components

All components in our application are functional, based on the "React.FC" type. This means that we have to use React hooks [80] to store and manipulate data in component's state or to apply any life cycle related functionality. For example we only load attributes from server in the first render of the prediction page, then again only if user changes the language of the interface. Only one component has to load these attributes and can then distribute them to it's children components using props. All component variables are strongly typed using TypeScript's interfaces. The data can only flow "down" in the hierarchy, so children component cannot change data in parent component unless it specifically defines a way to manipulate it.

8.3.3 Multi-Lingual Support

To provide the best user experience we prepared two language version of the application. The language can be changed in the upper right corner, by clicking on one of the flags. All text for the page is stored in an JavaScript's object, that stores all text for language identified by it's code. Such object with an exemplary "title" key is shown in listing 8.4.

For example, the default language for the application is English, which means that every text is accessed via it's unique key under the "en" entry in translations object,

Listing 8.4: Example of translations object with "title" key

```
const translations: { [key in languageCodes]:
   ITranslationKeys } = {
    pl: {
        title: 'Diagnoza maszynowa COVID-19 oparta o
           morfologie krwi i zdjecia rentgenowskie',
    },
    en: {
        title: 'COVID-19 machine diagnosis based on CBC &
           X-Ray',
    }
};
```

so we use "en.title" path for the title.

8.3.4 Accessibility

To allow for everyone to use the page, we provided alt descriptions for all images on the site. It can also be navigated without the mouse and the text is large and easily readable if someone is visually impaired. However, the application cannot be viewed without a browser supporting JavaScript.

Chapter 9

Discussion

9.1 Data management

Manipulating such amounts of data without a powerful, dedicated computer is difficult, especially when data storage standards are not met. Although there were attempts to standardize .csv format in RFC 4180 [81], actual practice not following these is in conflict with libraries and frameworks that do. Not following well-defined standards caused a lot of overhead work that had to be done in this work. Even though the goals of this work were met, a significant number of issues could have been avoided by following standards.

9.2 Natural language data analysis in X-ray reports

Using the MTT language, we developed rule-based natural language processing model, which is able to correctly featurize the majority of reports, with the average of 2.6 recognized features per report. These features, paired with other patient data, can later be used in the predictions related to COVID-19. In our future work, we are

planning to expand the models, increasing their coverage and therefore the number of features they can extract. We are also planning to manually annotate the part of data enabling us to train deep learning models for tasks such as Named Entity Recognition (NER) in medical domain. We are also planning to use the full corpus to train language models specific for medical domain in the Polish language. Embeddings extracted from such models can later be used to develop accurate medical NLU models [82]. In the future we are also planning to pair the featured datasets with the images to create a source task of predicting features and conditions present in the image that can later be used in training medical convolutions neural networks. Such task can also be used in pair with COVID-19 diagnosis in the multi task learning fashion.

9.3 CBC

In this work we have successfully implemented system for diagnosing COVID-19 based on CBC data. The main difficulty and the direct reason for the model's sub-optimal AUC score was the insufficiency of labeled positive COVID-19 cases. Although the lower number is to be expected as majority of the performed tests turn out to be negative (and let's hope it stays that way), the dataset could be expanded by performing additional CBC tests on patients with known positive PCR test result. In the future we are hoping to extend our work, testing additional re-sampling techniques as they show promising results. As CBC data is only 13.3% of our dataset, we are planning to extend the diagnostic model by additional features in the future. Besides predicting presence or absence of COVID-19 infection, we are also planning to predict the severity of infected patient's condition and need of hospitalization.

9.4 X-ray

The main problem in this task was the quality of the image data provided in the aggregated dataset. This data was stored in PNG format and therefore there was no easy way to separate the annotation from the actual X-ray data. Such problems can be avoided by using specialized data aggregation software, such as MIDAS [83] and following standards.

Our experiments have shown that transfer learning based on larger X-ray datasets can improve the models performance on the task of COVID-19 prediction. When it comes to the effect of providing model with additional information about other pneumonia cases the results appear to be inconclusive, but leaning towards the conclusion that training the models only on binary classification results in better performance. This leads to the final conclusion that COVID-19 diagnosis models can mainly benefit from addition data, not from their detailed description.

9.5 Ensemble Models and Future Work

In our work we present 3 separate models related with analysis of datasets linked with COVID-19 mentioned above. In the future we are hoping to obtain additional data helping us to connect those models. X-ray reports can be paired with their respected images and connected with additional tests such as CBC to create accurate depiction of the patient's health state on which the classification can be based. We are also planning to test different ensemble methods on the models to create a classification system that adapts to the amount of data it has about the patient. This is especially important as some tests can be absent from the patient's history necessitating the need for such versatile models.

Chapter 10

Summary

In this thesis we have presented the workflow of developing machine learning based solutions for COVID-19 diagnosis including the steps of data preparation, training the models and developing final application. We were able to develop 3 models used for featurization of chest X-ray reports and predicting COVID-19 based on X-ray images and CBC data. Through the analysis of the models and data we were able to answer posed research questions providing a baseline for future application. In our following work we are going to continue working on COVID-19 diagnosis extending the amount of analyzed data and connecting the models. Furthermore, such workflow with some adjustments can be applied in developing similar solutions for other diseases or in different medical domain.

Bibliography

[1] Nayaar Islam, Jean-Paul Salameh, Mariska Mg Leeflang, Lotty Hooft, Trevor A McGrath, Christian B Pol, Robert A Frank, Sakib Kazi, Ross Prager, Samanjit S Hare, et al. Thoracic imaging tests for the diagnosis of covid-19. *Cochrane Database of Systematic Reviews*, (11), 2020.

[2] *Python*, 2020. URL: `https://www.python.org`, accessed 12.2020.

[3] *Documentation of virtualenv*, 2020. URL: `https://virtualenv.pypa.io/en/latest/`, accessed 12.2020.

[4] *Preferred installer program (pip)*, 2020. URL: `https://pip.pypa.io/en/stable/`, accessed 12.2020.

[5] *os - Miscellaneous operating system interfaces*. URL: `https://docs.python.org/3/library/os.html`, accessed 12.2020.

[6]

[7] *Documentation of pandas*, 2020. version 1.1.4, URL: `https://pandas.pydata.org`, accessed 12.2020.

[8] *Documentation of numpy*, 2020. version 1.19.4, URL: `https://numpy.org`, accessed 12.2020.

[9] *Documentation of xlsxwriter*, 2020. version 1.3.7, URL: `https://xlsxwriter.readthedocs.io`, accessed 12.2020.

[10] *Documentation of flask*, 2020. version 1.1.2, URL: `https://flask.palletsprojects.com/en/1.1.x/`, accessed 12.2020.

[11] *TypeScript*, 2020. version 4.1.3, URL: `https://www.typescriptlang.org`, accessed 12.2020.

[12] *JavaScript*, 2020. URL: `https://developer.mozilla.org/en-US/docs/Web/JavaScript`, accessed 12.2020.

[13] *Documentation of React*, 2020. version 17.0.1, URL: `https://reactjs.org`, accessed 12.2020.

[14] *Web Server Gateway Interface reference*. URL: `https://wsgi.readthedocs.io/en/latest/`.

[15] *Documentation of Flask-RESTful*, 2020. version 0.3.8, URL: `https://flask-restful.readthedocs.io/en/latest/`, accessed 12.2020.

[16] James; Mogul Jeffrey C.; Nielsen Henrik Frystyk; Masinter Larry; Leach Paul J.; Berners-Lee Tim Fielding, Roy T.; Gettys. Hypertext transfer protocol – http/1.1. 1999.

[17] *Documentation of MySQL*, 2020. version 8.0.22, URL: `https://www.mysql.com`, accessed 12.2020.

[18] *Documentation of SQLAlchemy*, 2020. URL: `https://www.sqlalchemy.org`, accessed 12.2020.

[19] *Documentation of Flask-SQLAlchemy*, 2020. version 2.4.4, URL: `https://flask-sqlalchemy.palletsprojects.com/en/2.x/`, accessed 12.2020.

[20] *Documentation of Flask-Migrate*, 2020. version 2.5.3, URL: `https://flask-migrate.readthedocs.io/en/latest/`, accessed 12.2020.

[21] *Documentation of Alembic*, 2020. URL: `https://alembic.sqlalchemy.org/en/latest/`, accessed 12.2020.

[22] *React's Virtual DOM*. URL: `https://reactjs.org/docs/faq-internals.html`, accessed 12.2020.

[23] *Documentation of react-router-dom*, 2020. version 5.2.0, URL: `https://reactrouter.com/web/guides/quick-start`, accessed 12.2020.

[24] *Repository of axios*, 2020. version 0.21.1, URL: `https://github.com/axios/axios`, accessed 12.2020.

[25] *Documentation of lodash*, 2020. version 4.17.20, URL: `https://lodash.com`, accessed 12.2020.

[26] *Documentation of SASS*. URL: `https://sass-lang.com`, accessed 12.2020.

[27] *Documentation of SCSS*. URL: `https://sass-lang.com/documentation/syntax/`, accessed 12.2020.

[28] *Node Package Manager (npm)*. version 6.14.10, URL: `https://www.npmjs.com`, accessed 12.2020.

[29] *Documentation of react-dom*, 2020. version 16.9.10, URL: `https://reactjs.org/docs/react-dom.html`, accessed 12.2020.

[30] *Documentation of Keras*, 2020. version 2.2.4, URL: `https://keras.io`, accessed 12.2020.

[31] *Documentation of TensorFlow*, 2020. version 2.2.1, URL: `https://www.tensorflow.org`, accessed 12.2020.

[32] *Repository of node-sass*, 2020. version 4.14.1, URL: `https://github.com/sass/node-sass`, accessed 12.2020.

[33] *Repository of react-file-reader*, 2018. version 1.1.4, URL: `https://github.com/GrillWork/react-file-reader`, accessed 12.2020.

[34] *Repository of react-tooltip*, 2020. version 4.2.11, URL: `https://github.com/wwayne/react-tooltip`, accessed 12.2020.

[35] *Repository of mysqlclient*, 2020. version 2.0.3, URL: `https://github.com/PyMySQL/mysqlclient`, accessed 12.2020.

[36] *Repository of tqdm*, 2020. version 4.52.0, URL: `https://github.com/tqdm/tqdm`, accessed 12.2020.

[37] *Documentation of matplotlib*, 2020. version 3.3.3, URL: `https://matplotlib.org`, accessed 12.2020.

[38] *Documentation of requests*, 2020. version 2.25.0, URL: `https://requests.readthedocs.io/en`, accessed 12.2020.

[39] *Documentation of urllib3*, 2020. version 1.25.10, URL: `https://urllib3.readthedocs.io/en/latest/`, accessed 12.2020.

[40] *Documentation of xlrd*, 2020. version 1.2.0, URL: `http://www.python-excel.org`, accessed 12.2020.

[41] World Health Organization. *ICD-10 Version:2019*. URL: `https://icd.who.int/browse10/2019/en#/X`, accessed 12.2020.

[42] *scala*. URL: `https://www.scala-lang.org`, accessed 12.2020.

[43] Alfred V. Aho, Brian W. Kernighan, and J. Peter. Weinberger “awk – a pattern scanning and processing language”. 1978.

[44] Alex Shinn. match.scm – portable hygienic pattern matcher. 2006.

[45] Andrew K. Wright. Robert cartwright "a practical soft type system for scheme". 1997.

[46] *SentiOne*. Homepage: `https://sentione.com/`.

[47] *grep reference*. URL: `https://www.gnu.org/software/grep/`, accessed 12.2020.

[48] Stephen Cole Kleene. Representation of events in nerve nets and finite automata. Technical report, RAND PROJECT AIR FORCE SANTA MONICA CA, 1951.

[49] Phil Hazel. Pcre - perl compatible regular expressions. 2015.

[50] Stanford CoreNLP. *Documentation of tokensregex*. URL: `https://stanfordnlp.github.io/CoreNLP/tokensregex.html`, accessed 12.2020.

[51] Steven Bird, Ewan Klein, and Edward Loper. Natural language processing with python. 2009.

[52] Gerald Gazdar. *Chris Mellish "Natural Language Processing in LISP"*. Addison-Wesley Publishing Company, 1989.

[53] Aleksander Buczyński. Aleksander wawer "shallow parsing in sentiment analysis of product reviews". 2008.

[54] L. Franklin. Deremer "practical translators for lr(k) languages". 1969.

[55] *Documentation of PEG*. URL: `https://pegjs.org/`, accessed 12.2020.

[56] E. Douglas. Appelt "the common pattern specification language". 1998.

[57] Rajasekar Krishnamurthy and Sriram Raghavan. Huaiyu zhu "evolution of rule-based information extraction: From grammars to algebra". 2008.

[58] Unais Sait. Curated dataset for covid-19 posterior-anterior chest radiography images (x-rays)., 2020.

[59] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[60] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[61] Perry Sprawls. X-ray image formation and contrast. *Physical Principles of Medical Imaging*, 2, 2011.

[62] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[63] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[64] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[67] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[68] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[69] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[70] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[71] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

[72] BJM Abma. Evaluation of requirements management tools with support for traceability-based change impact analysis. *Master's thesis, University of Twente, Enschede*, 2009.

[73] Ivan Tomek et al. An experiment with the edited nearest-nieghbor rule. 1976.

[74] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[75] Bank data: Smote, Aug 2020.

[76] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[77] Haibo He and Yunqian Ma. *Imbalanced learning: foundations, algorithms, and applications*. John Wiley & Sons, 2013.

[78] *JavaScript Object Notation*. URL: `https://www.json.org/`, accessed 12.2020.

[79] *Documentation of JSON Web Tokens (jwt)*. URL: `https://jwt.io`, accessed 12.2020.

[80] *React's Hooks*. URL: `https://reactjs.org/docs/hooks-reference.html`, accessed 12.2020.

[81] The Internet Society. *Request For Comments (RFC) 4180*, 2005. URL: `https://tools.ietf.org/html/rfc4180`, accessed 12.2020.

[82] Emily Alsentzer, John R Murphy, Willie Boag, Wei-Hung Weng, Di Jin, Tristan Naumann, and Matthew McDermott. Publicly available clinical bert embeddings. *arXiv preprint arXiv:1904.03323*, 2019.

[83] Barbara Klaudel, Aleksander Obuchowski, Bartosz Rydziński, Roman Karski, Mateusz Glembin, Paweł Syty, and Patryk Jasik. Medical image dataset annotation service (midas). 2020.

List of Figures

Listings

List of Tables