

**Politechnika Warszawska**

W Y D Z I A Ł E L E K T R O N I K I  
I T E C H N I K I N F O R M A C Y J N Y C H



Instytut Radioelektroniki i Technik Multimedialnych

# Praca dyplomowa magisterska

na kierunku Elektronika  
w specjalności Elektronika i Informatyka w Medycynie

Multimodalny system uczenia maszynowego  
do aplikacji medycznych

**Igor Dawid Markiewicz**

Numer albumu 269030

promotor  
prof. dr hab. inż. Władysław Skarbek

WARSZAWA 2020



## **Streszczenie pracy**

Tytuł: MULTIMODALNY SYSTEM UCZENIA MASZYNOWEGO DO APLIKACJI MEDYCZNYCH

Głównym celem niniejszej pracy jest zaprojektowanie i zbudowanie systemu mającego za zadanie wspomagać lekarzy w trakcie diagnozy schorzeń. Zostało to osiągnięte przez podzielenie systemu na kilka części: przykładowe modele uczenia maszynowego (klasyfikacja obrazów zmian skórnych oraz klasyfikacja raka piersi), baza danych, serwisy pośredniczące, infrastrukturę oraz graficzny interfejs użytkownika. Przed budową systemu nastąpiła analiza wymagań, dobór odpowiednich rozwiązań oraz przeprowadzono dodatkowe badania związane z modelami uczenia maszynowego. Na zakończenie działanie systemu zostało zaprezentowane w serwisie Google Cloud Platform.

**Słowa kluczowe:** uczenie maszynowe, Keras, Tensorflow, scikit-learn, Play, Slick, SQLAlchemy, Flask, Docker, Scala, Python, Linux, MySQL, backend, frontend, AJAX, REST API, Google Cloud Platform



### **Abstract of thesis**

Title: MULTIMODAL MACHINE LEARNING SYSTEM FOR MEDICAL APPLICATIONS

The main goal of this master's thesis is to design and build a system that is intended to assist doctors in diagnosis of diseases. It was achieved by dividing the system into several parts: examples of machine learning models (skin lesions image classification and breast cancer classification), database, intermediary services, infrastructure and graphical user interface. Before building the system, requirements were analyzed, appropriate solutions were selected, and additional research related to machine learning models was conducted. Finally, the system was presented on the Google Cloud Platform.

**Keywords:** machine learning, Keras, Tensorflow, scikit-learn, Play, Slick, SQLAlchemy, Flask, Docker, Scala, Python, Linux, MySQL, backend, frontend, AJAX, REST API, Google Cloud Platform





.....  
miejscowość i data

.....  
imię i nazwisko studenta

.....  
numer albumu

.....  
kierunek studiów

### OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....  
czytelny podpis studenta



## **Podziękowania**

Chciałbym serdecznie podziękować mojemu promotorowi Profesorowi Władysławowi Skarbkowi za merytoryczną pomoc i zainteresowanie okazane mi podczas pisania niniejszej pracy. Chciałbym również podziękować mojej rodzinie za okazane wsparcie, które ułatwiło mi realizację mojego celu.

# Spis treści

<b>1 Wstęp</b>	<b>14</b>
1.1 Etap pracy . . . . .	14
1.2 Motywacje do powstania pracy . . . . .	14
<b>2 Projekt systemu</b>	<b>15</b>
2.1 Specyfikacja wymagań systemu . . . . .	15
2.2 Komponenty systemu . . . . .	15
2.3 Hypertext transfer protocol . . . . .	16
2.4 Secure sockets layer . . . . .	17
2.5 Cross site scripting . . . . .	18
2.6 Cross site request forgery . . . . .	18
2.7 Representational state transfer application programming interface . . . . .	19
2.8 Asynchronous JavaScript and XML . . . . .	20
2.9 Docker . . . . .	20
2.10 Przegląd wybranych platform chmurowych . . . . .	21
2.11 Realizacja systemu . . . . .	22
2.11.1 Google Cloud Engine . . . . .	22
2.11.2 Docker Compose . . . . .	23
2.11.3 Baza danych . . . . .	23
2.11.4 Biblioteki i serwisy uczenia maszynowego . . . . .	24
2.11.5 Serwer główny . . . . .	25
2.11.6 Serwer pośredniczący – serwer proxy . . . . .	26
2.11.7 Kodowanie przykładów . . . . .	27
2.11.8 Graficzny interfejs użytkownika . . . . .	27
2.11.9 Schemat systemu . . . . .	31
<b>3 Modele i dane</b>	<b>33</b>
3.1 Tensorowe sieci neuronowe . . . . .	33
3.2 Metryki oceny jakości klasyfikatorów . . . . .	34
3.3 Rak piersi . . . . .	34
3.3.1 Naiwny klasyfikator bayesowski . . . . .	34
3.3.2 Regresja logistyczna . . . . .	35
3.3.3 Maszyna wektorów nośnych (SVM) . . . . .	35
3.3.4 Las losowy . . . . .	36
3.3.5 Gradient tree boosting . . . . .	38
3.3.6 Wielowarstwowy perceptron (MLP) . . . . .	38
3.3.7 Założenia . . . . .	39
3.3.8 Wyniki . . . . .	40
3.4 Zmiany skórne . . . . .	41
3.4.1 Sieć neuronowa <i>Xception</i> . . . . .	42

3.4.2	Sieć neuronowa <i>Xception</i> * . . . . .	44
3.4.3	Preprocessing danych . . . . .	45
3.4.4	Procedury uczenia i testowania . . . . .	46
3.4.5	Augumentacja w czasie rzeczywistym . . . . .	46
3.4.6	Augumentacja w miejscu . . . . .	47
3.4.7	Szersza analiza wyników walidacji dla augmentacji w czasie rzeczywistym . .	49
<b>4</b>	<b>Opis projektu</b>	<b>51</b>
<b>5</b>	<b>Podsumowanie</b>	<b>52</b>
<b>6</b>	<b>Bibliografia</b>	<b>54</b>





## 1 Wstęp

Współczesna medycyna stoi przed wieloma wyzwaniami związanymi z analizą danych i podejmowaniem decyzji. W dobie globalizacji oraz masowego przepływu ogromnych ilości informacji rodzi się potrzeba wspomagania analizy i diagnostyki medycznej. Coś, co kiedyś było eksperymentem lub spojrzeniem w odległą przyszłość, obecnie staje się rzeczywistością konieczną do sprawnego działania oraz odkrywania nowych rozwiązań. Szczególnie ważnymi aspektami są systemy wczesnego ostrzegania oraz systemy wspomagania diagnostyki. Niniejsza praca skupiać się będzie na wykorzystaniu algorytmów uczenia maszynowego do analizy danych medycznych oraz budowie systemu informującego lekarzy lub techników medycznych o wykrytych nieprawidłowościach, celem ich dalszej analizy przez ludzi lub bardziej zaawansowane algorytmy. Umożliwi to wykrywanie potencjalnych stanów patologicznych z możliwością szybkiej reakcji przed właściwymi analizami.

### 1.1 Etap pracy

Jako główne etapy pracy przedstawione zostały:

- specyfikacja wymagań systemu
- wybór architektury systemu
- wybór bazy danych oraz implementacja zależności
- wybór przykładowych zbiorów danych
- wybór przykładowych algorytmów uczenia maszynowego
- wybór sposobu uczenia
- wyznaczenie podstawowych metryk jakości algorytmów
- w przypadku wielu konkurencyjnych algorytmów – selekcja modelu
- budowa systemu typu backend oraz frontend

Pozostawiono także możliwość podjęcia dodatkowych celów takich, jak np: przeprowadzenie badań dla niektórych algorytmów i danych.

### 1.2 Motywacje do powstania pracy

Motywacją do powstania pracy było zainteresowanie zagadnieniami związanymi z algorytmami uczenia maszynowego oraz wdrażaniem systemów informatycznych. Temat powstał jako połączenie obu tych dziedzin, przez co utworzył całościowo jeden, spójny system, którego realizacji podjęto się w niniejszej pracy.

## 2 Projekt systemu

### 2.1 Specyfikacja wymagań systemu

W początkowej fazie budowy systemu nastąpiła specyfikacja wymagań użytkownika końcowego (dalej zwanego jako *użytkownik*) oraz ogólnych oczekiwani w względem systemu jako najważniejszych podstaw jego projektu. Z racji, że głównym zadaniem systemu jest wspomaganie lekarzy przy diagnozie chorób wyspecyfikowano następujące żądania:

- system powinien zapewniać wygodny interfejs graficzny służący jako frontend dla użytkownika w postaci przeglądarki internetowej (lekki klient)
- system powinien zapewniać możliwość podglądu wyników dotychczas przeprowadzonych już badań w postaci rozkładów prawdopodobieństw
- system powinien zapewniać możliwość dodawania nowych przykładów celem ich diagnozy
- system powinien zapewniać logowanie użytkowników
- system powinien zapewniać użytkownikowi informację o prawidłowym formacie danych wymaganych dla poszczególnych badań
- system powinien zapewniać podstawową obsługę błędów np: w przypadku nieprawidłowego formatu danych wprowadzonego przez użytkownika lub wewnętrznego wyjątku w systemie
- system powinien przekazywać użytkownikowi oprócz wyników badań również pewne metadane takie, jak np: id danego badania, datę jego zlecenia oraz tytuł wprowadzany przez użytkownika
- ze względów praktycznych przyjęto, że do analizy będą wysyłane pojedyncze przykłady
- system powinien zapisywać wszystkie przesłane przykłady wraz z informacją, od którego użytkownika pochodzą tak, aby w razie potrzeby można było je wykorzystać
- komunikacja między serwisami oraz między systemem a użytkownikiem powinna odbywać się przez *REST API*[1] z wykorzystaniem protokołu *HTTP(S)*[2]
- powinno być zapewnione podstawowe bezpieczeństwo danych takie jak, np: stosowanie *Secure Sockets Layer – SSL*[3], zabezpieczenia w przypadku *Cross Site Scripting – XSS*[4] czy *Cross Site Request Forgery – CSRF*[5]
- system powinien ze względów praktycznych być wysoce przenośny tak, aby zapewnić możliwość jego umieszczenia w natywnych środowiskach chmurowych

### 2.2 Komponenty systemu

Po analizie wymagań zdecydowano zastosować następujący podział systemu:

- część zawierająca osobne serwisy z których każdy reprezentuje jeden, serwowany model uczenia maszynowego

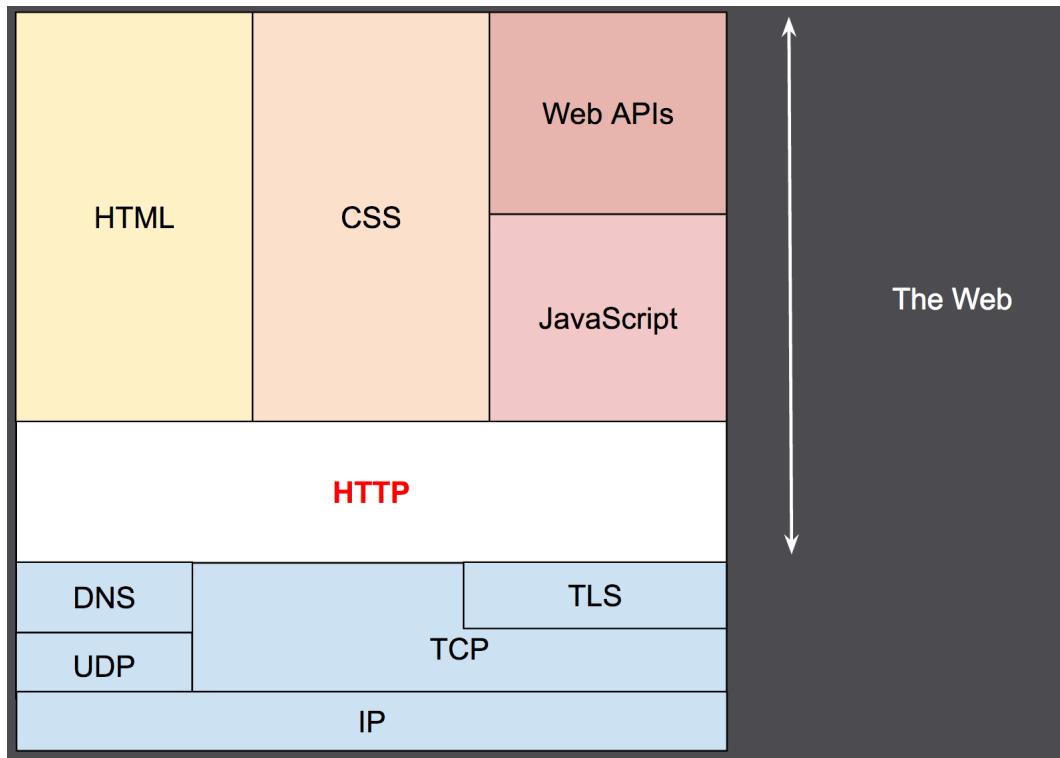
- baza danych do przechowywania zależności
- trwały storage służący do zapisu analizowanych przykładów
- serwer obsługujący klientów oraz łączący się z bazą danych i serwerem pośredniczącym
- serwer pośredniczący między serwerem obsługującym klientów a serwisami uczenia maszynowego oraz bazą danych

### 2.3 Hypertext transfer protocol

*Hypertext Transfer Protocol (HTTP)* jest protokołem komunikacyjnym w warstwie aplikacyjnej modelu *TCP/IP*[6] funkcjonującym na zasadzie żądanie–odpowiedź w modelu klient–serwer. Cechami tego protokołu są:

- bezstanowość – nie jest wymagane przechowywanie informacji o sesji klienta po stronie serwera (choć niektóre serwery ze względów praktycznych implementują takie rozwiązanie)
- niezależność od typów danych – protokół HTTP może przesyłać dane dowolnego typu
- źródła są identyfikowane i lokalizowane przez *Uniform Resource Locator (URL)* [7] reprezentujące konkretny schemat
- sesyjność – sekwencja żądań–odpowiedzi
- dwa główne elementy żądań i odpowiedzi:
  - *header* – nagłówek zawierający metadane
  - *body* – ciało zawierające właściwą treść wiadomości (czasem może być opcjonalne)
- typy żądań:
  - *GET* – żądanie reprezentacji pewnego zasobu
  - *HEAD* – jak *GET*, ale bez ciała wiadomości
  - *POST* – żądanie, by serwer zaakceptował wiadomość w ciele żądania jako nowe źródło
  - *PUT* – żądanie aktualizacji konkretnego źródła
  - *DELETE* – żądanie usunięcia wybranego źródła
  - *TRACE* – żądanie zwrócenia otrzymanej wiadomości przez serwer od klienta (funkcja echo)
  - *OPTIONS* – żądanie informacji o wspieranych przez serwer żądaniach
  - *CONNECT* – konwersja połączenia do transparentnego tunelu TCP/IP w przypadku żądania wykorzystującego *SSL* i połączenia przez *HTTP proxy*
  - *PATCH* – żądanie aktualizacji konkretnego pola w konkretnym źródle
- typy komunikatów:

- 1XX – informacyjne
- 2XX – potwierdzenia
- 3XX – przekierowania
- 4XX – błędy po stronie klienta
- 5XX – błędy po stronie serwera



Rys. 1: Rola protokołu HTTP[8]

## 2.4 Secure sockets layer

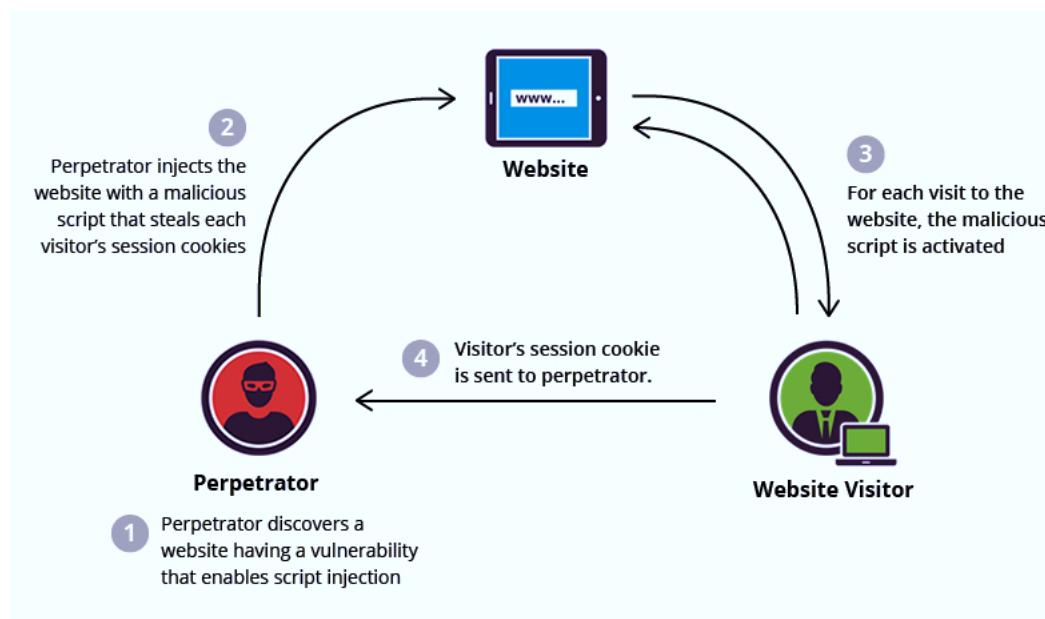
*Secure Sockets Layer (SSL)* [3] jest rozwiązaniem pozwalającym na uwierzytelnianie internetowych źródeł (takich jak strony internetowe) oraz szyfrowanie danych. W tym celu jest przeprowadzana procedura generacji par kluczy – publicznego (ogólnodostępnego dla wszystkich zainteresowanych) oraz prywatnego według certyfikatu X.509[9]. Algorytm najpierw weryfikuje witrynę z użyciem klucza publicznego, a następnie używa go do szyfrowania standardowych wiadomości które, będą mogły być odszyfrowane tylko za pomocą klucza prywatnego. Protokół HTTP z zabezpieczeniem SSL nosi nazwę *HTTPS*.

W niniejszej pracy komunikacja między serwisami odbywa się z wykorzystaniem surowego protokołu HTTP (z wyjątkiem komunikacji z bazą danych, która ma własny protokół). Zdecydowano się na ten krok ze względu na to, że wszystkie serwisy działają w ramach jednego systemu operacyjnego, więc zastosowanie HTTPS nie przyniosłoby dużych korzyści. Protokół HTTPS został natomiast wykorzystany do komunikacji między użytkownikiem a serwerem odbierającym od niego żądania.

Mimo że jego zastosowanie jest w pracy ograniczone (zrezygnowano ze względów praktycznych na ubieganie się o odpowiedni certyfikat), to stanowi przykład wykorzystania takiego rozwiązania w praktyce.

## 2.5 Cross site scripting

*Cross Site Scripting (XSS)*[4] jest rodzajem podatności, w której atakujący umieszcza na danej stronie skrypt (przeważnie JavaScript) powodujący wykonywanie niepożądanych akcji na tej stronie w przypadku jej odwiedziny przez użytkownika. Jednym z przykładów takiej akcji jest przesyłanie do atakującego ciasteczek sesyjnych użytkowników.

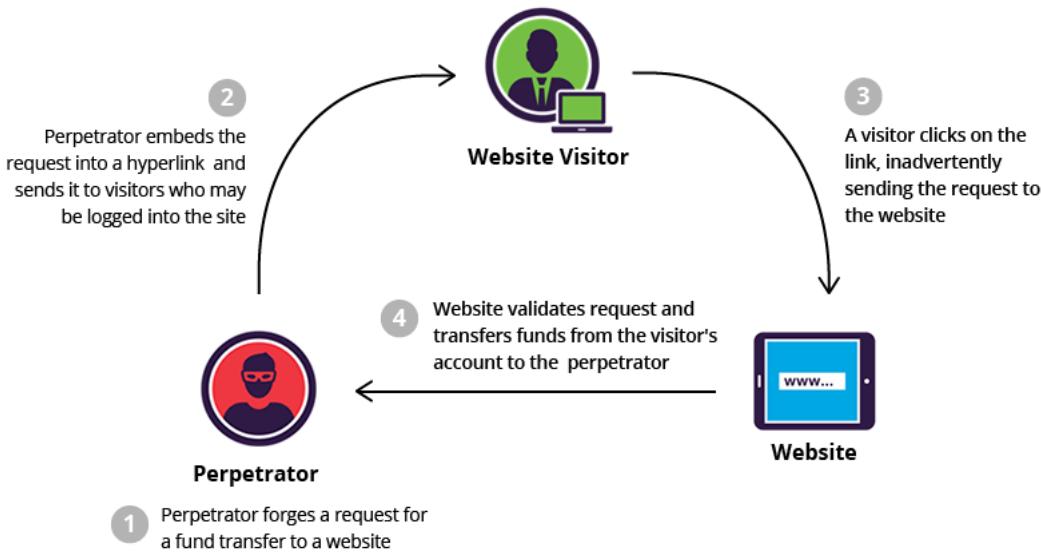


Rys. 2: Schemat działania ataku typu XSS[4]

Przykładowym zabezpieczeniem, wykorzystywany w przypadku ciasteczek w niniejszym projekcie, jest ustawienie w nich opcji *httpOnly*, która blokuje możliwość odczytu ich zawartości przez API inne niż HTTP (czyli np: JavaScript).

## 2.6 Cross site request forgery

*Cross Site Request Forgery (CSRF)*[5] jest podatnością, w której atakujący wykorzystuje nieuwagę ofiary podsypując, jej spreferowany materiał (np: link). Jeśli użytkownik jest zalogowany na docelowej stronie dla, której jest przeprowadzany atak i wejdzie w interakcję ze spreferowanym materiałem, to spowoduje to wysłanie żądania atakującego w imieniu użytkownika który, już przeszedł autentykację.



Rys. 3: Schemat działania ataku typu CSRF[5]

Zabezpieczeniem wykorzystywanym w niniejszym projekcie jest tzw. *CSRF token*. Jest to losowo generowany przez serwer ciąg znaków (dla każdej sesji), który jest następnie przesyłany do użytkownika. Atakujący z jednej strony nie jest w stanie zgadnąć jego wartości, z drugiej zaś, ze względu na separację dostępu przez *HTTP access control* między różnymi źródłami, złośliwe skrypty nie będą w stanie poznać jego wartości. Po wysyłaniu żądania klient dodaje swój token, a serwer potwierdza jego autentyczność.

## 2.7 Representational state transfer application programming interface

*Representational state transfer application programming interface (REST API)*[1] jest wzorcem architektonicznym rozwoju serwisów sieciowych z najczęstszą implementacją w postaci protokołu HTTP. Jego głównymi cechami są:

- model klient–serwer
- bezstanowość – niezależność serwera od innych żądań oraz stanów wewnętrznych (stan sesji jest przetrzymywany po stronie klienta)
- cashability – możliwość zapisania odpowiedzi serwera i jej użycia w późniejszych, takich samych żądaniach
- jednorodny interfejs
- warstwowość systemu – każda warstwa widzi tylko swoje interakcje

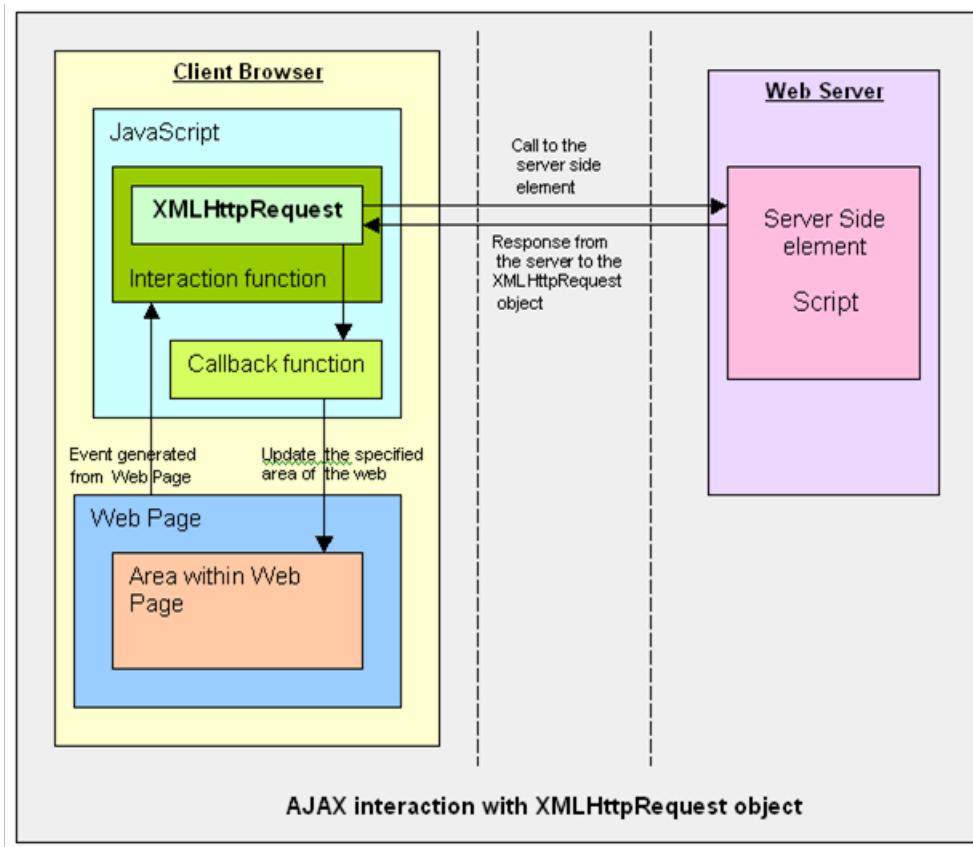
Serwis, który implementuje REST API określany jest mianem *RESTful*.

W ramach jednorodności interfejsu postanowiono, że wszystkie informacje będą przesyłane w ramach reprezentacji danych typu JSON[10].

## 2.8 Asynchronous JavaScript and XML

*Asynchronous JavaScript And XML (AJAX)*[11] jest techniką pozwalającą generować asynchroniczny model żądań–odpowiedzi po stronie klienta. W jego skład wchodzą:

- HTML lub XHTML i CSS dla warstwy prezentacji
- *Document Object Model* do dynamicznego wyświetlania widoków oraz interakcji z danymi
- JSON lub XML do reprezentacji danych (w niniejszym projekcie użyty został JSON) oraz XSLT do manipulacji
- obiekt *XMLHttpRequest* do asynchronicznej komunikacji
- JavaScript do zespolenia wszystkich powyższych mechanizmów



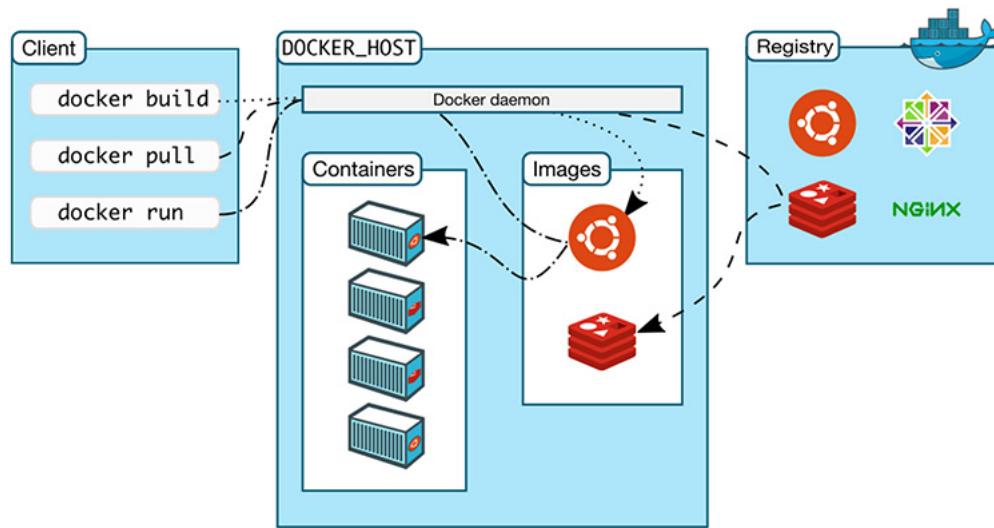
Rys. 4: Schemat działania AJAX[12]

## 2.9 Docker

Docker[13, 14] to rozwiązanie, które wykorzystuje wirtualizację na poziomie systemu operacyjnego. Zapewnia to z jednej strony enkapsulację programu ze wszystkimi jego zależnościami oraz izolację od systemu operacyjnego hosta (z którym komunikacja zazwyczaj następuje przez bindowanie portów), a z drugiej strony lekkość, wygodę użytkowania oraz wysoką skalowalność.

Można wyróżnić następujące, główne elementy Docker-a:

- Docker Client – odpowiada za interakcję z użytkownikami
- Docker Daemon – proces uruchomiony w systemie operacyjnym hosta odpowiadający za REST API
- Docker Registry - repozytorium obrazów dockerowych
- Docker Images - obrazy dockerowe, czyli zbindowane kody źródłowe napisane przez programistę oraz wymagane zależności
- Docker Containers - uruchomione procesy obrazów dockerowych (każdy obraz może mieć jeden lub więcej kontenerów)



Rys. 5: Architektura Docker-a[15]

Ze względów praktycznych istnieje możliwość przenoszenia plików między systemem macierzystym hosta a kontenerami Dockera stosowana np: z racji wygody umieszczania plików konfiguracyjnych czy tego, że sam kontener powinien być bezstanowy.

## 2.10 Przegląd wybranych platform chmurowych

System zdecydowano się zaimplementować w środowisku chmurowym[16], co zapewniło możliwość wygodnego, wydajnego i wysoce modyfikowalnego zarządzania całym projektem. W tym celu należało najpierw wybrać odpowiedniego dostawcę usług chmurowych.

	Zalety	Wady
Amazon Web Services[17]	1. najstarszy reprezentant na rynku z największym doświadczeniem 2. duża liczba różnych serwisów oraz opcji 3. globalny zasięg 4. aws free tier z limitami godzinowo – pojemnościowymi	1. złożony w użyciu 2. często wyższa cena niż u konkurencji
Microsoft Azure[18]	1. integracja z innymi rozwiązaniami firmy Microsoft 2. duża liczba różnych serwisów oraz opcji 3. większa otwartość na rozwiązania hybrydowe niż u konkurencji 3. 12 miesięcy popularnych serwisów za darmo + 200 \$ przez 30 dni na wszystkie serwisy + część serwisów za darmo	1. problemy z dokumentacją 2. przejściowe problemy z zarządzaniem
Google Cloud Platform[19]	1. duże zaangażowanie w projektach typu open source 2. liczne prezenceny w elastycznych ofertach 3. duże doświadczenie w zakresie Dev Ops 4. 300 \$ do wykorzystania przez 12 miesięcy + always free tier	1. najpóźniej debiutujący reprezentant z mniejszym doświadczeniem biznesowym 2. mniejsza ilość rozwiązań oraz opcji niż u konkurencji (następuje jednak szybki rozwój)

**Tab. 1:** Porównanie wybranych dostawców usług chmurowych

Ostatecznie na dostawcę został wybrany Google Cloud Platform. Decydującymi czynnikami okazały się atrakcyjna oferta finansowa oraz duży udział w projektach typu open source.

## 2.11 Realizacja systemu

W kolejnych podrozdziałach nastąpiło omówienie poszczególnych elementów zaproponowanego systemu.

### 2.11.1 Google Cloud Engine

*Google Cloud (Compute) Engine*[20, 21] jest jednym z serwisów Google Cloud w systemie *Infrastructure as a Service (IaaS)*[22] pozwalającym na specyfikację fizycznego sprzętu, ustawień sieciowych, wybór systemów operacyjnych oraz uruchomienie maszyn wirtualnych. Dzięki temu z powodzeniem można dostosować dane instancje do różnorodnych zadań. Ponadto zapewnia również wygodny dostęp do *Google Cloud Storage*[23], czyli serwisu zapewniającego przechowywanie danych z dostępem do każdego innego serwisu Google Cloud. Ze względu na wysoką elastyczność Google Cloud Engine oraz ograniczony zasięg projektowanego systemu zdecydowano, że to właśnie ten serwis będzie stanowił podstawę do jego budowy.

W projekcie wykorzystano dwie instancje tego typu:

- w przypadku uczenia maszynowego – instancja służąca do uczenia modelów w tensorflow/keras o następujących parametrach:
  - system operacyjny Ubuntu 16.04
  - 4 vCPU z processorem z rodziny Intel Skylake
  - 15 GB RAM
  - karta graficzna NVIDIA Tesla V100 do akceleracji *CUDA*[24]
  - 20 GB HDD
- główna instancja do obsługi systemu:
  - system operacyjny Ubuntu 16.04

- 1 vCPU z processora z rodziny Intel Skylake
- 3.75 GB RAM
- 20 GB HDD

### 2.11.2 Docker Compose

*Docker Compose*[25] jest rozwiązańem pozwalającym na wygodne zarządzanie wielokontenerową aplikacją, które obejmuje takie zagadnienia jak tworzenie wewnętrznej sieci między kontenerami, zadania restartowania kontenerów czy ich specyfikacje na podstawie dostępnych obrazów lub plików dockerowych. Ze względów praktycznych Docker Compose jest przeważnie używany w fazie rozwoju oprogramowania (*development*), natomiast w fazie produkcyjnej jego rolę przejmuje natywna architektura danego dostawcy usług zapewniająca zawsze bardziej korzystne rozwiązania. W niniejszej pracy, ze względów finansowych oraz ograniczonego zasięgu systemu zdecydowano, się wykorzystać Docker Compose również w fazie produkcyjnej wraz z Google Cloud Engine.

### 2.11.3 Baza danych

Jako bazę danych postanowiono zastosować *MySQL*[26] w wersji 8.0.17. Za wyborem tego rozwiązania stały:

- darmowy dostęp
- powszechność użycia
- dobra dokumentacja
- planowana baza była nieskomplikowana, w związku z tym nie wymagała bardzo zaawansowanych funkcjonalności

Baza danych została zbudowana z dwóch encji:

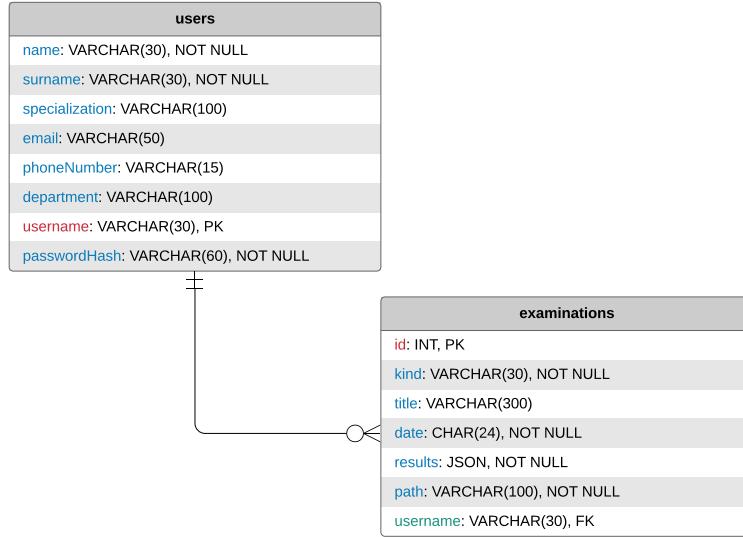
1. **users** – encja zawierająca dane o użytkownikach:

- *name* – imię
- *surname* – nazwisko
- *specialization* – specjalizacja
- *email* – adres email
- *phoneNumber* – numer telefonu
- *department* – przynależna jednostka
- *username* – nazwa logowania, klucz główny
- *passwordHash* – hash hasła logowania

2. **examinations** – encja zawierająca dane o badaniach:

- *id* – identyfikator badania nadawany automatycznie, klucz główny

- *kind* – rodzaj badania
- *title* – tytuł badania wprowadzany przez użytkownika
- *date* – data i czas zlecenia badania
- *results* – JSON z rozkładem prawdopodobieństwa poszczególnych klas
- *path* – ścieżka do pliku w systemie z zapisanym przykładem
- *username* – nazwa logowania, klucz obcy



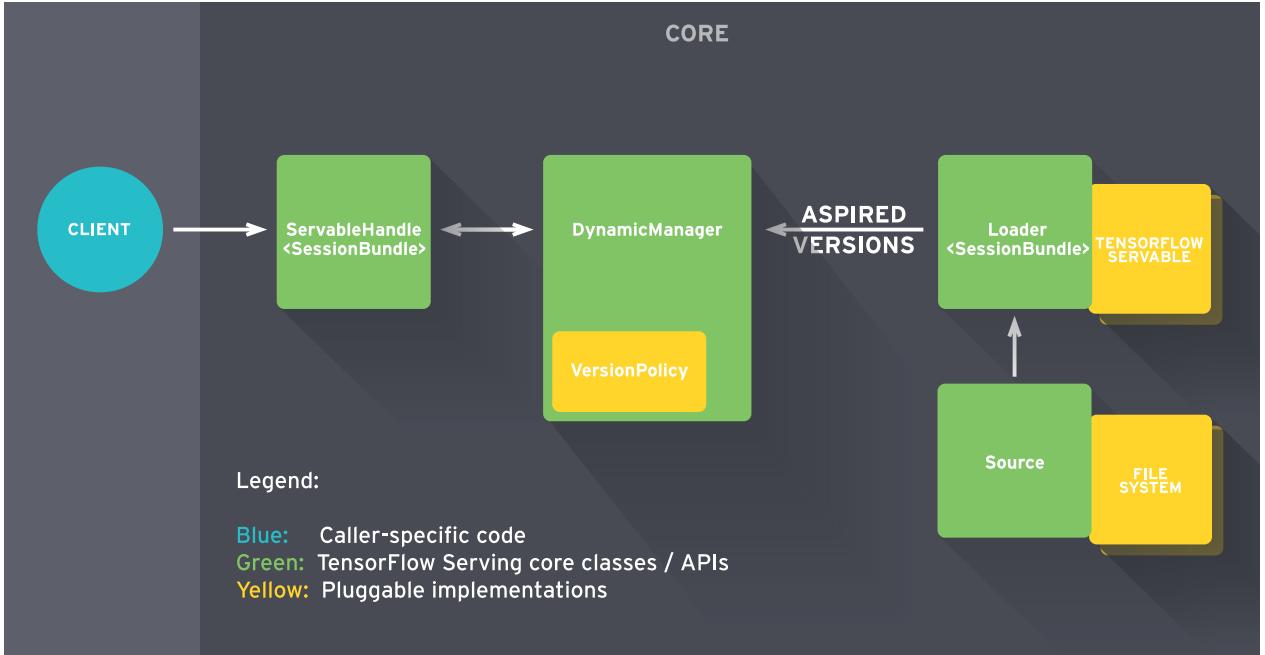
Rys. 6: Diagram UML dla bazy danych

W bazie zastosowano kaskadowe aktualizacje dla usuwania i aktualizowania rekordów.

#### 2.11.4 Biblioteki i serwisy uczenia maszynowego

W celu prezentacji systemu postanowiono wykorzystać dwa modele. Pierwszy z nich dotyczy sieci neuronowej z biblioteki *Keras*[27] wspartej na bibliotece *Tensorflow*[28] w wersji 1.13. Biblioteka Keras umożliwiła wydajną obsługę samej sieci, Tensorflow z kolei zapewnił możliwość wykorzystania *Tensorflow Serving*[29] – wysoce wydajnej architektury, przeznaczonej do serwowania wyuczonych modeli w trybie produkcyjnym. Dzięki możliwości użycia jako obrazu Docker-a sprawia to że dane rozwiązanie jest bardzo efektywne i wygodne. Obraz Tensorflow Serving zawiera dwie możliwości komunikacji z klientem:

- standardowe REST API z przesyaniem danych w formacie JSON, wykorzystywane w niniejszym projekcie z danymi przesyłanymi w ciele żądania
- wysoce wydajne *gRPC*[30] z własnym kodowaniem danych



Rys. 7: Architektura Tensorflow Serving[29]

W systemie jest reprezentowany jako obraz o nazwie *tensorflow/serving*.

W przypadku drugiego modelu postanowiono wybrać jeden z dostępnych w pakiecie *scikit-learn*[31] w wersji 0.21.2 oraz *XGBoost*[32] w wersji 0.80. W celu jego serwowania z powodu braku poszczególnego rozwiązania, postanowiono stworzyć własny serwis. Został on oparty na framewoku *Flask*[33] w wersji 1.1.1, zapewniającej funkcjonalność mikroserwera do ładowania wyuczonego, zserializowanego modelu oraz obsługi przychodzących żądań przez REST API. W systemie ten serwis jest reprezentowany jako obraz o nazwie *breast-cancer-server*.

Dla obu modeli serwisy ich obsługujące mają tylko jedno zadanie – po otrzymaniu żądania następuje jego parsowanie, inferencja, a następnie odesłanie odpowiedzi z rozkładem prawdopodobieństwa klas do klienta.

### 2.11.5 Serwer główny

Główny serwer został wykonany w języku *Scala*[34] 2.11 przy użyciu framework-a *Play*[35] w wersji 2.7.3 działającego w modelu *Model–View–Controller (MVC)*[36]. Jest to *web framework* oparty o framework *Akka*[37], który zapewnia wysokowydajną architekturę do wielowątkowego i równoległego przetwarzania danych. Play zawiera wygodne rozwiązania w zakresie wywoływanie REST API innych serwisów (*Play WS*), kontroli przychodzących żądań (*Play filters*), jak również możliwość natywnego łączenia kodu Scali oraz HTML (*Twirl*).

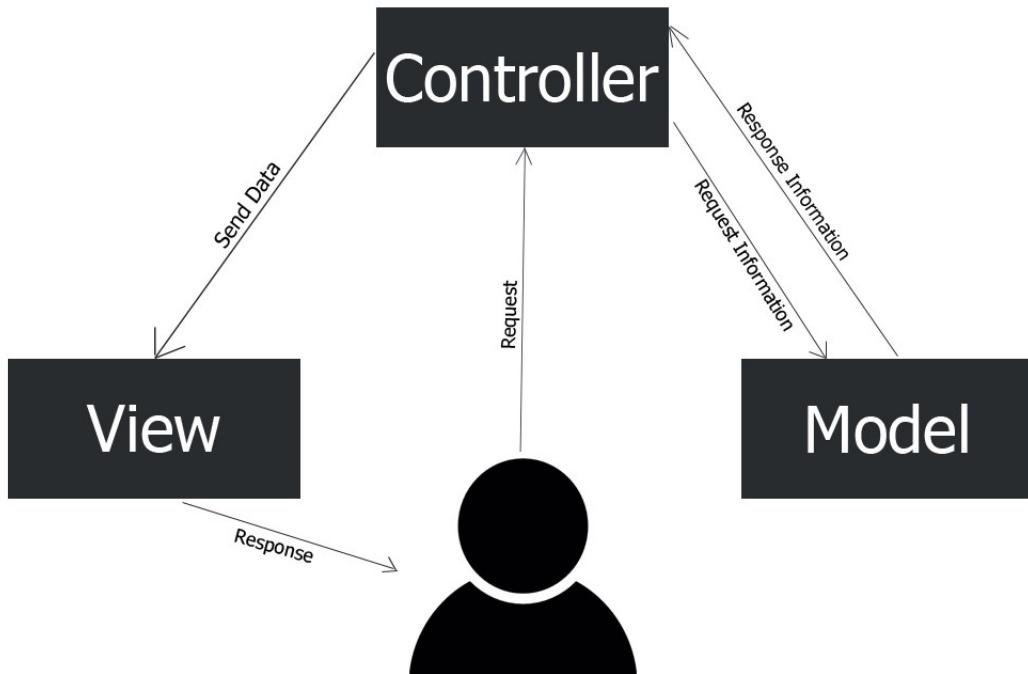
Do zadań tego komponentu należą:

- wyświetlanie wszystkich widoków dla użytkownika
- logowanie, wylogowanie oraz autentykacja użytkowników – następuje to poprzez porównanie wprowadzonego *username* z tymi, znajdującymi się w bazie danych oraz porównanie hasha

dla wprowadzonego hasła z odpowiednim hashem w bazie danych. Hasła zostały zhashowane przy użyciu procedury *bcrypt*[38].

- przesyłanie ciasteczek z metadanymi o sesji
- zapewnienie bezpieczeństwa danych
- prezentacja dotychczas wykonanych badań wraz z ich sortowaniem nierosnąco po czasie, a także sortowanie nierosnąco rozkładów prawdopodobieństw w wynikach
- przekazywanie nowych przykładów do serwera pośredniczącego przez REST API

## Model-View-Controller



Rys. 8: Schemat działania modelu MVC[36]

W celu komunikacji z bazą danych został użyty framework *Slick*[39] w postaci pluginu *play-slick* w wersji 4.0.2 zapewniający z jednej strony wygodny, silnie typowany dostęp do informacji zawartych w bazie danych, a z drugiej strony możliwość używania rozwiązań z natywnych kolekcji języka Scala. W serwerze zaimplementowano filtr zabraniający żądań nie będących typu AJAX (czyli np. bezpośrednie odwoływanie się przez URL). Jedynymi wyjątkami są wczytywanie strony głównej, wylogowanie oraz żądania odnośnie statycznego kontentu.

Obraz z tym serwisem w projekcie został oznaczony jako *play-server*.

### 2.11.6 Serwer pośredniczący – serwer proxy

Serwer proxy został wykonany we frameworku Flask w wersji 1.1.1, a dostęp do bazy danych odbywa się przez framework *SQLAlchemy*[40] w wersji 1.3.10. Do zadań tego serwisu należą:

- przyjmowanie zgłoszeń z przykładami od głównego serwera
- zapisywanie nowych przykładów oraz wysyłanie ich do serwisów uczenia maszynowego
- odbiór odpowiedzi od serwisów uczenia maszynowego, ich wzbogacenie w metadane, a następnie zapisanie do bazy danych
- zwrócenie informacji z odpowiednim statusem HTTP do głównego serwera

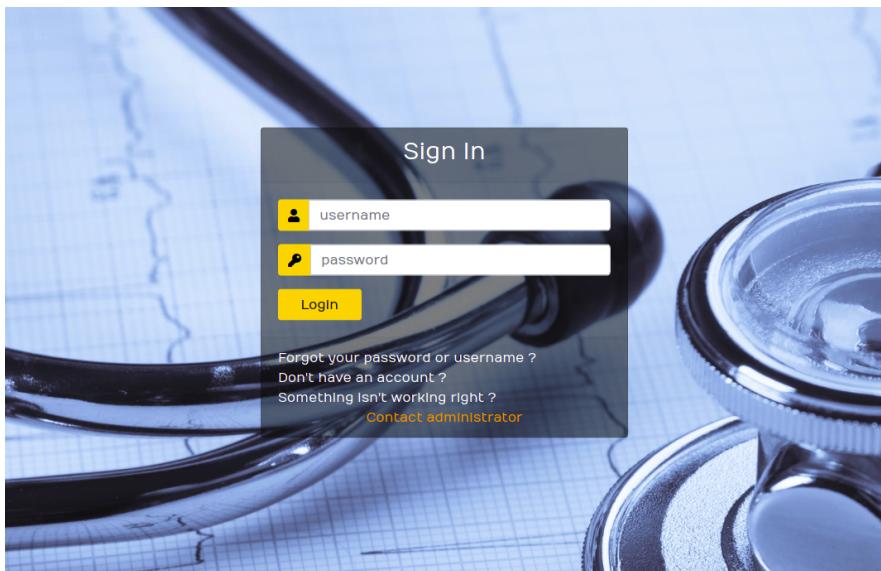
Obraz z tym serwisem w projekcie został oznaczony jako *proxy-server*.

#### 2.11.7 Kodowanie przykładów

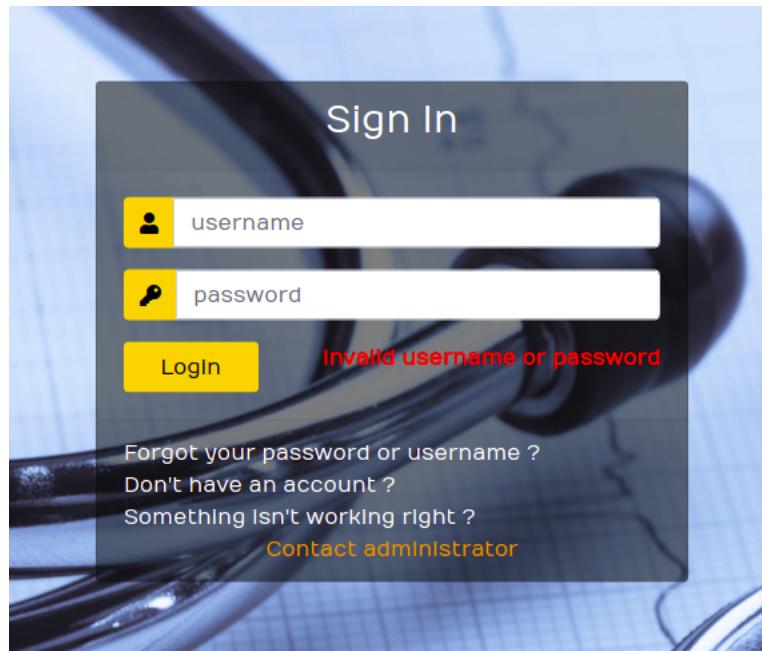
Postanowiono przyjąć spójny system kodowania wszystkich przesyłanych przykładów z użyciem protokołu *Base64*[41] reprezentującego dane binarne w postaci ciągu znaków *ASCII*[42]. Jego zaletami są prostota oraz wygoda użytkowania, jednak niewątpliwą wadą jest zwiększenie zajętości miejsca przez dane.

#### 2.11.8 Graficzny interfejs użytkownika

Widoki systemu zostały wykonane z użyciem HTML, Twirl, CSS oraz biblioteki JavaScript *jQuery* [43] zapewniającej nie tylko większą interaktywność ale, również możliwość wygodnego wykonywania żądań typu AJAX. Ponadto z racji na niewielkie rozmiary interejsu użytkownika zdecydowano się wykonać go jako *Single-page application (SPA)*[44].



Rys. 9: Główny panel logowania

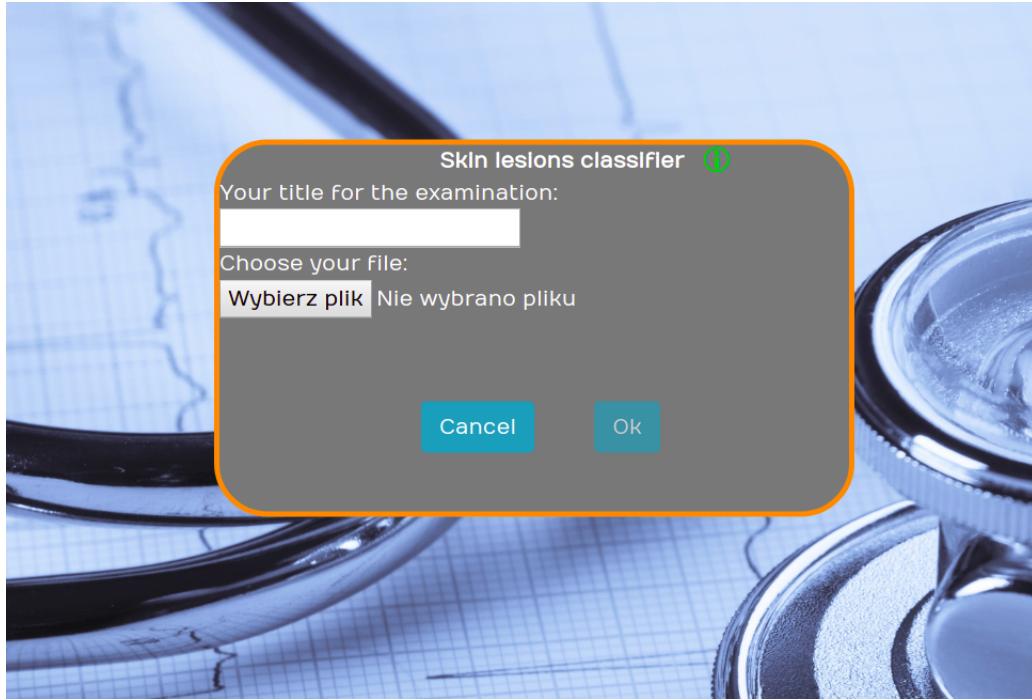


Rys. 10: Główny panel logowania z informacją o błędny logowaniu

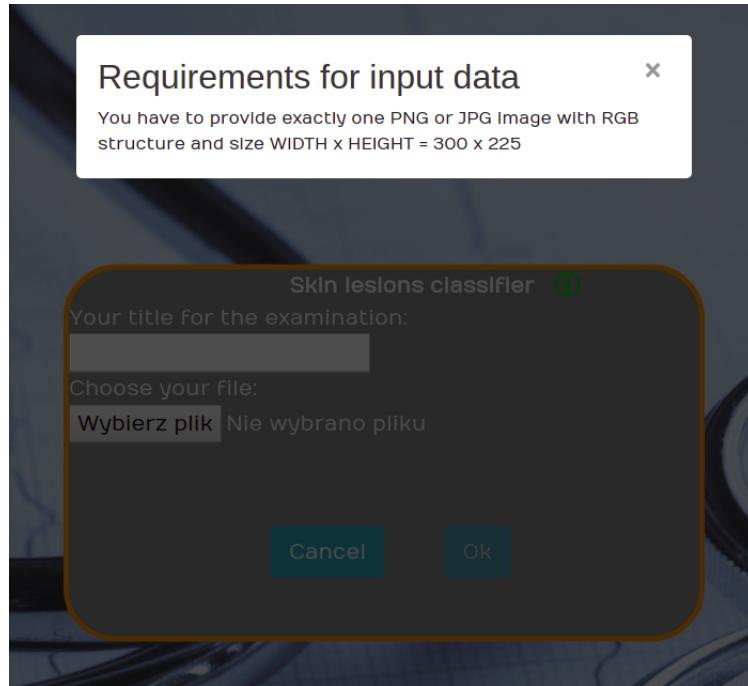
The dashboard features a table with columns: Id, kind, title, date, and results. The 'results' column contains a list of tumor types and their percentages. A blue button 'Add new examination' is located at the top left. The top right corner shows the user 'Logged as: kurek' and a 'Logout' button. A stethoscope and a heart rate monitor are visible in the background.

Add new examination				
Id	kind	title	date	results
63	breast-cancer	br2	2019-32-10 13:32:26	<ul style="list-style-type: none"> <li>malignant - 100.0 %</li> <li>benign - 0.0 %</li> </ul>
62	skin-lesions	sk2	2019-50-09 21:50:50	<ul style="list-style-type: none"> <li>nv - 47.54 %</li> <li>mel - 24.42 %</li> <li>bkl - 14.27 %</li> <li>df - 5.49 %</li> <li>bcc - 3.69 %</li> <li>vasc - 2.52 %</li> <li>akiec - 2.07 %</li> </ul>
61	skin-lesions	sk	2019-47-09 21:47:45	<ul style="list-style-type: none"> <li>nv - 47.54 %</li> <li>mel - 24.42 %</li> <li>bkl - 14.27 %</li> <li>df - 5.49 %</li> <li>bcc - 3.69 %</li> <li>vasc - 2.52 %</li> <li>akiec - 2.07 %</li> </ul>
60	breast-cancer	br	2019-47-09 21:47:09	<ul style="list-style-type: none"> <li>malignant - 100.0 %</li> <li>benign - 0.0 %</li> </ul>
59	breast-cancer	u2	2019-26-09 21:26:37	<ul style="list-style-type: none"> <li>malignant - 100.0 %</li> <li>benign - 0.0 %</li> </ul>
58	skin-lesions		2019-24-09 21:24:17	<ul style="list-style-type: none"> <li>nv - 47.54 %</li> <li>mel - 24.42 %</li> <li>bkl - 14.27 %</li> <li>df - 5.49 %</li> <li>bcc - 3.69 %</li> </ul>

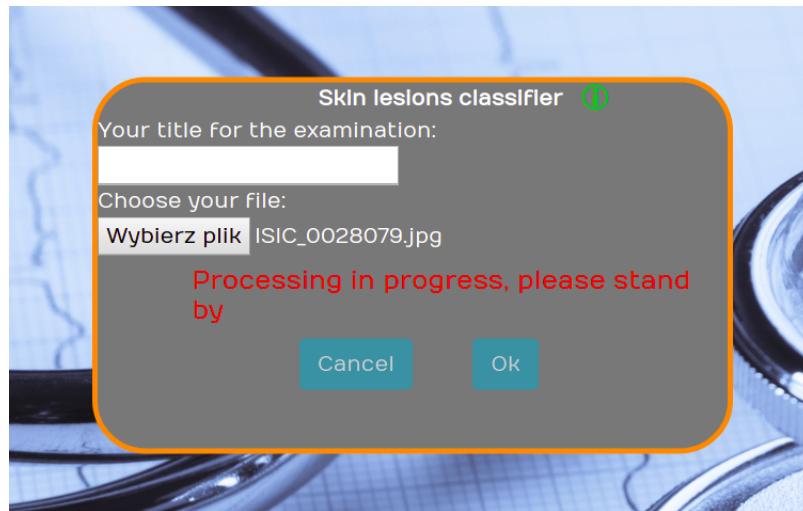
Rys. 11: Główny kokpit



Rys. 12: Definicja przykładowego badania



Rys. 13: Definicja przykładowego badania z dodatkową informacją po kliknięciu w zielony symbol koło nazwy badania

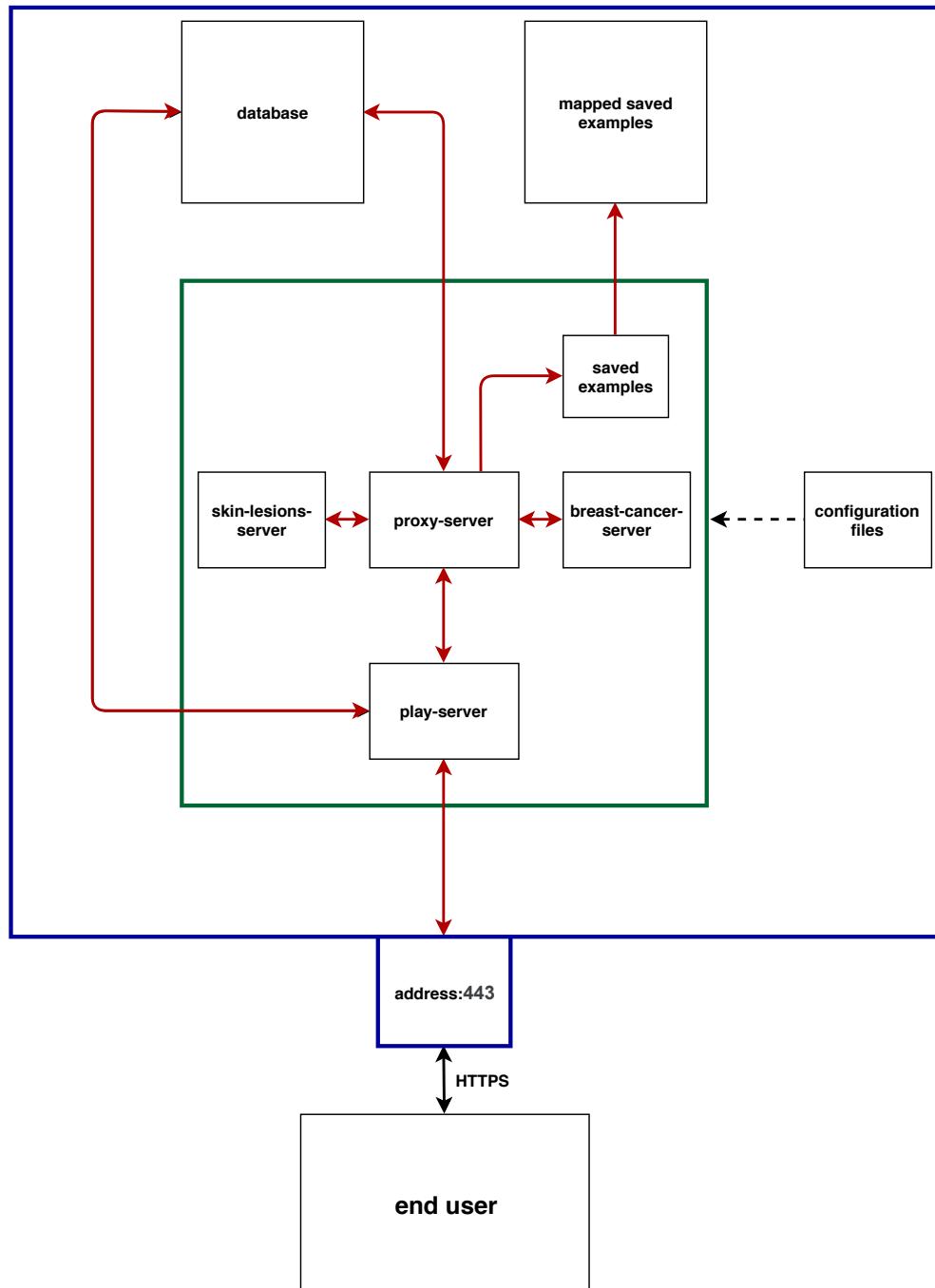


Rys. 14: Definicja przykładowego badania z prośbą o oczekивание на wynik



Rys. 15: Generyczny wyjątek odsyłany w przypadku niepowodzenia

### 2.11.9 Schemat systemu



Rys. 16: Architektura systemu. Kolorem niebieskim oznaczono instancje Google Cloud Engine, zielonym Docker Compose a czerwonym komunikację między serwisami.

Jako podstawowy schemat działania systemu po uruchomieniu możemy przedstawić:

1. Użytkownik końcowy wchodzi na odpowiednią stronę *www*, po czym otrzymuje widok logowania

2. Następuje faza logowania, podczas której wprowadzane są dane do autentykacji przez użytkownika, a play-server odpytuje bazę danych o podanego użytkownika i jeśli istnieje, to sprawdza zgodność hashy dla haseł. W przypadku pozytywnej weryfikacji użytkownik zostaje zalogowany wraz z przydzieloną sesją w ciasteczku do kokpitu głównego łącznie z wynikami dotychczasowych badań pobranymi przez play-server z bazy danych. W przeciwnym razie zostaje wyświetlona informacja o niepowodzeniu logowania.
3. Zalogowany użytkownik ma teraz możliwość:
  - przejrzeć dostępne badania
  - wylogować się, przez co zostaje usunięta jego sesja, a on sam jest przekierowany do widoku logowania
  - dodać nowe badanie przez kliknięcie na odpowiednią pozycję w rozwijanym menu
    - (a) użytkownik wprowadza opcjonalny tytuł badania oraz plik z odpowiednim przykładem. Ma również możliwość podejrzenia dodatkowych informacji o wymaganiach
    - (b) przykład z metadanymi jest przesyłany do play-server, który przekierowuje go do proxy-server
    - (c) proxy-server zapisuje przykład na dysk do odpowiedniego dla danego badania folderu wraz z nazwą, w której prefixem jest nazwa użytkownika. Następnie Docker Compose mapuje dane foldery na foldery zewnętrzne
    - (d) proxy-server przesyła przykład do odpowiedniego serwisu uczenia maszynowego, po czym odbiera odpowiedź z rozkładem prawdopodobieństwa poszczególnych klas
    - (e) proxy-server otrzymane rezultaty wzbogaca w metadane i zapisuje do bazy danych
    - (f) proxy-server zwraca do play-server informacje ze statusem HTTP 200 w przypadku pozytywnego zakończenia procesu lub HTTP 500 w przeciwnym razie
    - (g) play-server po otrzymaniu wiadomości ze statusem HTTP 200 odpytuje bazę danych o wyniki badań dla danego użytkownika, a następnie je zwraca i przekierowuje użytkownika do kokpitu głównego. W przeciwnym razie zostaje zwrócony użytkownikowi końcowem widok z informacją o wystąpieniu wyjątku

### 3 Modele i dane

Zbudowany system został użyty do hostowania dwóch przykładowych algorytmów klasyfikacji. Każdy z nich został zaprezentowany w kontekście konkretnego zbioru danych, jak również odpowiednich metryk jakości pozwalających ocenić jego przydatność.

#### 3.1 Tensorowe sieci neuronowe

W ramach pracy magisterskiej postanowiono zastosować reprezentację sieci neuronowych w postaci tensorowej, która pozwala na ich spójną prezentację w formie niegraficznej (Skarbek [45]). Poniżej przedstawiony jest fragment teoretyczny wymagany do zdefiniowania wykorzystywanej sieci w następnych rozdziałach.

- $\alpha \mapsto \mathcal{I}_{\gamma_1 x \gamma_2 y}^{\beta}$  – tensor zapisany pod oznaczeniem  $\alpha$  jest przekazywany do wejścia sieci przyjmującego tensory o liczbie kanałów (głębokości) równej  $\beta$  oraz wymiarach przestrzennych  $\gamma_1$  wzdłuż osi  $x$  oraz  $\gamma_2$  wzdłuż osi  $y$ . Jeśli  $\gamma_1, \gamma_2$  nie są podane, to przyjmuje się, że sieć może obsługiwać dowolne rozmiary przestrzenne.
- $\alpha \mapsto \mathbb{C}_{\alpha \sigma \beta}^{\gamma}$  – tensor jest zapisywany pod oznaczeniem  $\alpha$ 
  - $\alpha$  – wartość parametru *striding* (zakładając, że jest taki sam wzdłuż osi  $x$  i  $y$ )
  - $\beta$  – rozmiar przestrzenny jądra (zakładając, że jest taki sam wzdłuż osi  $x$  i  $y$ )
  - $\gamma$  – liczba jąder
  - $\delta$  – typ konwolucji, zwykły, jeśli parametr jest pusty oraz *depthwise separable convolution*, jeśli jest oznaczony jako *DS*
  - $\epsilon$  – dodatkowe operacje jak np:
    - \*  $b$  – *batch normalization*
    - \*  $r$  – funkcja aktywacji *ReLU*
- $\alpha \mapsto \mathbb{P}_{\alpha \sigma \beta}^{\gamma}$  – warstwa typu *pooling*:
  - $\alpha$  – wartość parametru *striding* (zakładając, że jest taki sam wzdłuż osi  $x$  i  $y$ )
  - $\beta$  – rozmiar przestrzenny okna (zakładając, że jest taki sam wzdłuż osi  $x$  i  $y$ )
  - $\gamma$  – rodzaj operacji, jeśli  $a$ , to *average pooling*, jeśli  $g$ , to *global average pooling*
- $\alpha \mapsto \mathbb{D}_{\alpha}$  – *dropout* o wartości  $\alpha$
- $\mathbb{F}_{\alpha}^n$  – *fully connected layer (dense layer)* z  $n$  neuronami oraz dodatkowymi operacjami  $\alpha$  (jak w przypadku warstwy konwolucyjnej)
- $\mathcal{R}$  – funkcja *ReLU* wydzielona jako osobny element

- $\langle \alpha \rangle^n$  – połączenie rezydualne z operacjami  $\alpha$  powtórzone  $n$ -krotnie
- $\langle \alpha | \beta \rangle^n$  – połączenie rezydualne z operacjami w obu odgałęzieniach – odpowiednio  $\alpha$  oraz  $\beta$  powtórzone  $n$  razy, tzw. *cast adder block*

### 3.2 Metryki oceny jakości klasyfikatorów

Do oceny jakości klasyfikatorów na podstawie prawdziwego wektora wartości klas  $\mathbf{y}$  oraz przewidzianego  $\hat{\mathbf{y}}$  (oba o rozmiarach  $m$ ) postanowiono wykorzystać następujące metryki:

- celność predykcji (*accuracy*) –  $100 \cdot \frac{l}{m}$ , gdzie  $l$  jest liczbą stanów zgodnych na odpowiadających sobie pozycjach w obu wektorach
- współczynnik  $F_1$  ( $F_1$  score):
  - dla klasyfikacji binarnej:  $F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$
  - dla klasyfikacji z  $N$  klasami – makrouśrednianie:  $F_1 \text{ macro} = \frac{1}{N} \sum_{i=1}^N F_{1,i}$ , gdzie  $F_{1,i}$  są wyznaczane w schemacie klasyfikacji *One – vs – All*

### 3.3 Rak piersi

Jako pierwszy zbiór danych został wybrany zbiór dotyczący zmian w tkance piersiowej pochodzący z Uniwersytetu w Winsconsin [46]. Zadaniem postawionym w tym przypadku była klasyfikacja przykładów po atrybutach numerycznych. Zbiór zawiera 569 rekordów, z których każdy ma przypisaną dokładnie jedną klasę (B (*benign*) – zmiana łagodna, 357 przykładów oraz M (*malignant*) – zmiana złośliwa, 212 przykładów). Każdy przykład posiada 32 atrybuty, z czego jeden jest identyfikatorem (atrribut *id*), a kolejny typem zmiany (atrribut *diagnosis*). Pozostałe 30 atrybutów stanowią atrybuty predykcyjne będące liczbami rzeczywistymi. Zbiór nie zawiera wartości pustych oraz nieokreślonych.

Poniżej przedstawione zostały wykorzystane algorytmy do klasyfikacji opisanych danych. Jeśli nie jest oznaczone inaczej to, przyjęte zostało, że  $\mathbf{x} = [x_1, x_2, \dots, x_n]$  oznacza n-iltrubutowy wektor wejściowy,  $K$  liczbę klas, a  $m$  liczbę przykładów.

#### 3.3.1 Naiwny klasyfikator bayesowski

Zakładając niezależność atrybutów od siebie, można uzyskać klasyfikator o następującej postaci [47, 48]:

$$\hat{y} = \arg \max_{k \in \{1, 2, \dots, K\}} p(C_k) \prod_{i=1}^n p(x_i | C_k) \quad (1)$$

Estymacja prawdopodobieństwa warunkowego odbywa się przez zliczanie zdarzeń. W przypadku rozkładu ciągłego należy albo przeprowadzić dyskretyzację atrybutów, albo przyjąć odpowiednią gęstość rozkładu prawdopodobieństwa.

### 3.3.2 Regresja logistyczna

W przypadku binarnego rozkładu klas ( $K = 2$ ,  $y \in \{0, 1\}$ ) możemy zdefiniować [47, 49, 50]:

$$h_{\mathbf{w}, \beta}(\mathbf{x}) = p(y = 1 | \mathbf{x}, \mathbf{w}, \beta) = \frac{1}{1 + e^{-(\mathbf{x} \cdot \mathbf{w} + \beta)}} \in (0, 1) \quad (2)$$

$$p(y = 0 | \mathbf{x}, \mathbf{w}, \beta) = 1 - p(y = 1 | \mathbf{x}, \mathbf{w}, \beta) \quad (3)$$

gdzie  $\mathbf{w}, \beta$  oznaczają pewne parametry modelu. Następnie definiujemy:

- funkcję wiarygodności:

$$L(\mathbf{w}, \beta) = \prod_{i=1}^m \left( \frac{1}{1 + e^{-(\mathbf{x}_i \cdot \mathbf{w} + \beta)}} \right)^{y_i} \left( 1 - \frac{1}{1 + e^{-(\mathbf{x}_i \cdot \mathbf{w} + \beta)}} \right)^{1-y_i} \quad (4)$$

- oraz odpowiadającą funkcję kosztu:

$$l(\mathbf{w}, \beta) = - \left( \frac{1}{m} \right) \ln (L(\mathbf{w}, \beta)) \quad (5)$$

Po wykorzystaniu metody minimalizacji np. spadku gradientu do funkcji kosztu otrzymujemy zestaw parametrów modelu dla, którego wnioskowanie możemy zdefiniować jako:

$$\hat{y} = \begin{cases} 1, & \text{dla } h_{\mathbf{w}, \beta}(\mathbf{x}) \geq 0,5 \\ 0, & \text{w.p.p} \end{cases} \quad (6)$$

### 3.3.3 Maszyna wektorów nośnych (SVM)

W przypadku binarnego rozkładu klas ( $K = 2$ ,  $y \in \{-1, 1\}$ ) możemy próbować znaleźć hiperpłaszczyznę rozdzielającą przykłady danych klas tak aby odległości przykładów od granicy decyzyjnej była maksymalizowana [47, 51]. Zakładając warunki na rozdzielenie danych liniowo separowalnych:

$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_i + b \geq 1, & \text{dla } y_i = 1 \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1, & \text{dla } y_i = -1 \end{cases} \implies y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad (7)$$

można zdefiniować następnie rozluźnioną funkcję kosztu (tzw. *miękkie marginesy*) dla danych nieseparowalnych liniowo:

$$l(\mathbf{w}, \beta) = \left[ \frac{1}{m} \sum_{i=1}^m \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b)) \right] + \frac{\lambda}{2} \|\mathbf{w}\|^2 \quad (8)$$

Powyższa funkcja kosztu dba z jednej strony o maksymalizację odległości przykładów od marginesu klasyfikacji, a z drugiej strony pozwala na liniową nieseparowalność danych wprowadzając kary dla przykładów znajdujących się po zlej stronie granicy. Jej optymalizacja może zostać przeprowadzona np. przy wykorzystaniu metod programowania kwadratowego.

Wnioskowanie przeprowadza się następnie przez:

$$\hat{y} = \begin{cases} 1, & \text{dla } \mathbf{w} \cdot \mathbf{x}_i + b \geq 0 \\ -1, & \text{dla } \mathbf{w} \cdot \mathbf{x}_i + b < 0 \end{cases} \quad (9)$$

W przypadku SVM, dość powszechnie wykorzystywana praktyka jest używanie tzw. *kernel trick*. Polega ona na wykorzystaniu odpowiedniej funkcji jądrowej do nieliniowej transformacji atrybutów na poziomie samego modelu (z powodzeniem może zastąpić transformację atrybutów a priori, przed zastosowaniem modelu, co bywa dość uciążliwe). Transformacji  $\phi$  pewnego punktu  $\mathbf{x}_i$ :

$$\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i) \quad (10)$$

odpowiada zastosowanie odpowiedniej funkcji jądrowej  $k$  takiej, że:

$$(\forall_{\mathbf{x}_i, \mathbf{x}_j}) \quad k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \quad (11)$$

gdzie  $\langle \cdot, \cdot \rangle$  oznacza produkt wewnętrzny. Taki zabieg można rozumieć dwójako – jako wprowadzenie nieliniowości do algorytmu lub przeniesienie liniowego algorytmu do nowej przestrzeni. To z kolei rodzi szanse na lepszą klasyfikację danych.

### 3.3.4 Las losowy

Podstawowym obiektem w lesie losowym jest drzewo, którego budowa stanowi osobny algorytm [47]. Przykładowy pseudokod budowania drzewa decyzyjnego w schemacie *Top–Down Induction of Decision Tree* [47, 52]:

**funkcja** buduj–drzewo( $P, d, S$ )

**argumenty wejściowe:**

- $P$  - zbiór przykładów etykietowanych pojęcia  $c$
- $d$  - domyślna etykieta kategorii
- $S$  - zbiór możliwych testów

**zwraca:** drzewo decyzyjne reprezentujące hipotezę przybliżającą  $c$  na zbiorze  $P$ ;

1: **jeśli** kryterium–stopu( $P, S$ ) **to**

2:   utwórz liść **l**;

3:    $d_l := \text{kategoria}(P, d)$ ;

4:   **zwróć** **l**;

5: **koniec jeśli**

6: utwórz węzeł **n**;

7:  $t_n := \text{wybierz–test}(P, S)$ ;

8:  $d := \text{kategoria}(P, d)$ ;

9: **dla wszystkich**  $r \in R_{t_n}$  **wykonaj**

10:    $n[r] := \text{buduj–drzewo}(P_{t_n r}, d, S - \{t_n\})$ ;

11: koniec dla

12: zwróć n

$d_l$  - kategoria  $d$  liścia 1

$t_n$  - związany z węzłem  $n$  test  $t$

$R_{t_n}$  - zbiór możliwych wyników testu  $t$  dla węzła  $n$

$P_{tnr}$  - zbiór przykładów z  $P$ , które dla testu  $t$  i węzła  $n$  są stowarzyszone z wynikiem  $r$

$n[r]$  - węzeł lub liść potomny, do którego prowadzi z węzła  $n$  gałąź odpowiadająca wynikowi  $r$

Można w ogólności następująco wyjaśnić:

- kryterium stopu – daje wynik pozytywny, gdy zachodzi przynajmniej jeden z następujących przypadków:
  - Zbiór przykładów  $P$  jest pusty
  - Zbiór testów  $S$  jest pusty
  - Zbiór przykładów  $P$  zawiera przykłady wyłącznie jednej kategorii
- kategoria( $P, d$ ) =  $\begin{cases} d, & \text{gdy } P = \emptyset \\ \arg \max_{d'} |P^{d'}|, & \text{w pozostałych przypadkach} \end{cases}$
- rodzaj testu – często utożsamiany z testowaniem pojedynczego atrybutu. W takim przypadku dla sytuacji z dyskretną dziedziną można np: użyć bezpośrednio możliwych wartości atrybutu jako wyników testu. Dla sytuacji z ciągłą dziedziną przeprowadzana jest jej dyskretyzacja (np. poprzez wybór środków między wartościami danego atrybutu dla przykładów znajdujących się w danym węźle, a zbiorem możliwych wartości testu są podprzedziały liczbowe)
- wybór testu – w przypadku klasyfikacji dość częstym wyborem jest tzw. *information gain*. Wybieramy taki test dla przykładów w danym węźle, który najbardziej je różnicuje. Przykładowymi miarami takiej informacji są np. entropia czy współczynnik Giniego.

Las losowy jest zbiorem drzew, w którym:

- dla każdego drzewa jest losowany podzbiór przykładów ze zbioru trenującego ze zwracaniem o rozmiarze oryginalnego zbioru trenującego
- w każdym węźle każdego drzewa losowany jest niewielki podzbiór atrybutów, dla którego jest wybierany test
- drzewa rosną do dużych rozmiarów
- klasyfikacja odbywa się metodą głosowania, zliczana jest liczba drzew głosujących za poszczególnymi kategoriami, a następnie jest wybierana ta, na którą oddano najwięcej głosów

Las losowy jest przykładem metody *bootstrap aggregating (bagging)*, która opiera się na redukcji błędu predykcji przez agregację silnych klasyfikatorów (o małym obciążeniu), ale dużej wariancji. Uśrednianie wyników powoduje w takim przypadku znaczącą jej redukcję.

### 3.3.5 Gradient tree boosting

Ideą metody typu *boosting* jest minimalizacja błędu predykcji poprzez agregację słabych klasyfikatorów (o dużym obciążeniu), ale małej wariancji. Dzieje się to po przez budowę kolejnych modeli (algorytm sekwencyjny), z których każdy oprócz pierwszego (będącego modelem inicjalizującym) stara się poprawić łączną predykcję wszystkich poprzednich modeli działających w schemacie adytywnym [47].

Miara jakości modelu:

$$Q(h_{1:n}) = \sum_{x \in T} l(y(x), h_{1:n}(x)) + \sum_{i=1}^n \Gamma(h_i) \quad (12)$$

gdzie:

- $h_i$  –  $i$ -ty model bazowy (drzewo)
- $h_{1:n}$  – model zespołowy z  $n$  drzewami
- $h_i(x)$  – predykcja zespołu drzew  $h_1, h_2, \dots, h_n$  dla przykładu  $x$ :

$$h_{1:n} = \sum_{i=1}^n h_i(x) \quad (13)$$

- $T$  – zbiór trenujący o rozmiarze  $m$
- $y(x)$  – prawdziwa wartość atrybutu docelowego dla przykładu  $x$ , w przypadku klasyfikacji  $K = 2$ ,  $y(x) \in \{0, 1\}$
- $l$  – funkcja strat, miara jakości predykcji
- $\Gamma(h_i)$  – składnik regularyzacji dla drzewa  $h_i$  będący karą za złożoność w celu ograniczenia ryzyka nadmiernego dopasowania

Przy tak dobranym modelu następuje następnie optymalizacja funkcji celu metodami gradientowymi dla kolejnych drzew:

$$Q(h_i) = \sum_{x \in T} l(y(x), h_{1:i-1}(x) + h_i(x)) + \Gamma(h_i) \quad (14)$$

Po optymalizacji otrzymujemy odpowiednie kryterium wyboru podziału w węzłach oraz optymalne wartości w liściach drzewa, przy czym zakłada się a priori, że drzewa nie mogą mieć więcej niż konkretną liczbę liści lub być głębsze niż pewna maksymalna wartość.

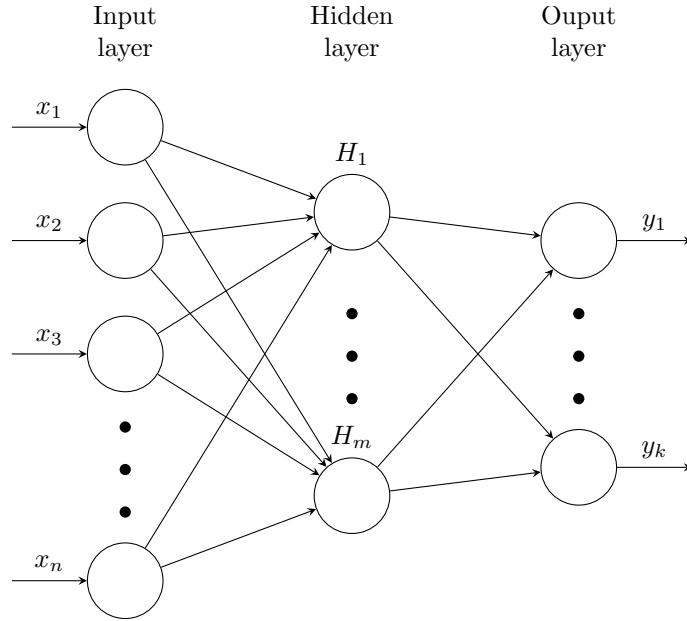
W przypadku klasyfikacji można wykorzystać funkcję straty opartą o regresję logistyczną.

### 3.3.6 Wielowarstwowy perceptron (MLP)

Sieć neuronową typu wielowarstwowy perceptron można przedstawić w kontekście tensorowych sieci neuronowych jako:

$$\boxed{\alpha} \rightarrow \mathcal{I}_x \quad \mathbb{F}_{f_1}^{l_1} \quad \mathbb{F}_{f_2}^{l_2} \dots \quad \mathbb{F}_{f_n}^{l_n} \mapsto \boxed{\text{score}}$$

gdzie  $l_1, l_2, \dots, l_n$  oznaczają liczbę neuronów w poszczególnych warstwach, a  $f_1, f_2, \dots, f_n$  funkcje aktywacji w poszczególnych warstwach. Można również zastosować niektóre z dodatkowych operacji takich jak dropout czy batch normalization.



Rys. 17: Przykład MLP z jedną warstwą ukrytą.

### 3.3.7 Założenia

- *Naive Bayes* – założono rozkład normalny
- *Support Vector Machine* – kernel trick z radialną funkcją bazową
- *Random Forest* – 100 estymatorów o maksymalnej głębokości `max_depth = 4`
- *Gradient Tree Boosting* – 100 estymatorów o maksymalnej głębokości `max_depth = 4`
- *Multilayer Perceptron*

**Zaproponowane MLP jako symboliczna sieć neuronowa:**

$$\boxed{\begin{array}{c} @ \mapsto \mathcal{I}_x \quad \mathbb{F}_r^{10} \mapsto \text{score} \\ \hline \mathcal{NET} \left( \alpha := 30_x; \text{optima} := [\text{Crossentropy}, \text{Adam}, \text{Softmax}] \right) \end{array}}$$

Wszystkie algorytmy oprócz *Gradient Tree Boosting* pochodzą z pakietu *scikit-learn*[31] (algorytm *Gradient Tree Boosting* zaś z *XGBoost*[32]). Proces wyboru odpowiedniego algorytmu uczenia maszynowego został przeprowadzony w czterech krokach:

1. przeprowadzenie 10 krotnej walidacji krzyżowej (z wcześniejszą permutacją elementów) na całym dostępnym zbiorze danych dla wszystkich algorytmów (na tych samych podzbiorach)

2. ocena jakości poszczególnych klasyfikatorów na podstawie statystyk dla celności predykcji oraz współczynnika  $F_1$ , a następnie wybór ostatecznego modelu
3. podział w sposób losowy na podzbior ucący (80 %) oraz testowy (20 %) wszystkich dostępnych danych oraz zastosowanie wybranego modelu
4. ocena końcowego modelu

W trakcie walidacji krzyżowej oraz finalnego uczenia – testowania zastosowano standaryzację danych. Każda z  $x_{i,j}$  wartości predykcyjnych (gdzie  $i \in \{1, 2 \dots 30\}$  oraz  $j \in \{1, 2 \dots 569\}$ ) została znormalizowana:

$$x\_norm_{i,j} = \frac{x_{i,j} - \hat{\mu}_{x_i}}{\hat{\sigma}_{x_i}}$$

Gdzie  $\hat{\mu}_{x_i}$  oraz  $\hat{\sigma}_{x_i}$  (odpowiednio estymata wartości średniej oraz odchylenia standardowego) zostają wyznaczone na podstawie odpowiednich zbiorów uczących.

### 3.3.8 Wyniki

	Min [%]	Max [%]	Mean [%]	Median [%]	Standard deviation [%]
<b>Naive Bayes</b>	87.5	96.5	92.8	93.9	3.6
<b>Logistic regression</b>	93.0	100.0	97.5	98.2	2.0
<b>SVM</b>	94.7	100.0	97.4	97.4	1.4
<b>Random forest</b>	93.0	98.2	96.1	96.5	1.5
<b>Gradient Boosting</b>	94.7	100.0	97.0	96.5	1.4
<b>MLP</b>	94.7	100.0	98.1	98.2	1.8

**Tab. 2:** Celność predykcji dla zbiorów walidacyjnych. Zielonym kolorem zaznaczono najlepsze wyniki w danej kolumnie.

	Min	Max	Mean	Median	Standard deviation
Naive Bayes	0.8	0.96	0.9	0.93	0.06
Logistic regression	0.92	1.0	0.97	0.97	0.02
SVM	0.92	1.0	0.96	0.97	0.02
Random forest	0.88	0.97	0.95	0.95	0.03
Gradient Boosting	0.94	1.0	0.96	0.96	0.02
MLP	0.94	1.0	0.98	0.98	0.02

**Tab. 3:** Współczynnik  $F_1$  dla zbiorów walidacyjnych. Zielonym kolorem zaznaczono najlepsze wyniki w danej kolumnie.

Można zauważyć, że zarówno w przypadku celności predykcji, jak i wartości współczynnika  $F_1$  najlepszym algorytmem okazał się *Multilayer Perceptron*. W efekcie został on wybrany jako finalny model, dla którego w przypadku zbioru testowego otrzymano:

- $accuracy \approx 98.2\%$
- $F_1 \approx 0.98$

### 3.4 Zmiany skórne

Jako drugi zbiór danych zostały wybrane zmiany skórne w formacie obrazowym znajdujące się w zbiorze HAM10000 (*Human Against Machine 10000*) [53, 54]. Zadaniem postawionym w tym przypadku jest klasyfikacja obrazów. Cel ten zrealizowano wykorzystując sieć neuronową *Xception*. Zbiór zawiera 10015 zdjęć w formacie *jpg* o rozmiarach 600x450x3 (RGB), pochodzących z różnych źródeł na przestrzeni ostatnich dwudziestu paru lat. Pierwotne zbiory danych zostały zreorganizowane, oczyszczone, ustandaryzowane oraz zapisane we spójnym formacie, natomiast pochodzące z nich obrazy sklasyfikowane różnymi metodami (specjalisci, badania kontrolne, konsensus, badania mikroskopowe). Należy zauważyć, iż w zbiorze znajdują się zdjęcia które są naturalnie augmentowane tzn. istnieją grupy zdjęć które przedstawiają tę samą zmianę skórą u tego samego pacjenta jednak z innej perspektywy (inny kąt zdjęcia, odległość od skóry, rodzaj aparatu itp.), dlatego unikalnych zdjęć jest mniej niż wynosi liczność zbioru. Każde ze zdjęć ma przypisaną dokładnie jedną z siedmiu klas. System do klasyfikacji obrazów został oparty o bibliotekę Keras [27].

Klasa	Liczebność	Unikalna liczebność (liczba różnych przypadków)	Opis
akiec	327	228	Actinic Keratoses & Intraepithelial Carcinoma (Bowen's disease)
bcc	514	327	Basal cell carcinoma
bkl	1099	727	Benign keratosis
df	115	73	Dermatofibroma
mel	1113	614	Melanocytic nevi
nv	6705	6705	Melanoma
vasc	142	98	Vascular skin lesions
wszystkie	10015	8773	

Tab. 4: Prezentacja danych dotyczących zmian skórnego

### 3.4.1 Sieć neuronowa *Xception*

Niech  $\mathbf{W} \in \mathbb{R}^{n \times m \times k_h \times k_w}$  reprezentuje tensor jądra klasycznej konwolucji <sup>1</sup>, gdzie  $n$  oznacza liczbę jąder,  $m$  głębokość pojedynczego jądra, a  $k_h$  oraz  $k_w$  odpowiednio przestrzenną wysokość i długość jądra. Jako tensor wejściowy z kolei przyjmujemy  $\mathbf{x} \in \mathbb{R}^{m \times h \times w}$ , gdzie  $h, w$  oznaczają przestrzenną wysokość i długość tensora wejściowego. W wyniku operacji zwykłej konwolucji:

$$\mathbf{y} = \mathbf{W} * \mathbf{x} \quad (15)$$

otrzymujemy tensor  $\mathbf{y} \in \mathbb{R}^{n \times h' \times w'}$ , gdzie  $h', w'$  są nowymi wymiarami przestrzennymi. W szczególności dla ustalonego  $o \in [n]$  mamy:

$$y_o = \sum_{i=1}^m W_{o,i} * x_i \quad (16)$$

gdzie  $W_{o,i} = \mathbf{W}[o, i, :, :]$ ,  $x_i = \mathbf{x}[i, :, :]$  oraz  $y_o = \mathbf{y}[o, :, :]$ .

W przypadku sieci *Xception* (Chollet [55]) oprócz zwykłej konwolucji zastosowano *depthwise separable convolution* [55, 56], którą możemy zdefiniować jako złożenie konwolucji zwykłej typu *pointwise* z konwolucją typu *depthwise* (lub w odwrotnej kolejności – w przypadku omawianej sieci jest realizowany pierwszy scenariusz). Dla konwolucji *pointwise* jądro jest typu  $\mathbf{P} \in \mathbb{R}^{n \times m \times 1 \times 1}$  co stwierdza że, konwolucje tego typu wykonujemy de facto tylko po głębokości tensora wejściowego – stąd nazwa danej konwolucji. Po przeprowadzeniu *pointwise convolution* następuje konwolucja typu *depthwise* –  ${}_{DW}^*$ , którą możemy z kolei zdefiniować jako serię normalnych konwolucji na kolejnych kanałach tensora wejściowego z odpowiadającymi im dwuwymiarowymi jądrami:

$$\mathbf{y} = \mathbf{D}_{DW}^* \mathbf{x} \quad (17)$$

gdzie  $\mathbf{D} \in \mathbb{R}^{n \times 1 \times k_h \times k_w}$  przy czym  $n$  odpowiada  $n$  z *pointwise convolution* oraz:

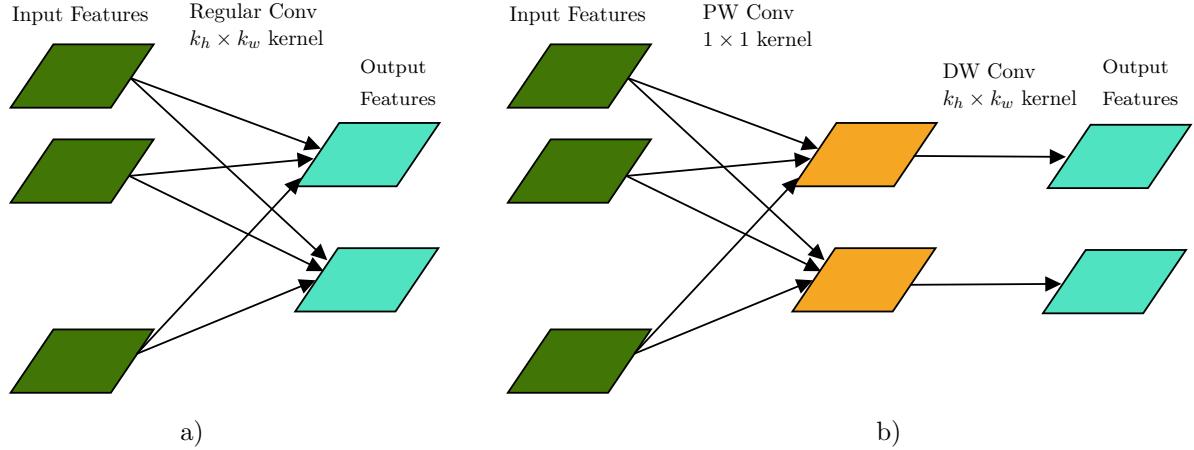
$$y_o = D_o * x_o \quad (18)$$

<sup>1</sup>Lub korelacji – ze względów praktycznych stosowanie operacji odwracania jadra często nie jest porzebne.

gdzie  $D_o = \mathbf{D}[o, 1, :, :]$ ,  $x_o = \mathbf{x}[o, :, :]$ .

W takim razie konwolucję zwykłą jak w (15) można zrealizować przez:

- wykonanie *pointwise convolution* –  $\mathbb{R}^{n \times h \times w} \ni \mathbf{y} = \mathbf{P} * \mathbf{x}$
- a następnie *depthwise convolution* –  $\mathbb{R}^{n \times h' \times w'} \ni \mathbf{z} = \mathbf{D}_{DW} * \mathbf{y}$

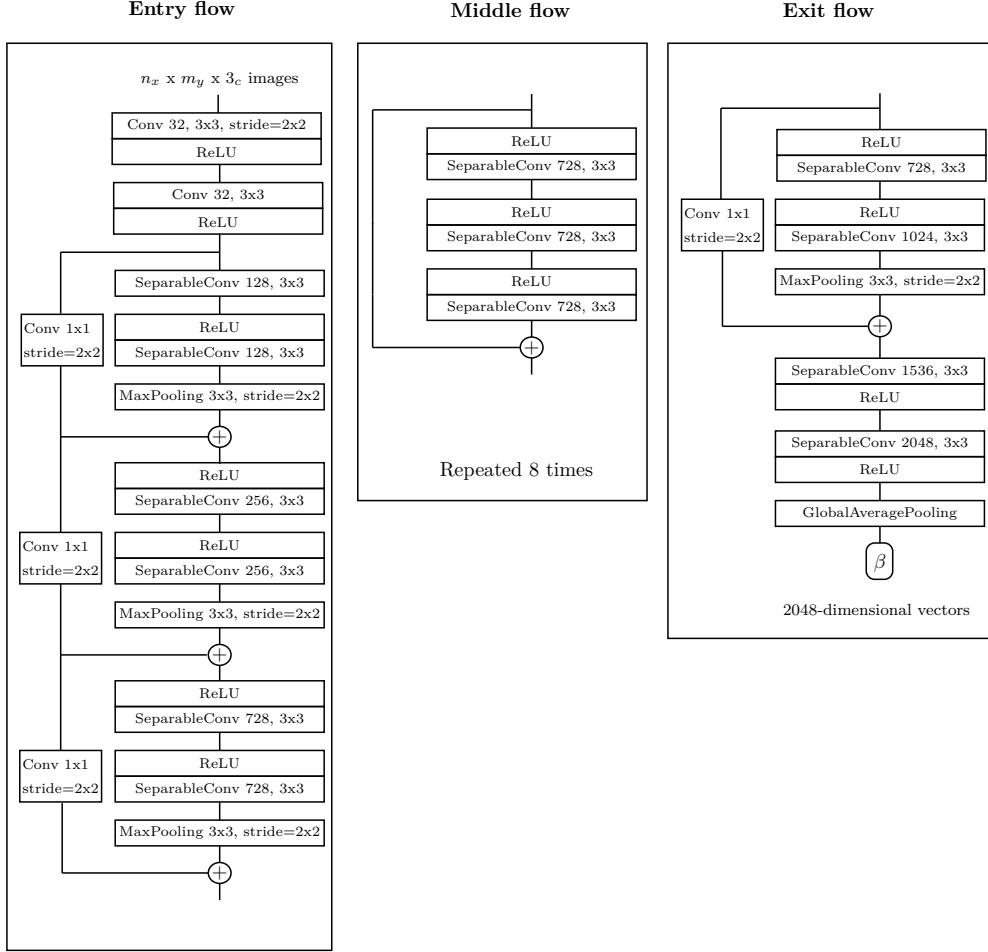


**Rys. 18:** Graficzne porównanie różnych typów konwolucji: a) – standardowa konwolucja, b) – *separable convolution* w wersji *pointwise* (PW) + *depthwise* (DW) [56].

Można pokazać, że powyższa sekwencja konwolucji ma mniejszą złożoność obliczeniową, co stanowi jej główną zaletę względem tradycyjnej konwolucji. Dzięki temu można zmniejszyć czas uczenia sieci oraz oczekiwania na odpowiedź lub zastosować większe konwolucje bez zmian czasu obliczeń. Z badań przeprowadzonych w [55, 56] wynika z kolei hipoteza o tym, że taka dekompozycja przynosi podobne lub lepsze rezultaty niż standardowa konwolucja. Dzięki tym czynom zdecydowano się na wykorzystanie i przetestowanie sieci *Xception* w niniejszej pracy w celu klasyfikacji danych obrazowych.

### Xception jako symboliczna sieć neuronowa:

$$\begin{aligned}
 & @ \rightarrow \mathcal{I}_{xy}^3 \mathbb{C}_{2\sigma 3}^{32} \mathbb{C}_3^{64} \left\langle \mathbb{C}_{2\sigma 1}^1 \mid {}_{DS}\mathbb{C}_3^{128} {}_{DS}\mathbb{C}_3^{128} {}_a\mathbb{P}_{2\sigma 3} \right\rangle \left\langle \mathbb{C}_{2\sigma 1}^1 \mid {}_{DS}\mathbb{C}_3^{256} {}_{DS}\mathbb{C}_3^{256} {}_a\mathbb{P}_{2\sigma 3} \right\rangle \\
 & \left\langle \mathbb{C}_{2\sigma 1}^1 \mid {}_{DS}\mathbb{C}_3^{728} {}_{DS}\mathbb{C}_3^{728} {}_a\mathbb{P}_{2\sigma 3} \right\rangle \left\langle \mathcal{R} {}_{DS}\mathbb{C}_3^{728} {}_{DS}\mathbb{C}_3^{728} {}_{DS}\mathbb{C}_3^{728} \right\rangle^8 \\
 & \left\langle \mathbb{C}_{2\sigma 1}^1 \mid \mathcal{R} {}_{DS}\mathbb{C}_3^{728} {}_{DS}\mathbb{C}_3^{1024} {}_m\mathbb{P}_{2\sigma 3} \right\rangle {}_{DS}\mathbb{C}_3^{1536} {}_{DS}\mathbb{C}_3^{2048} {}_g\mathbb{P} \mapsto \boxed{\beta} \\
 & \boxed{\stackrel{Xception}{\mathcal{NET}}(\alpha := n_x m_y 3_c)}
 \end{aligned}$$



**Rys. 19:** Sieć *Xception* na podstawie [55, 57]. Po wszystkich blokach Convolution oraz SeparableConvolution następuje batch normalization (niezaznaczone na rysunku).

Można zauważyć, że ze względu na operację *global average pooling* nie mają wpływu rozmiary przestrzenne (wzdłuż osi  $x$  i  $y$ ) tensora wejściowego  $\alpha$ . Jedynym wymaganym warunkiem jest istnienie trzech kanałów.

### 3.4.2 Sieć neuronowa *Xception*\*

Sieć *Xception*\* stanowi połączenie sieci *Xception* z warstwami znajdującymi się za *Global Averaging Pooling* oraz odpowiednim wyborem hiperparametrów, umożliwiające elastyczne dostosowanie jej do zadanego problemu. W przypadku postawionego problemu klasyfikacji zdecydowano się na wykorzystanie kolejno:

- warstwy typu *Dropout* z prawdopodobieństwem wyzerowania połączenia równym 0.5
- warstwą typu *FullyConnected* z 7 neuronami oraz brakiem funkcji aktywacji (odpowiada to

liczbie dostępnych klas)

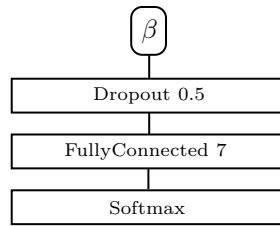
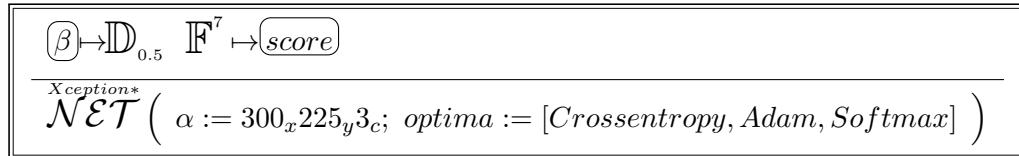
- funkcją typu *Softmax*

Jako hiperparametry zostały z kolei wybrane:

- funkcja strat – categorical crossentropy
- inicjalizacja wag – *ImageNet* [58]
- optymalizacja – *Adam* z początkowym współczynnikiem szybkości uczenia równym  $10^{-5}$
- *ReduceLROnPlateau* – jeśli przez dwie kolejne epoki nie nastąpi poprawa *accuracy* na zbiorze walidacyjnym, to aktualny współczynnik szybkości uczenia zostaje zmniejszony dwukrotnie (z minimalną możliwą do uzyskania wartością równą  $10^{-6}$ )
- rozmiar batch-a – 16
- liczba epok – 20

Następnie cały model jest poddawany procesowi uczenia.

#### Rozszerzenie architektury *Xception*:



**Rys. 20:** Fragment końcowy sieci *Xception*\*

#### 3.4.3 Preprocessing danych

Preprocessing danych jest wykonywany w następujących krokach:

1. zmiana rozmiarów obrazów: 600x450x3  $\rightarrow$  300x225x3
2. augmentacja obrazów w zbiorze trenującym w czasie rzeczywistym na podstawie dostępnego aktualnie batch-a:
  - $rotation\_range = 180$

- $width\_shift\_range = 0.2$
- $height\_shift\_range = 0.2$
- $zoom\_range = 0.2$
- $shear\_range = 0.2$
- $horizontal\_flip = True$
- $vertical\_flip = True$
- $fill\_mode = 'nearest'$

3. skalowanie wartości pikseli

$$pixel\_val\_norm = \frac{pixel\_val}{127.5} - 1$$

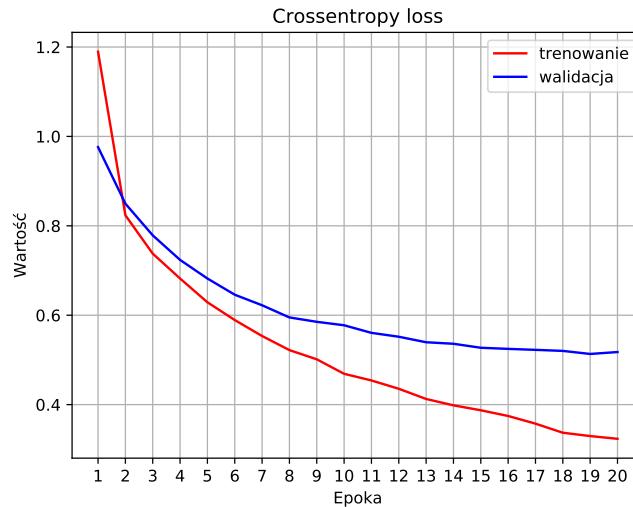
Postanowiono dodatkowo zbadać wpływ augmentacji wykonanej w miejscu, tj. przed rozpoczęciem procesu uczenia, aby porównać jej efekt z augmentacją wykonywaną w czasie rzeczywistym. Zostało to zrealizowane w następujących podpunktach.

#### 3.4.4 Procedury uczenia i testowania

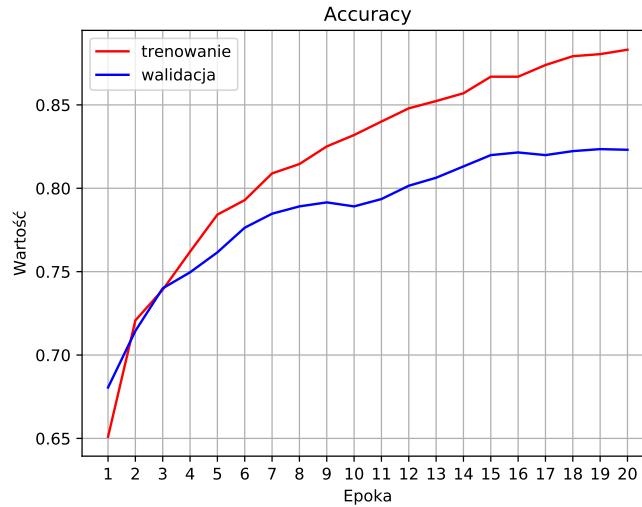
Proces uczenia i testowania został zrealizowany w następujący sposób:

1. podział na zbiór uczący (75 %) oraz walidacyjny (testowy) (25 %)
2. jeśli w zbiorze treningowym znajdują się naturalnie zaugmentowane kopie obrazów ze zbioru walidacyjnego, zostają one wyrzucone, aby zapobiec przeklejowi danych
3. zastosowanie preprocessingu danych

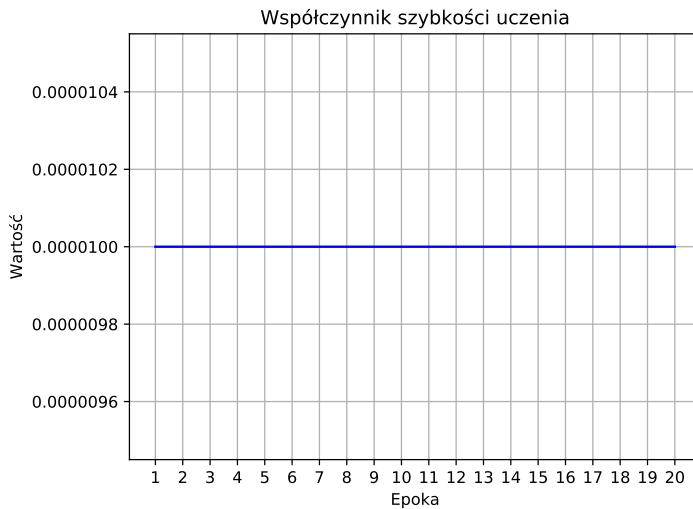
#### 3.4.5 Augmentacja w czasie rzeczywistym



Rys. 21: Funkcje strat dla zbioru trenującego i walidacyjnego



Rys. 22: Funkcje celności predykcji dla zbioru trenującego i walidacyjnego

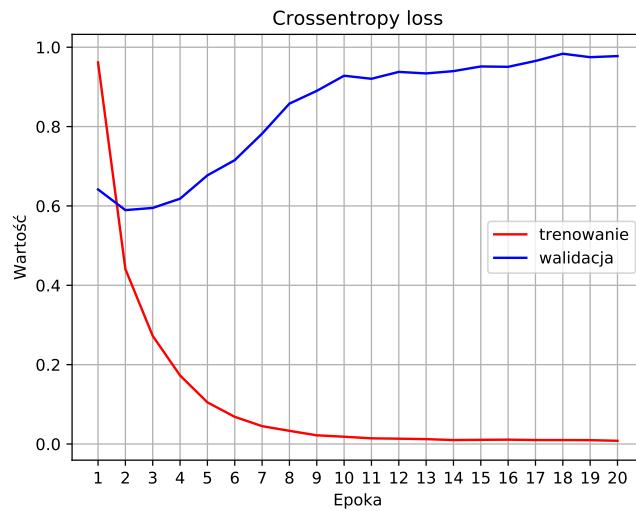


Rys. 23: Funkcja wartości współczynnika szybkości uczenia

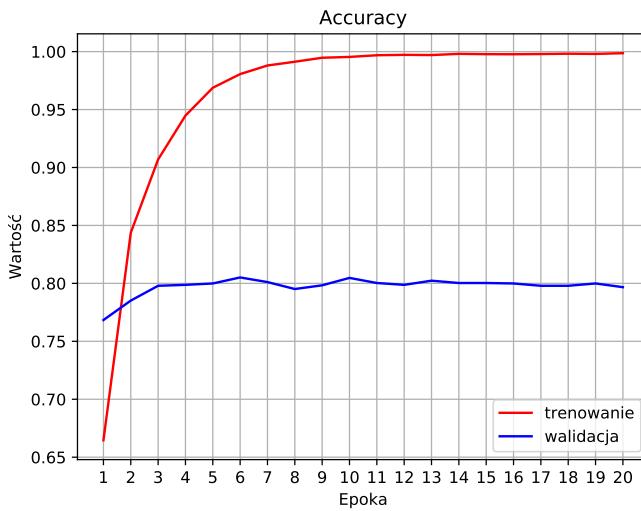
Można zaobserwować łagodny i efektywny przebieg uczenia oraz brak zmian wartości współczynnika szybkości uczenia co oznacza, że podczas całego procesu następował wzrost *accuracy* co najmniej co dwie kolejne epoki.

### 3.4.6 Augmentacja w miejscu

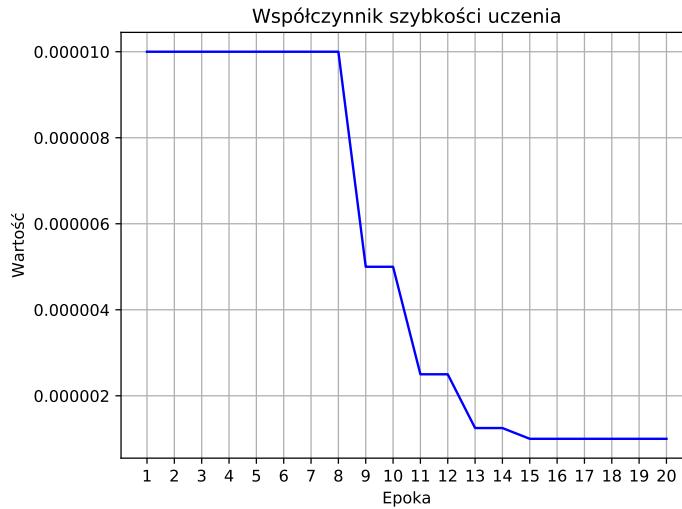
Augmentacja w miejscu została przeprowadzona przed procesem uczenia i spowodowała zwiększenie liczby obrazów w zbiorze trenującym dla każdej klasy do wartości ok 4500 (w przybliżeniu tyle wynosił rozmiar najliczniejszej klasy przed augmentacją). W trakcie uczenia obrazy nie były już augmentowane.



Rys. 24: Funkcje strat dla zbioru trenującego i walidacyjnego



Rys. 25: Funkcje celności predykcji dla zbioru trenującego i walidacyjnego



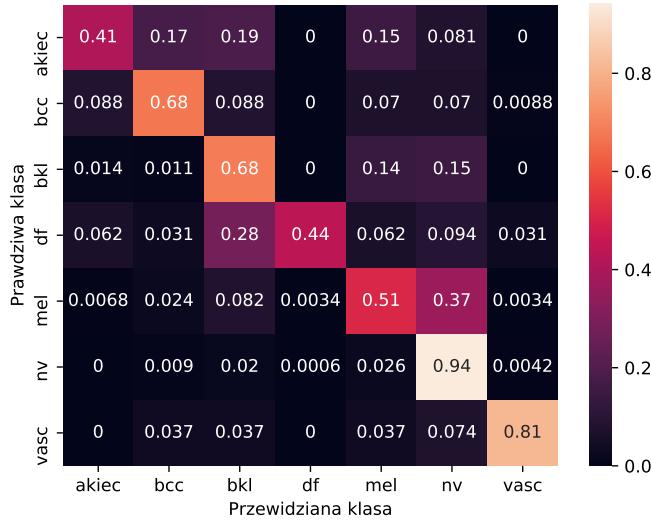
Rys. 26: Funkcja wartości współczynnika szybkości uczenia

W przypadku funkcji strat, jak również *accuracy* można zaobserwować klasyczne przykłady przeuczenia modelu już po kilku epokach. Mimo zmniejszania wartości współczynnika szybkości uczenia, nie udało się osiągnąć zadawalających rezultatów. Za tę sytuację odpowiada najprawdopodobniej fakt silnego niezrównoważenia klas. Klasy bardzo mało liczne zostają augmentowane do bardzo dużej wartości, w efekcie czego otrzymujemy dużą ilość obrazów różniących się tylko małymi przekształceniami afinycznymi, które następnie w każdej epoce podczas uczenia nie podlegają już zmianom. W przypadku augmentacji w czasie rzeczywistym, jej działanie ogranicza się do danego batcha w danej epoce, więc nawet podobne zestawy obrazów wnoszą dodatkową informację między kolejnymi epokami.

#### 3.4.7 Szersza analiza wyników walidacji dla augmentacji w czasie rzeczywistym

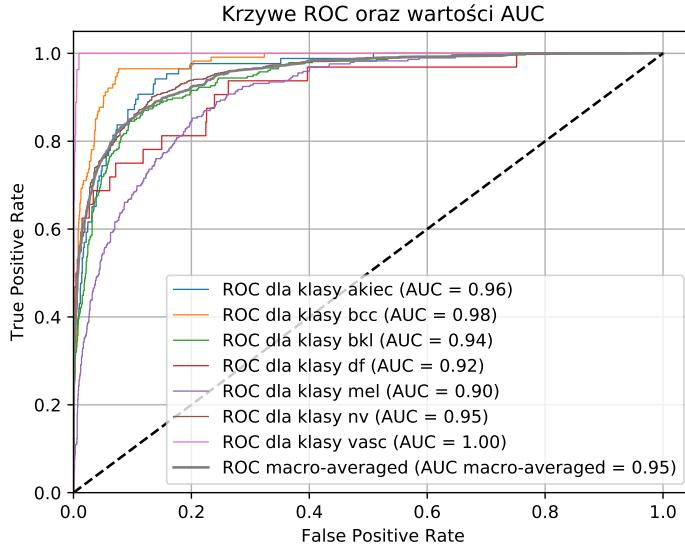
Jako finalne metryki zostały otrzymane:

- $accuracy \approx 82.3\%$
- $F_1 \text{ macro} \approx 0.66$



Rys. 27: Macierz pomyłek dla zbioru walidacyjnego

Na Rys. 27 można zaobserwować klasy, dla których model przewidywał najlepiej (*nv*, *vasc*) oraz najgorzej (*akiec* mylony z różnymi innymi klasami, *df* mylony najczęściej z *bkl* oraz *mel* mylony najczęściej z *nv*).



Rys. 28: Krzywe ROC i wartości AUC dla zbioru walidacyjnego (metoda *one vs all*)

Porównując z kolejnymi informacjami zawartymi na Rys. 28 z wartościami *accuracy* oraz *F<sub>1</sub> macro* możemy zaobserwować, że bardzo dobre przebiegi ROC oraz wartości AUC nie gwarantują odpowiednio dobrych wartości innych metryk.

## 4 Opis projektu

Poniżej przedstawiono skrótowy opis poszczególnych części projektu od strony powstałego oprogramowania. Więcej informacji znajduje się w udokumentowanym kodzie.

- *Multimodal-ML-System-Models* – modele uczenia maszynowego:
  - *breast\_cancer* – rak piersi:
    - \* *data* – dane o raku piersi
    - \* *models* – budowa i porównanie różnych modeli
    - \* *model* – zapisany, finalny model
  - *skin* – zmiany skórne:
    - \* *data* – dane oryginalne i przetworzone
    - \* *trainer* – budowa, uczenie i testowanie sieci neuronowej *Xception*\*
    - \* *stats* – zapisane statystyki
    - \* *model* – zapisany, finalny model
    - \* *augmentation\_on\_begining* – program do augmentacji w miejscu
- *Multimodal-ML-System-Server* – główny serwer
- *Multimodal-ML-System-Services* – serwisy główne i pomocnicze systemu:
  - *database* – skrypt SQL do zakładania bazy danych i wypełnienia jej przykładowymi danymi
  - *hashing* – hashowanie przykładowych haseł użytkowników z użyciem procedury *bcrypt*
  - *services* – główne serwisy systemu (oprócz *Multimodal-ML-System-Server*):
    - \* *composition* – kompozycja systemu z użyciem Docker Compose
    - \* *ml-services/breast-cancer* – autorski serwis do obsługi modelu z scikit-learn
    - \* *proxy\_server* – serwis pośredniczący

## 5 Podsumowanie

W niniejszej pracy z powodzeniem został zrealizowany system wspomagający lekarzy i techników w diagnozie chorób oraz zmian patologicznych w ramach celów postawionych w pierwszej części pracy. Jako najważniejsze aspekty można uznać:

- analizę i dobór narzędzi do budowy systemu
- budowę wielomodułowego systemu, w którym większość komponentów jest oparta o Dockera co zwiększa ich przenośność i ułatwia implementację w różnych środowiskach
- wybór przykładowych danych do uczenia maszynowego, ich analizę oraz dobór algorytmów uczenia maszynowego:
  - w przypadku raka piersi zostało porównanych sześć algorytmów, dla których przeprowadzono 10-krotną walidację krzyżową do analizy statystyk otrzymanych wartości celności predykcji oraz współczynnika  $F_1$ . Jako zwycięzcę wytypowano wielowarstwowy perceptron, co zgadzało się z teoretycznymi intuicjami posiadanymi przed porównaniem modeli (typowano, że najgorszy okaże się naiwny Bayes, a najlepszy Gradient Boosting lub wielowarstwowy perceptron, co znalazło potwierdzenie w wynikach badań). Następnie dla zwycięskiego modelu przeprowadzono finalne uczenie i testowanie na całości danych.
  - dla zmian skórnego dobrano konwolucyjną sieć neuronową *Xception*, która poddano modyfikacjom na potrzeby pracy. Za jej wyborem stała z jednej strony umiarkowanie skomplikowana architektura, z drugiej zastosowanie nowego rodzaju konwolucji pozwalającego na efektywniejsze uczenie sieci i otrzymanie potentjalnie dobrych rezultatów. W tej części przeprowadzono także autorskie badanie dwóch różnych typów augmentacji danych. Do analizy sieci wykorzystano metryki jakości takie jak wartości funkcji strat, celność predykcji, współczynnik  $F_1$  z makrouśrednianiem, krzywe ROC, współczynnik AUC czy macierz pomyłek.
- prezentację systemu w usłudze *Google Cloud Platform*

Praca ma potencjał rozwojowy głównie poprzez:

- zastosowanie natywnych rozwiązań danego dostawcy usług chmurowych w celu zapewnienia większej wydajności oraz skalowalności – w przypadku Google Cloud oznacza to np.:
  - zastosowanie do serwowania zbudowanych obrazów Dockera serwisów *Kubernetes*[59] lub *App Engine*[60] w celu dynamicznego skalowania aplikacji
  - zastosowanie jako stałego magazynu danych *Google Cloud Storage*[23] zamiast dysku twardego w Google Cloud Engine
  - instalacja bazy danych w środowisku *Cloud SQL*[61]
  - zastosowanie zamiast Docker Compose natywnej architektury sieciowej Google Cloud

- dla serwisów opartych o Flask zastosowanie konwencji *Web Server Gateway Interface (WSGI)*[62] zamiast wykorzystywanej w niniejszej pracy wielowątkowości, w celu zapewnienia większej niezawodności i wydajności przy obsłudze wielu żądań
- zastosowanie specjalizowanej obsługi wyjątków – w chwili obecnej generyczna obsługa wyjątków zapewnia zwrot odpowiedniej informacji do użytkownika, jednak nie określa dokładnego powodu niepowodzenia, nawet jeśli jest to wino użytkownika (np. niespełnienie wymagań odnośnie przekazywanych przykładów do analizy)
- dodanie serwisów umożliwiających częściowe dopasowanie teoretycznie niepoprawnie zdefiniowanych przykładów np. obrazy o złych rozmiarach mogłyby zostać dostosowane do wymagań systemu automatycznie
- dodanie systemu logów
- zastosowanie połączenia *VPN*[63] zamiast publicznego adresu

W momencie oddawania pracy jedynym niezrealizowanym, aczkolwiek pierwotnie planowanym celem jest zastosowanie algorytmów uczenia maszynowego do trzeciego, przykładowego typu danych – analizy szeregu czasowego. Niestety, ze względów czasowo – finansowych nie zostało to wykonane.

## 6 Bibliografia

- [1] Wikipedia, *Representational state transfer*: [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)
- [2] Wikipedia, *Hypertext transfer protocol*: [https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)
- [3] Russell Aaron, ssl.com, *What is SSL ?*: <https://www.ssl.com/faqs/faq-what-is-ssl/>
- [4] Zohra Fatema, Nascenia, *Why cross site scripting is determinal and how to prevent ?* : <https://www.nascenia.com/why-cross-site-scripting-is-detrimental-and-how-to-prevent/>
- [5] Imperva, *Cross site request forgery (CSRF) attack*: <https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/>
- [6] Wikipedia, *Internet protocol suite*: [https://en.wikipedia.org/wiki/Internet\\_protocol\\_suite](https://en.wikipedia.org/wiki/Internet_protocol_suite)
- [7] Wikipedia, *URL*: <https://en.wikipedia.org/wiki/URL>
- [8] Mozilla, *HTTP – wiadomości ogólne*: [https://developer.mozilla.org/pl/docs/Web/HTTP/HTTP\\_wiadomosci\\_ogolne](https://developer.mozilla.org/pl/docs/Web/HTTP/HTTP_wiadomosci_ogolne)
- [9] Russell Aaron, ssl.com, *What is an X.509 Certificate ?*: <https://www.ssl.com/faqs/what-is-an-x-509-certificate/>
- [10] Wikipedia, *JSON*: <https://en.wikipedia.org/wiki/JSON>
- [11] Wikipedia, *Ajax (programming)*: [https://en.wikipedia.org/wiki/Ajax\\_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming))
- [12] louis2anderson, *AJAX Diagram*: <http://louis2anderson.blogspot.com/2011/05/ajax-diagram.html>
- [13] docker.com: <https://www.docker.com/>
- [14] Wilson Bibin, devopscube, *What is Docker ? How Does it Work ?*: <https://devopscube.com/what-is-docker/>
- [15] Janetakis Nick, nickjanetakis.com, *Understanding How the Docker Daemon and Docker CLI Work Together*: <https://nickjanetakis.com/blog/understanding-how-the-docker-daemon-and-docker-cli-work-together>
- [16] Wikipedia, *Cloud computing*: [https://en.wikipedia.org/wiki/Cloud\\_computing](https://en.wikipedia.org/wiki/Cloud_computing)
- [17] aws.amazon.com: <https://aws.amazon.com/>
- [18] azure.microsoft.com: <https://azure.microsoft.com/en-us/>
- [19] cloud.google.com: <https://cloud.google.com/>
- [20] cloud.google.com, *Compute Engine*: <https://cloud.google.com/compute/>
- [21] Wikipedia, *Google Compute Engine*: [https://en.wikipedia.org/wiki/Google\\_Compute\\_Engine](https://en.wikipedia.org/wiki/Google_Compute_Engine)
- [22] Wikipedia, *Infrastructure as a service*: [https://en.wikipedia.org/wiki/Infrastructure\\_as\\_a\\_service](https://en.wikipedia.org/wiki/Infrastructure_as_a_service)

- [23] cloud.google.com, *Cloud Storage*, <https://cloud.google.com/storage/>
- [24] developer.nvidia.com, <https://developer.nvidia.com/cuda-zone>
- [25] docs.docker.com, *Overview of Docker Compose*: <https://docs.docker.com/compose/>
- [26] mysql.com: <https://www.mysql.com/>
- [27] tensorflow.org, *Keras overview*: <https://www.tensorflow.org/guide/keras/overview>
- [28] tensorflow.org: <https://www.tensorflow.org/>
- [29] tensorflow.org, *Serving Models*: <https://www.tensorflow.org/tfx/guide/serving>
- [30] Wikipedia, *GRPC*: <https://en.wikipedia.org/wiki/GRPC>
- [31] scikit-learn.org: <https://scikit-learn.org/stable/>
- [32] xgboost.readthedocs.io, *XGBoost Documentation*: <https://xgboost.readthedocs.io/en/latest/>
- [33] flask.palletsprojects.com: <http://flask.palletsprojects.com/en/1.1.x/>
- [34] scala-lang.org: <https://www.scala-lang.org/>
- [35] playframework.com: <https://www.playframework.com/>
- [36] Margam Girish, medium.com, *Model-View-Controller (MVC)*: <https://medium.com/datadriveninvestor/model-view-controller-mvc-75bcb0103d66>
- [37] akka.io: <https://akka.io/>
- [38] Wikipedia, *Bcrypt*: <https://en.wikipedia.org/wiki/Bcrypt>
- [39] scala-slick.org: <http://scala-slick.org/>
- [40] www.sqlalchemy.org: <https://www.sqlalchemy.org/>
- [41] Wikipedia, *Base64*: <https://en.wikipedia.org/wiki/Base64>
- [42] Wikipedia, *ASCII*: <https://en.wikipedia.org/wiki/ASCII>
- [43] jquery.com: <https://jquery.com/>
- [44] Wikipedia, *Single-page application*: [https://en.wikipedia.org/wiki/Single-page\\_application](https://en.wikipedia.org/wiki/Single-page_application)
- [45] Skarbek Władysław, *Symbolic Tensor Neural Networks for Digital Media – from Tensor Processing via BNF Graph Rules to CREAMS Applications*, Fundamenta Informaticae, vol. 168, no. 2-4, pp. 89-184, 2019: <https://content.iospress.com/journals/fundamenta-informaticae/168/2-4>, <https://arxiv.org/abs/1809.06582>
- [46] UCI Machine Learning Repository, *Breast Cancer Wisconsin (Diagnostic) Data Set*: <https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>
- [47] Cichosz Paweł, podsumowanie wykładów z *Metody odkrywania wiedzy*: <http://elektron.elka.pw.edu.pl/~pcichosz/mow/>
- [48] Wikipedia, *Naive Bayes classifier*: [https://en.wikipedia.org/wiki/Naive\\_Bayes\\_classifier](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)

- [49] ml-cheatsheet.readthedocs.io, *Logistic Regression*: [https://ml-cheatsheet.readthedocs.io/en/latest/logistic\\_regression.html](https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html)
- [50] Swaminathan Saishruthi, towardsdatascience.com, *Logistic Regression — Detailed Overview*: <https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc>
- [51] Wikipedia, *Support-vector machine*: [https://en.wikipedia.org/wiki/Support-vector\\_machine](https://en.wikipedia.org/wiki/Support-vector_machine)
- [52] Cichosz Paweł, *Systemy Uczące się*, Wydanie Drugie, Wydawnictwo Naukowo-Techniczne, Warszawa 2000, 2007
- [53] Tschandl Philipp, Rosendahl Cliff, Kittler Harald, *The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions*, 2018: <https://arxiv.org/abs/1803.10417>
- [54] Harvard Dataverse, *The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions*: <https://dataVERSE.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T>
- [55] Chollet François, *Xception: Deep Learning with Depthwise Separable Convolutions*, 2016: <https://arxiv.org/abs/1610.02357>
- [56] Guo Jianbo, Li Yuxi, Lin Weiyao, Chen Yurong, Li Jianguo, *Network Decoupling: From Regular to Depthwise Separable Convolutions*, 2018: <https://arxiv.org/abs/1808.05517>
- [57] keras.io, *Xception*: <https://keras.io/applications/#xception>
- [58] image-net.org: <http://www.image-net.org/>
- [59] cloud.google.com, *Google Kubernetes Engine*: <https://cloud.google.com/kubernetes-engine/>
- [60] cloud.google.com, *App Engine*: <https://cloud.google.com/appengine/>
- [61] cloud.google.com, *Cloud SQL*: <https://cloud.google.com/sql/>
- [62] Wikipedia, *Web Server Gateway Interface*: [https://en.wikipedia.org/wiki/Web\\_Server\\_Gateway\\_Interface](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface)
- [63] Wikipedia, *Virtual private network*: [https://en.wikipedia.org/wiki/Virtual\\_private\\_network](https://en.wikipedia.org/wiki/Virtual_private_network)