

**ТЕХНОЛОГИЧНО УЧИЛИЩЕ “ЕЛЕКТРОННИ СИСТЕМИ”  
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Платформа за анализ на медицински изображения

Дипломант:

*Илия Първанов*

Научен ръководител:

*инж. Росен Недялков*

СОФИЯ

2020



# Увод

През последните години в сферата на информационните технологии нашумя терминът машинно самообучение, който е част от сферата на изкуствения интелект. Машинното самообучение е дисциплина, която изучава алгоритми, които изграждат математически модели на базата на голямо количество данни (наричани данни за обучение), за да правят прогнози (предположения) или да взимат решения, без да са изрично програмирани за това. Причините за стремглавото развитие на тази програмна концепция са значителните иновации в топологиите и методите за трениране на дълбоки изкуствени невронни мрежи (deep artificial neural networks, един от множеството алгоритми за машинно самообучение; оттук нататък наричани невронни мрежи), бързото напредване на изчислителната мощност на хардуера и възможността за събиране на все по-големи количества информация. Невронните мрежи към момента са незаменими при анализиране и извличане на знания от изображения и текст.

Едно следствие от бързото развитие на алгоритмите за машинно самообучение е тяхното навлизане в множество сфери на човешкия живот - селското стопанство, търговията, фотографията, правораздаването, организацията на бизнеса и тн. Въвеждането в експлоатация на алгоритми за машинно самообучение във всички тези области би трябвало да помогне на човека да върши работата си по по-ефективен и безпроблемен начин. А една конкретна област, в която оптимизирането на човешката работа е от животоспасяващо значение, е медицината и по-специфично радиологията. Радиологията е клон на медицината, който се занимава със създаването на медицински изображения с цел диагностициране и третиране на вътрешни болести в хора или животни.

Един от сериозните проблеми в тази област е човешката грешка. Различните изследвания достигат до различни изводи по отношение на точните цифри - според едно от тях всяка година в световен план се допускат грешки в около 4% от радиологичните изследвания, което прави 40 милиона грешно поставени диагнози. При много от тези случаи грешката се оказва фатална. Въпреки десетилетията изследвания, медицински и технически прогрес, тази статистика е останала почти непроменена през последните 60 години.

Но защо се допускат толкова много грешки? Факторите са многобройни - умора, разсейване, липса на достатъчно време вследствие на голямото количество пациенти, както и един интересен феномен, наречен 'satisfaction of search'. Този феномен се изразява в това, че след намиране на една аномалия в изображение, е твърде вероятно радиологът да не идентифицира други, тъй като е удовлетворен от работата си и бърза да премине към следващото изследване.

Един компютърен алгоритъм не би се сблъскал с нито един от гореизброените проблеми, но и не притежава годините опит, знания и способността да прави сложни разсъждения. Целта на настоящата дипломна работа е да се използват максимално силните страни на човешкия ум и на изкуствения интелект, за да бъде намален шансът за допускане на грешка при интерпретиране на радиологични изображения.

Целите на настоящата дипломна работа са:

- Изграждане на уеб платформа за съхранение и анализ на медицински изследвания, която да може да се ползва от болници и лекари
- Получаване на помощ при диагностициране под формата на прогнози за вероятни диагнози
- Възможност за бързо и безопасно плащане през платформата за получаване на пълния набор от функционалности

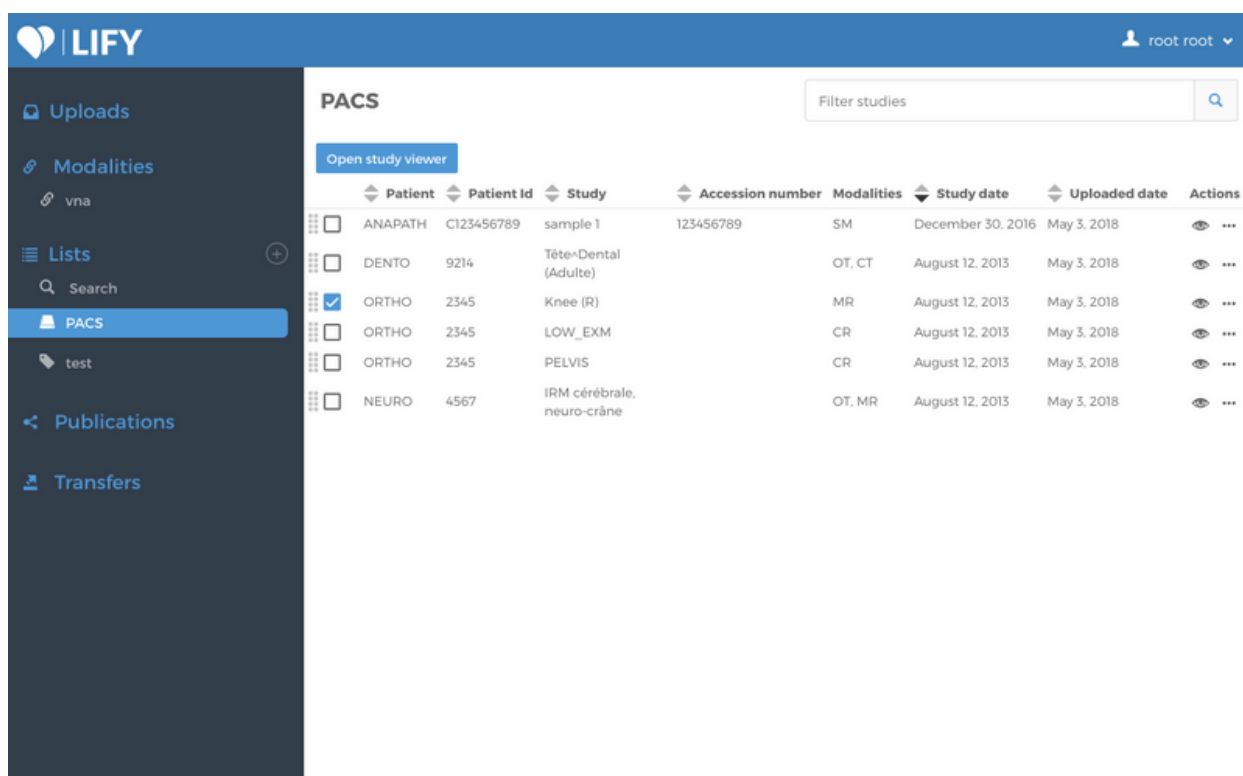
# Първа глава

## 1.1 Обзор на подобни продукти

### 1.1.1 Системи за управление на медицински изследвания

#### 1.1.1.1 Lify

Lify е уеб система за автоматизация на съхранението на медицински изображения и сътрудничество между специалисти. Позволява сигурно споделяне на медицински изследвания между институции по имейл или чрез генериран URL. Предоставя също и възможност за преглеждане и анализ на множество видове медицински изследвания, вкл. КТ, ЯМР, 4D и други, посредством интегриран от трети страни софтуер.



The screenshot shows the Lify PACS interface. On the left is a dark sidebar with navigation options: Uploads, Modalities (vna), Lists (Search, PACS, test), Publications, and Transfers. The main area is titled 'PACS' and contains a table of studies. A search bar 'Filter studies' is at the top right. A button 'Open study viewer' is above the table. The table has columns: Patient, Patient Id, Study, Accession number, Modalities, Study date, Uploaded date, and Actions. The 'ORTHOD' study is selected with a blue checkbox.

	Patient	Patient Id	Study	Accession number	Modalities	Study date	Uploaded date	Actions
<input type="checkbox"/>	ANAPATH	C123456789	sample 1	123456789	SM	December 30, 2016	May 3, 2018	👁️ ...
<input type="checkbox"/>	DENTO	9214	Tête«Dental (Adulte)		OT, CT	August 12, 2013	May 3, 2018	👁️ ...
<input checked="" type="checkbox"/>	ORTHOD	2345	Knee (R)		MR	August 12, 2013	May 3, 2018	👁️ ...
<input type="checkbox"/>	ORTHOD	2345	LOW_EXM		CR	August 12, 2013	May 3, 2018	👁️ ...
<input type="checkbox"/>	ORTHOD	2345	PELVIS		CR	August 12, 2013	May 3, 2018	👁️ ...
<input type="checkbox"/>	NEURO	4567	IRM cérébrale, neuro-crâne		OT, MR	August 12, 2013	May 3, 2018	👁️ ...

Фигура 1.1 – Преглед на списък от изследвания в Lify

### **1.1.1.2 AbbaDox Rad**

AbbaDox Rad е софтуерна система за радиолози, чиято цел е да намали разходите за обработка и съхраняване на изследвания и да увеличи ефективността на работата на радиолозите. Програмата поддържа споделяне на изображения и диагнози и позволява на телерадиологични институции и радиологични центрове да се справят с критични оперативни проблеми без големи финансови инвестиции.

### **1.1.1.3 QMENTA Trials**

QMENTA Trials е част от QMENTA платформата, чиито целеви клиенти са лаборатории и институции, занимаващи се с разработка на нови лекарства и фармацевтични продукти. QMENTA Trials позволява гъвкаво съхранение на данни от множество изследвания, както и лесен трансфер на данните. Освен това дава възможност за съхраняване на подробна история на състоянието на пациента. Системата разчита на многообразие от алгоритми за автоматично създаване на обобщени доклади и извличане на полезни знания от изследванията.

### **1.1.1.4 Merge PACS**

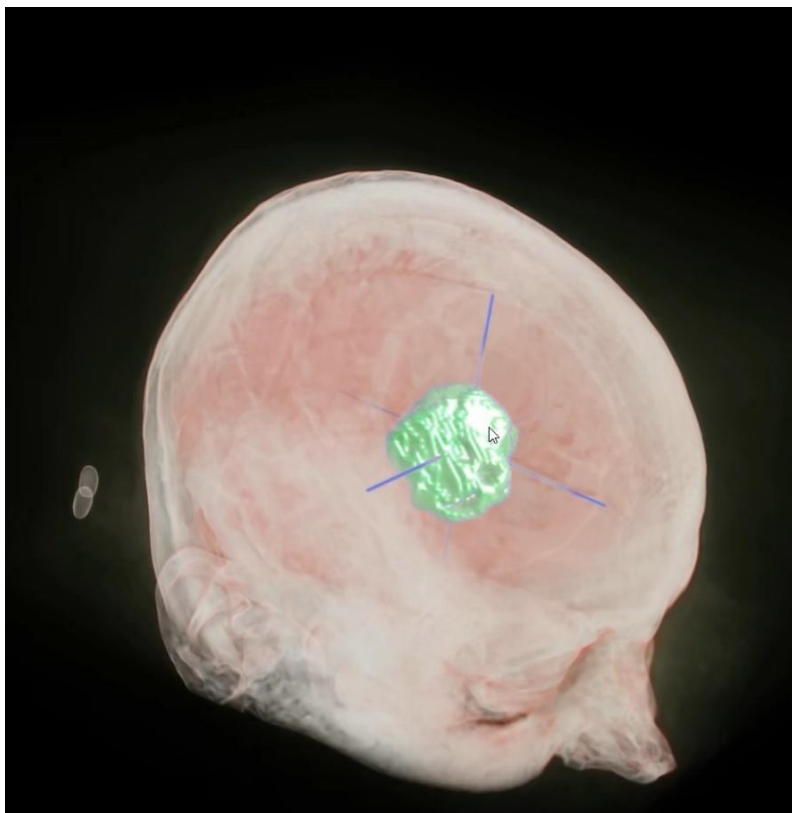
Merge PACS решава следния проблем: в едно голямо медицинско заведение обикновено специалистите са принудени да преглеждат резултати и изображения от изследвания на поне няколко работни станции, които използват различни софтуерни системи. Това затормозява персонала, забавя провеждането на изследвания и увеличава разходите за поддръжка на системите. Целта на Merge PACS е да бъде единствената платформа, до която специалистите в едно заведение се докосват. Merge PACS съчетава възможността за провеждане и съхранение на различни по вид изследвания, както и лесното им управление. Платформата също предоставя

допълнителни инструменти за извличане на статистики и знания от данните.

## **1.1.2 Системи за анализ на изображения от медицински изследвания**

### **1.1.2.1 Project InnerEye**

InnerEye е изследователски проект на Microsoft, имплементиращ различни техники за машинно самообучение с цел идентифициране на тумори в тримерни радиологични изображения. При подготовка за лечение на тумори на специалистите се налага ръчно да маркират зоните, в които е установен туморът. Този метод е неточен и отнема много време. Софтуерът на Microsoft InnerEye автоматично може да маркира тези зони, както и да маркира здравите органи около тумора, които трябва да бъдат предпазени от лъчите, с които се поразяват и отстраняват туморите.



*Фигура 1.2 Автоматично маркирана от InnerEye зона на тумор в тримерно изображение, получено чрез ядрено-магнитен резонанс*

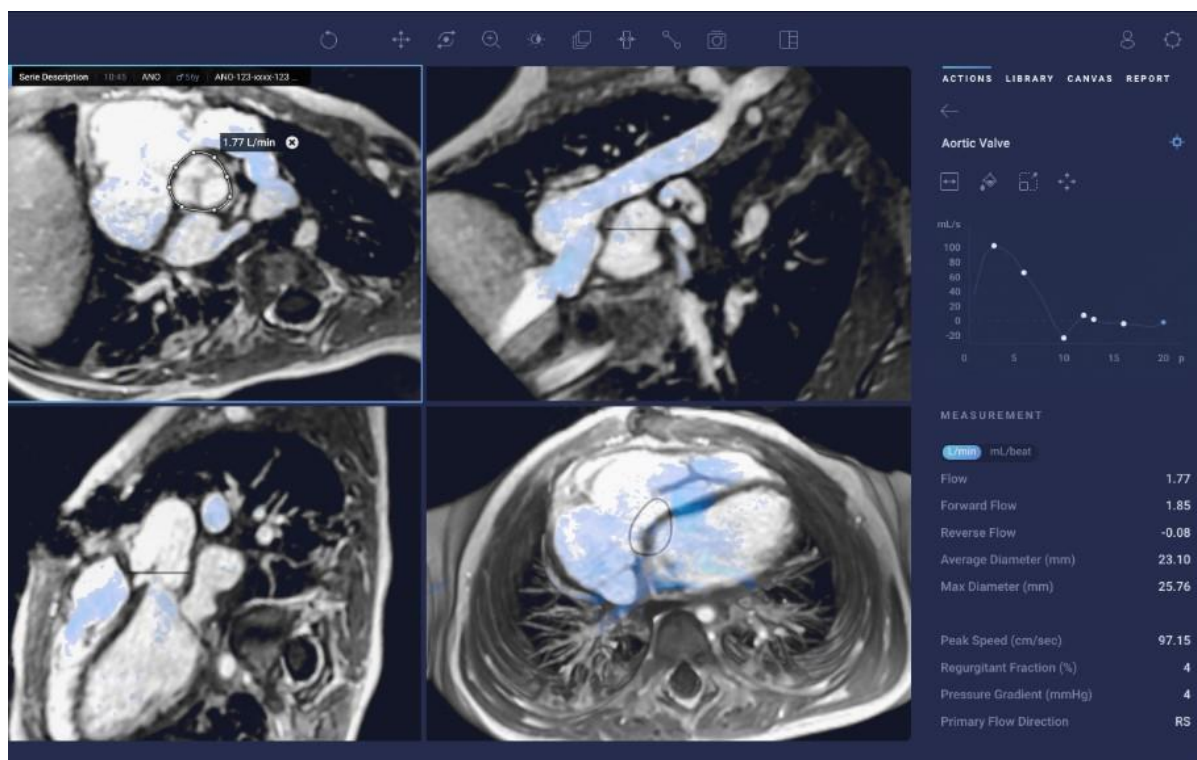
### **1.1.2.2 ProFound AI**

ProFound AI е софтуерно решение, базирано на алгоритми за машинно самообучение, което бързо анализира томосинтезни изображения за наличието на ракови клетки, и отчита процентна вероятност за наличието на такива клетки. Софтуерът се изпълнява на сървърната платформа PowerLook, която разчита на графични ускорители на Nvidia за бързо изпълнение на алгоритмите.

### **1.1.2.3 Arterys**

Arterys е интернет платформа за съхранение и анализ на медицински изображения. Това, което отличава Arterys, са усъвършенстваните методи за анализ на изображения от сърдечносъдови и белодробни изследвания. Създателите на Arterys твърдят, че броят на пропуснати аномалии спада с между 42% и 70% в здравните заведения, където софтуерът е въведен в експлоатация. Още едно предимство, което те посочват, е модерният, интуитивен и лесен за навигация потребителски интерфейс.





Фигура 1.3 Преглед на изследване в Arterys

## 1.2 Обзор на начини и технологии за реализиране на уеб приложение

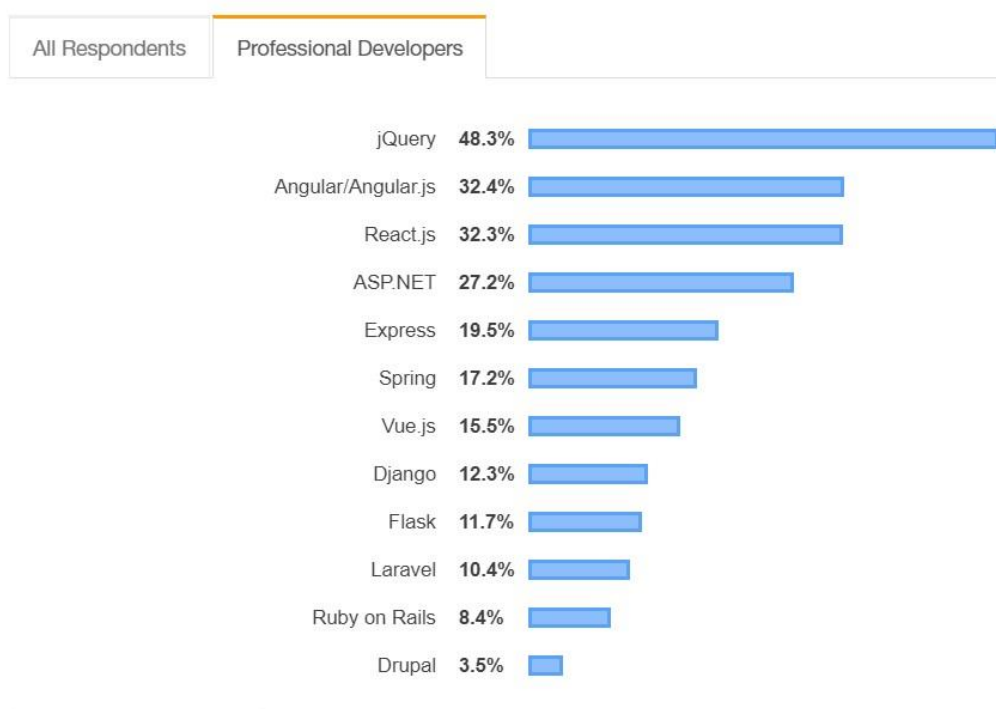
Най-често едно уеб приложение се разделя на сървърна част (back-end) и клиентска част (front-end), като за всяка от тях изборът на технологии е ключов, тъй като от него зависи колко лесно приложението ще се надгражда, променя и поддържа, и дали ще може да се пригоди за едновременно използване от много голям брой потребители. Най-гъвкавата, но и най-времеемка опция е да се започне от най-базовите съставни компоненти на един уебсайт – HTML, CSS, Javascript и сървърен програмен език като PHP, Java, C# или друг. Обратнопропорционална по необходимо време и знания опция е използване на система за управление на съдържанието (CMS – content management system) като Wordpress, Joomla, Drupal или друга. С подобни системи се започва бързо, но са сравнително лимитирани по отношение на създаване на сложни или нестандартни

уебсайтове. За такъв тип проекти добър междинен вариант е употребата на програмна рамка (framework).

Програмните рамки представляват готов набор от библиотеки за създаване на определен тип софтуер. Те съдържат облекчават значително разработката на големи програмни проекти, като намаляват шансовете за грешки. Ако бъде избрана широко разпространена софтуерна рамка, може да се разчита на това, че програмният ѝ код е добре оптимизиран и тестван. Що се отнася до уеб приложения, софтуерните рамки улесняват писането, поддръжката и мащабирането на този тип приложения, като предоставят готови библиотеки и начини за реализиране на често извършваните в уеб програмирането дейности – удостоверяване на потребители, достъп до бази данни, управление на сесии, защита срещу злонамерени потребители.

Много от програмните рамки за разработка на сървърна част на уеб приложения използват традиционния архитектурен модел MVC (Model-View-Controller) или Модел-Изглед-Контролер, чиято цел е програмната логика да се раздели на три взаимосвързани елемента, всеки със строго определена роля. Моделът е централният елемент. Той описва данните, с които работи уеб приложението. Изгледът е компонентът, който визуализира данните от модела в графичен интерфейс. Контролерът отговаря за извличането и обработката на данните от модела и подаването им на изгледа. Примери за уеб рамки, които имплементират MVC парадигмата за програмиране, са Spring MVC, Laravel и Ruby on Rails.

## Web Frameworks



*Фигура 1.4 - Най-популярните програмни рамки за уеб приложения според анкета на Stack Overflow*

Друг навлизащ все повече в употреба вариант е на сървърната част да работи програмна рамка, която предоставя само базова функционалност (често такава програмна рамка се описва като „минималистична“), а клиентът да се грижи за по-голямата част от функционалността, отново чрез програмна рамка като React, Angular или Vue. Това е архитектурата от т.нар. тип „дебел клиент“ (thick client). Този вид приложения също доста често са съставени от една страница (SPA, single-page application), чийто потребителски интерфейс се променя динамично в резултат на заявки, направени от клиента към сървър, по REST програмен интерфейс, WebSocket или друг метод за комуникация. Тогава сървърът не отговаря на заявките с изцяло нова HTML страница, а с нужните на клиента данни в удобен формат (JSON, XML). Този начин на разработка носи ясно разделение между сървърна и клиентска част, което позволява двете части да се развиват независимо една от друга, и улеснява откриването на

проблеми и бъгове. Едно от негативните последствия е по-трудното оптимизиране на страниците за намиране от търсачките (SEO).

Независимо от това кой начин за реализиране на уеб приложение е избран, важен остава изборът на подходяща база данни. Традиционните бази данни са релационни. При тях данните се съхраняват във вид на таблици, съставени от записи и колони. Записите в таблиците могат да имат различни видове връзки помежду си – „едно към едно“, „едно към много“, „много към много“. Релационните бази данни се администрат чрез система за управление на бази данни (СУБД) и в повечето случаи се използва език за структурирани запитвания (SQL, Structured Query Language).

Друг вид бази данни са нерелационни или NoSQL (Not only Structured Language). Те съдържат информация от тип ключ-стойност. Основните им предимства са опростен дизайн и улеснено хоризонтално мащабиране. Примери за NoSQL бази данни са MongoDB, Google Cloud Firestore, Apache Cassandra и Oracle NoSQL Database.

### **1.3 Обзор на начини и технологии за реализиране на дълбоки невронни мрежи**

#### **1.3.1 Развойна среда и необходим хардуер**

Изкуствените невронни мрежи в най-базовия си вариант са описани за пръв път преди повече от 50 години, но навлизат в широка употреба сравнително наскоро. Една от историческите причини за това е липсата на достатъчно изчислителна мощност. С развитието на микропроцесорите и по-точно с развитието на графичните ускорители (също наричани видео карти или графични процесори), това вече не е такава пречка. Графичните ускорители де факто се превърнаха в изискване за трениране на невронни мрежи. Причината е, че ако необходимите изчисления се изпълняват на

видео картата, процесът на „трениране“ (или обучение) се ускорява значително в сравнение с изпълнение върху конвенционален централен процесор (това зависи от конкретния хардуер, но в някои случаи ускорението е над десетократно [5]). Това се дължи на наличието на много повече ядра при видео картите. Броят на ядрата е от порядъка на хиляди при видео картите от висок клас. Върху всички тези ядра се разпределя огромното количество прости изчисления (например матрично умножение), които са необходими, за да бъде обучена една невронна мрежа.

Въпреки бързото напредване на възможностите на хардуера, дори в днешно време хардуерните изисквания са високи. Това затруднява обучението на модели, които да постигат резултати на световно ниво. Поради тази причина съществуват както безплатни, така и платени облачни услуги, които позволяват навлизане в областта без нужда от голяма предварителна инвестиция. Още едно преимущество на облачните услуги е, че обикновено облачните инстанции, които се предоставят на потребителите, са предварително конфигурирани с необходимия софтуер и най-често използваните библиотеки за машинно самообучение. Следва кратък обзор на някои услуги от този тип.

**Colaboratory** (на кратко Colab) е безплатна платформа на Google, която позволява писане и изпълняване на Python код директно в браузъра. Кодът се изпълнява в т.нар. Jupyter тетрадки, които са подходящи за бързо итериране и експериментиране. Тези тетрадки работят на предоставени от Google виртуални машини, всяка от които по подразбиране е конфигурирана с видео карта от модела Tesla K80 на Nvidia. За съжаление, в Colab са заложили ограничения, които правят платформата негодна за големи проекти, свързани с машинно самообучение. Тези ограничения са скоростта на предоставените графични процесори и максималното време, за което може да се използва една тетрадка, преди да бъде прекъсната връзката към нея (12 часа). При прекъсване на връзката цялата виртуална

машина се рестартира и всички свалени на нея файлове биват изтрети. Това са причините платформата да не беше избрана за целите на дипломната работа.

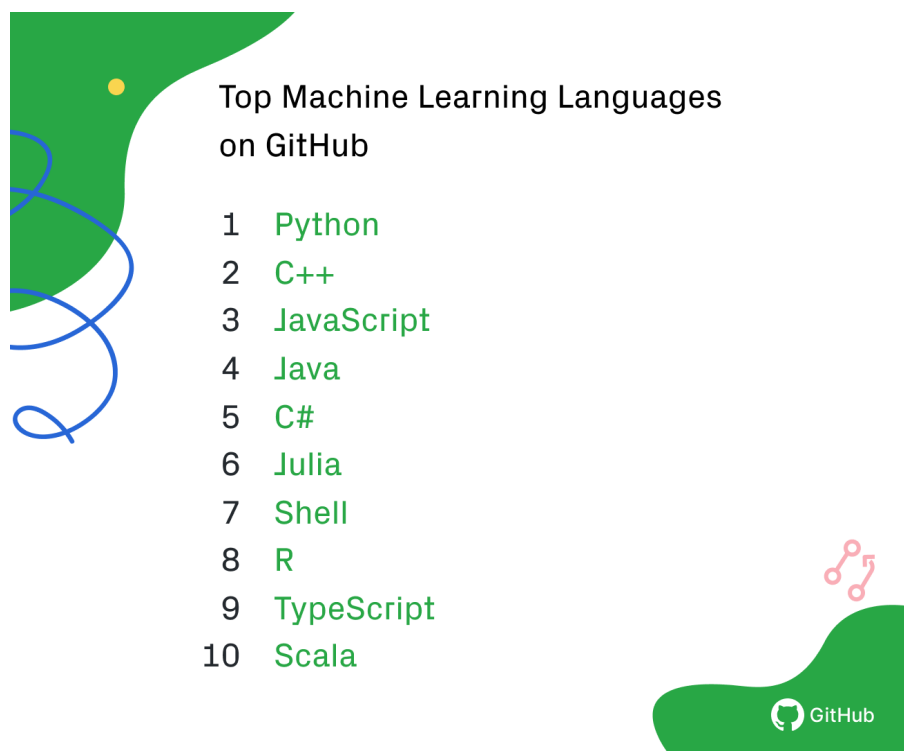
Друга подобна безплатна платформа е **Gradient** на Paperspace. Макар да си приличат по това, че и Colab, и Gradient предоставят възможността безплатно да се изпълнява Python код в Jupyter тетрадки върху сравнително бързи графични ускорителни, основната разлика е, че Gradient цели да бъде много по-всестранно развита платформа, която да облекчи разработването на модели за машинно самообучение от самото начало до предоставянето им на крайния потребител. Тук отново безплатното ползване е придружено от сериозни ограничения. Лимитът на времето, за което една инстанция може да работи без прекъсване, е 6 часа, максималният брой тетрадки за потребител е 5, а предоставените безплатно видео карти са Nvidia Quadro P5000. Съществуват два платени плана, които предоставят много по-добри условия за разработка, а също така Paperspace предлагат и специално пригодени планове за големи бизнеси.

Доста по-широко разпространен метод за обучение на невронни мрежи е чрез използване на платена облачна инфраструктура. Трите най-големи фирми, предоставящи облачни услуги – Amazon, Microsoft и Google, предоставят сравнително готови решения - Sagemaker на Amazon, Azure Machine Learning на Microsoft и AI Platform на Google. И трите предлагат много допълнителни възможности като например автоматично нагласяне на хиперпараметри, мониторинг на вече вкарани в употреба модели, специфични за целите на машинното самообучение DevOps инструменти и други.

### **1.3.2 Програмни езици и библиотеки за създаване и трениране на дълбоки невронни мрежи**

На теория проста невронна мрежа може да бъде създадена на всеки един модерен програмен език. Но за сериозна разработка на дълбоки невронни мрежи е необходим програмен език, който да е бърз, да има относително прост синтаксис (тъй като често по задачи за машинно самообучение работят специалисти, които не са програмисти по професия, а са експерти в областта, в която трябва да се приложи алгоритъмът), да може безпроблемно да работи с големи количества данни и да има достатъчно голяма общност от хора, които използват езика. Ето защо някои езици са се наложили пред други.

Python е езикът, за който вероятно се сеща почти всеки един програмист, когато чуе термина “машинно самообучение“. И това не е случайно. Според различни проучвания процентът на специалисти в областта на машинното самообучение варира между 50 и 80. При всички положения този процент е висок. Това не е случайно – някои от най-известните библиотеки, необходими за създаването на качествени модели, са написани на Python – Numpy, Pandas, Matplotlib, Scikit-learn, TensorFlow и много други.



Фигура 1.5 - Top програмни езици според броя Github хранилища с tag "machine-learning" [9]

Както и при създаването на уеб приложения, възможно е да започнем „от нулата“. Този подход обаче има сериозни недостатъци – дълго време на разработка и голяма вероятност за грешки. Ето защо и тук като основа обикновено се ползва библиотека като PyTorch, TensorFlow или scikit-learn. Тези библиотеки предоставят набор от готови функционалности, които са необходими за почти всеки проект за машинно самообучение като например методи за зареждане и обработка на големи количества данни, различни видове класификатори, методи за отчитане на това колко добре се справят тренирани модели, методи за визуализация на данните и други.



# Втора глава

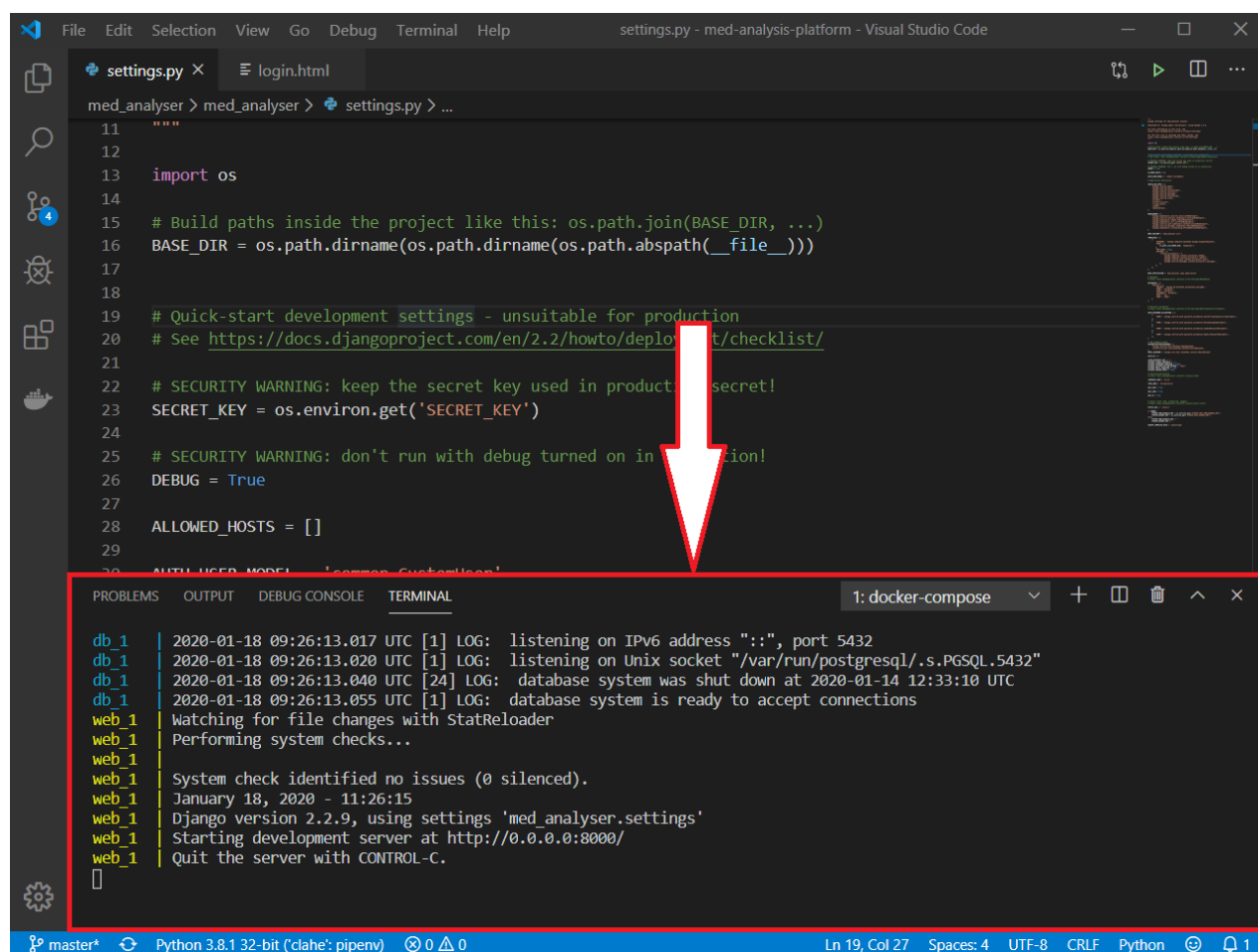
## 2.1 Функционални изисквания

- Система за регистрация и вход на потребители
- Специфични права за потребители, които са доктори, и такива, които са и болници
- Възможност за разглеждане, изтриване и редактиране на създадените от доктора изследвания
- Страница за всяко изследване, на която докторът, отговарящ за него, да може да прегледа цялата информация за изследването.
- Ако изображението към изследването е рентгенова снимка на гръден къш, докторът да може да прегледа поставената от невронната мрежа, както и „горещите точки“ от изображението, според които невронната мрежа се е ориентирала за предполагаемата диагноза. Също така да може той да постави диагноза. Ако двете диагнози се различават, това да бъде отбелязано в базата данни като дефект на алгоритъма
- Алгоритъмът автоматично да се тренира върху отбелязаните дефекти
- Възможност за закупуване на и за отказване от платен абонамент за болниците

## 2.2 Избор на среда за програмиране

### 2.2.1 Visual Studio Code

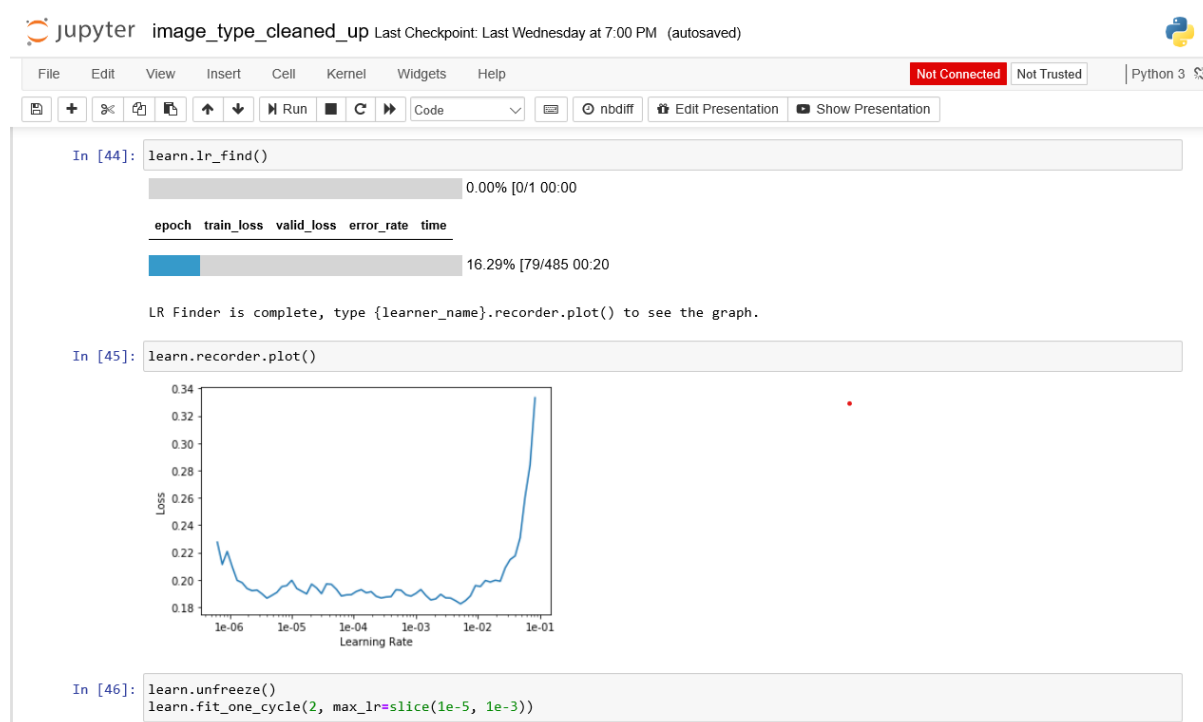
Visual Studio Code (VS Code) е сравнително нов текстов редактор с отворен код, разработван от Microsoft. Една от полезните възможности на VS Code е отварянето на вграден в потребителския интерфейс терминал и отварянето на множество табове в този терминал, което позволява по всяко време да се вижда изходният текст на Django сървъра (Фигура 2.1). Друго предимство е, че позволява инсталирането на приставки за добавяне на функционалност към редактора като лесно свързване към бази данни директно от редактора, автоматично откриване и поправяне на грешки в кода и много други.



Фигура 2.1 - Вграденият във VS Code терминал

## 2.2.2 Jupyter Notebook

Jupyter Notebook (накратко Jupyter или Jupyter тетрадка) е уеб приложение с отворен код, което позволява създаването и редактирането на документи, включващи множество отрязъци код, съпътстващ кода текст, визуализации и други. Напоследък популярността на Jupyter започна бързо да расте и се наложи почти като конвенция за разработване на невронни мрежи – процес, който е итеративен и експериментален. Именно такъв тип разработка – базирана на голям брой итерации и „пробване“ на различни подходи, позволява Jupyter чрез възможността множество парчета код да се изпълняват в произволен ред.



*Фигура 2.2 - Стандартен потребителски интерфейс на Jupyter тетрадка, в която са представени две визуализации*

## **2.3 Избор на технологии за разработка на уеб приложението**

### **2.3.1 Python**

Python е четвъртият най-използван програмен, скриптов или маркър език според анкетата на Stack Overflow за 2019 година. Причините за това са много, но основната е екосистемата от библиотеки и инструменти, изградена около езика, която го прави изключително гъвкав и пригоден за множество цели. Тази особеност на езика позволи той да бъде използван както за разработка на уеб частта, така и за разработка на невронните мрежи. Други негови предимства са лесен за научаване синтаксис, голяма общност и възможност както за обектно-ориентирано, така и за функционално програмиране. [4]

### **2.3.2 Miniconda**

Anaconda е специфична дистрибуция на Python, която е пригодена за научноизследователски цели (машинно самообучение, анализ на големи количества данни и др.) и улеснява инсталирането, управлението и настройката на широко използваните в тези сфери модули. Всеки проект е обособен в своя виртуална среда, която съдържа само необходимите за проекта модули. Това позволява същата среда лесно да се възпроизведе на друга машина. Anaconda използва conda като package manager. Miniconda е олекотена версия на Anaconda, която включва много по-малък брой модули от пълната версия. Това означава, че се инсталира по-бързо и заема по-малко пространство. Всички модули от Anaconda могат да се инсталират и на Miniconda.

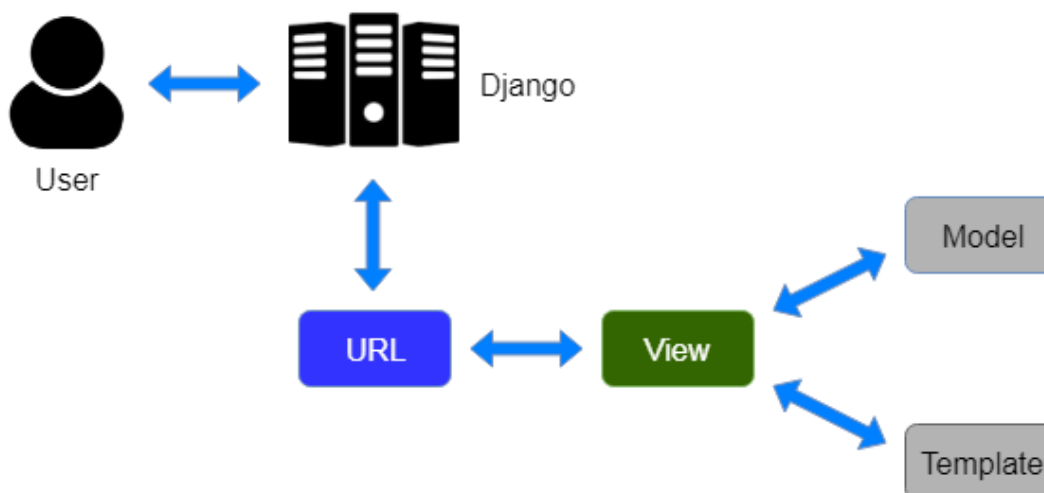
### **2.3.3 Django**

Django е най-широко използваната рамка (framework) за разработване на уеб приложения, написана на Python. Създателите ѝ я наричат рамка “с включени батерии”, което означава, че включва всички често използвани

при разработката на уеб приложения инструменти като например администраторски панел, модул за локализация, модул за оторизация и автентикация и др.

Въпреки че на практика Django използва стандартния MVC модел, създателите на Django го интерпретират по малко по-различен начин. Според тях изгледът (view) описва данните, които се представят на потребителя, т.е. не задължително как данните изглеждат, а кои данни ще бъдат визуализирани. На практика изгледът е функция за даден URL, която приема заявката от браузъра, и описва с кои данни да бъде отговорено. Добра софтуерна практика е съдържанието да се отдели от презентацията, така че всеки изглед се обръща към „шаблон“ (template), който описва как да се презентират данните. С други думи, може да се каже, че Django използва архитектурния модел Модел-Шаблон-Изглед (MTV, Model-Template-View, Фигура 2.3).

Основната причина Django да бъде избран за програмна рамка беше подробната, добре структурирана документация, голямата популярност сред Python общността, която предполага множество готови решения на често срещани проблеми и изобилие от допълнителни библиотеки, и сигурността на Django, която се изразява във вградена защита срещу най-често срещаните уязвимости, експлоатирани от злонамерени потребители.



Фигура 2.3 – MVT архитектурата на Django

### 2.3.4 PostgreSQL

По подразбиране Django използва SQLite като СУБД, но документацията препоръчва замяната ѝ при разработка на реален проект, така че за целите на настоящата дипломна работа е използвана системата за управление на бази данни PostgreSQL. Тя се интегрира лесно и безпроблемно с Django, поддържа почти пълния SQL стандарт и е проект с отворен код, който не се контролира от една централна организация, а е задвижван изцяло от общността, изградена около развиването на проекта.

### 2.3.5 Docker

Docker е набор от продукти, които използват виртуализация за предоставянето на софтуер в изолирани контейнери. Целта на внедряването на Docker в проекта е постигането на консистентна и лесна за възпроизвеждане среда на разработка, независима от операционната система на машината, на която се изпълнява проектът. Друга полза от Docker е, че позволява лесно управление на глобални променливи, обикновено идващи от средата, и на тайни API ключове. Освен това елиминира необходимостта от инсталиране на СУБД локално и поддържане на база данни.

### **2.3.6 Stripe**

Stripe е софтуерна платформа, която позволява лесно внедряване на разплащателна система в уеб и мобилни приложения. Предимствата на Stripe пред други системи, обработващи онлайн разплащания, са подробната документация, сравнително ниските такси (2.9% от размера на плащането + 30¢ на транзакция) и сигурността. Важна е и гъвкавостта, която Stripe предоставя – могат да бъдат внедрени различни разплащателни модели като еднократно плащане за услуга, плащане за множество продукти и плащане за абонамент през даден интервал от време. Още един бонус на Stripe е преносимостта на данните – ако в бъдеще бъде взето решение разплащателната система да се смени, всички до момента събрани данни от Stripe като кредитни карти, имейли на потребители, създадени абонаменти и тн. могат да бъдат запазени и прехвърлени в новата система.

### **2.3.7 Bootstrap 4**

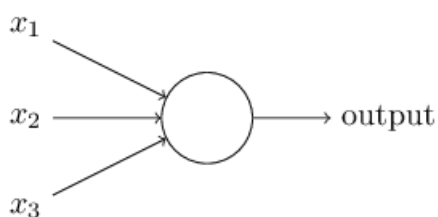
Bootstrap е безплатна CSS рамка за стилизиране на HTML елементи. Bootstrap позволява използване на множество готови елементи, стандартни за повечето уебсайтове днес. Също така са налични множество безплатни за използване шаблони, създадени от Bootstrap общността. Именно поради това беше избран Bootstrap за стилизирането на уеб приложението, а също и заради лесната интеграция с Django.

## **2.4 Избор на технологии за създаване на невронната мрежа**

### **2.4.1 Принцип на работа на невронна мрежа**

Както вече беше споменато, невронните мрежи са много стара идея, ако се съди по стандарта на скоростта на развитие на информационните технологии. Първата изкуствена невронна мрежа е създадена от Франк Розенблат през 1958 г. и е наречена перцептрон (или еднослоен

перцептрон). Целта на Розенблат е да моделира начина, по който човешкият мозък обработва информация.



Фигура 2.4 - Графично представяне на неврон

Еднослойният перцептрон представлява съвкупност от неврони. Всеки неврон е на практика сравнително проста математическа функция, която приема като аргументи един или повече двоични „сигнали“  $x_1, x_2, \dots$  и извежда един изходен двоичен „сигнал“ (Фигура 2.4). За

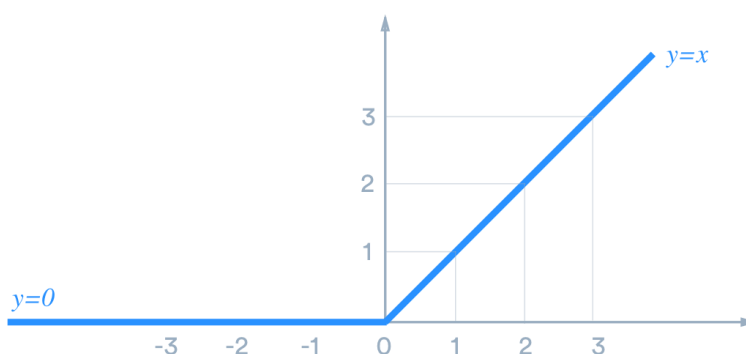
пресмятане на изходния сигнал се използват тегла  $w_1, w_2, \dots$  – числа, които изразяват колко е голямо влиянието на всеки входен сигнал спрямо стойността на изходния сигнал. Дали изходът на неврона ще е в 0 или 1 се определя от това дали сумата на произведенията на входните числа и теглата е по-голяма или по-малка от дадена стойност, наречена праг. Прагът също е параметър на неврона.

[6]

Принципът на работа на невроните в модерните невронни мрежи е много сходен с основната разлика,

че претеглената сума се подава на нелинейна

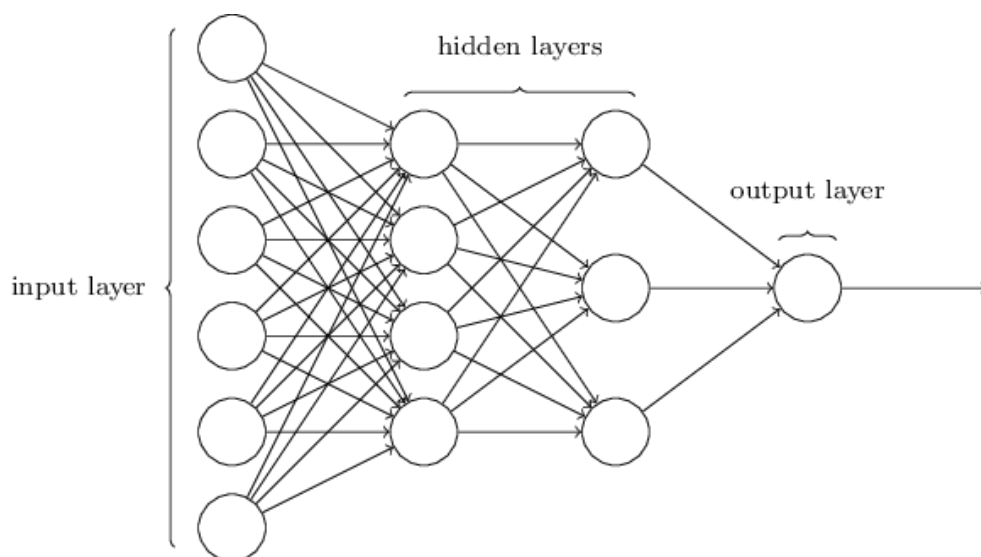
активационна функция (в миналото често се е използвала сигмоида, от известно време най-често използваната се нарича Rectified Linear Unit или ReLU, Фигура 2.5) и изходът от тази функция става изходът на неврона. Също така входовете и изходите не са ограничени само до стойностите 0 и 1. Това позволява много фино да променяме изхода, променяйки теглата. В противен случай съвсем малка промяна в някое тегло може да промени



Фигура 2.5 - Графика на ReLU функцията



драстична промяна в изхода, каквато е преминаването от 0 в 1 или обратно.  
[7]



*Фигура 2.6 - Графично представяне на многослоен перцептрон*

Многослойният перцептрон, вид изкуствена невронна мрежа, се състои от поне три слоя такива неврони, като първият слой се нарича входен, последният – изходен, а всички междинни слоеве – скрити (Фигура 2.6). “Напълно свързани“ слоеве са тези, при които изходът на всеки неврон от предния слой служи за вход на всеки един неврон от следващия слой. Най-важното и полезно свойство на многослойните перцептрони се описва от теоремата за универсална апроксимация (също наричана теорема на Цибенко), която гласи, че те могат с „достатъчно добро“ приближение да опишат която и да е математическа функция. За да се случи това обаче, е необходим процес на „обучение“, тоест на нагласяне на параметрите на невронната мрежа за постигане на желания резултат.

За обяснение на процеса на обучение често се използва класически пример в областта на машинното самообучение - класифициране на ръчно изписани цифри. На входния слой ще се подават всички пиксели от

изображението, а изходният слой ще се състои от 10 неврона, като всеки репрезентира вероятността на изображението да е съответната цифра (от 0 до 9). Броят на скритите слоеве и броят на невроните във всеки слой се определя от проектанта на невронната мрежа. В началото параметрите на мрежата се инициализират със случайни стойности. След като бъде подадено на невронната мрежа изображение например на числото 5, информацията се разпространява веднъж в права посока, достигайки до изхода, където невронът с най-голям изход е този, отнасящ се до числото 3. С други думи, предположението на невронната мрежа е неправилно, тъй като на този етап тя прави случайни предположения. За да бъде подобрен резултатът, се преминава към обратно разпространение на грешката (backpropagation). Чрез предварително избрана функция се пресмята „загубата“ (loss или също cost function), която изразява количествено колко неправилно е предположението на невронната мрежа. След това се пресмята градиентът на тази функция спрямо всяко тегло на всеки неврон от скритите слоеве и се променят теглата така, че да се минимизира загубата. Ако този процес бъде повторен няколко хиляди или няколко десетки хиляди пъти с различни изображения на цифрите от 0 до 9, накрая резултатът ще бъде невронна мрежа, която може с доста добра точност (най-вероятно над 90%) да разпознава ръчно написани цифри.

Модерните невронни мрежи са усложнен вариант на представения сравнително прост начин на работа. Усложнението се състои в добавяне на по-голям брой слоеве, добавяне на други видове слоеве, добавяне на техники и оптимизации за по-бързо трениране с по-добра генерализация (тъй като целта не е невронната мрежа да „назубри“ данните, върху които е тренирана и да може да разпознава само тях, а да научи принципните разлики във вида на различните цифри). Комбинацията от всички тези методи прави невронните мрежи най-ефективният инструмент не само за

анализиране на изображения, а също и за анализиране на видео, текст, звук, а дори и информация в табличен вид.

## **2.4.2 Google Cloud Platform (GCP)**

GCP е една от трите най-големи публични облачни платформи, която предоставя голямо разнообразие от услуги като например лесно мащабируем хостинг (Google App Engine), съхранение на големи количества данни (Google Cloud Storage) и достъп до виртуални машини с голяма изчислителна мощност (Google Compute Engine). Именно последната услуга позволи невронните мрежи да бъдат тренирани бързо. Нуждата от използване на облачна услуга бе породена от липсата на наличен локално хардуер. Главната причина за избора на GCP пред другите популярни облачни платформи е, че GCP предоставя напълно безплатно „кредит“ в размер на 300\$ при регистриране на нов акаунт. Други предимства на GCP са изгодните цени, възможността да се използват най-новите графични процесори и гъвкавостта при създаване на инстанции – всяка една характеристика на машината може да се избере спрямо нуждите на потребителя. Цената се променя динамично спрямо възможностите на избраната инстанция.

## **2.4.3 PyTorch**

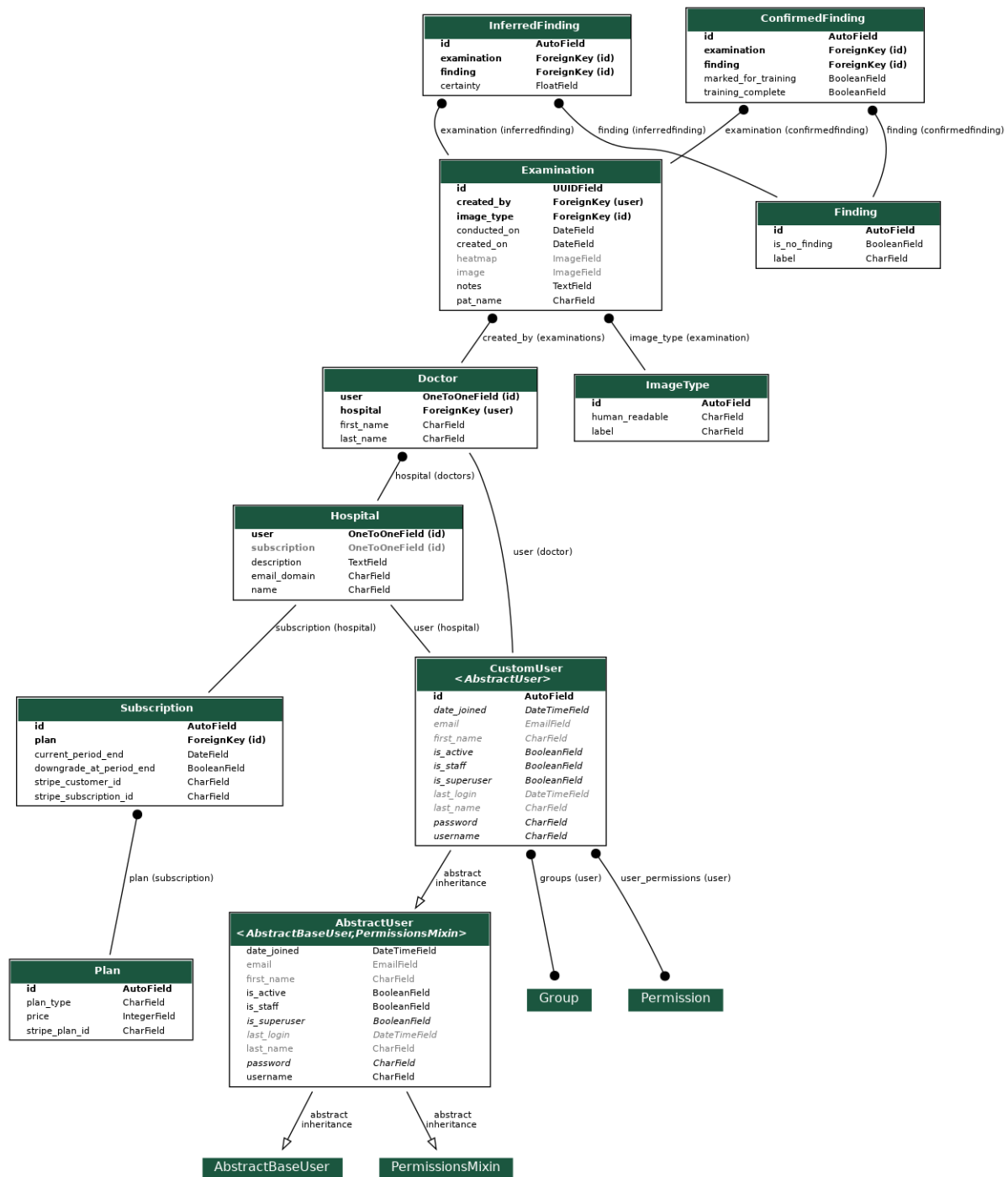
PyTorch е програмна рамка с отворен код, създадена от Facebook, и целяща да ускори процеса на реализиране на алгоритми за дълбоко машинно самообучение. Базирана на библиотеката Torch, PyTorch има за цел да предостави максимална гъвкавост и скорост на изпълнение. Позволява лесно прехвърляне на изчисления към един или повече графични ускорители. Постепенно PyTorch започва да се превръща в *de facto* програмна рамка за провеждане на сериозна изследователска дейност в областта на машинното самообучение. Едно доказателство за това е, че OpenAI, една от най-уважаваните изследователски организации,

занимаващи се с изкуствен интелект, в началото на 2020 г. обяви, че занапред ще използва PyTorch за всички проекти, освен ако няма специфична причина, поради която да употреби друга библиотека или рамка. [10]

#### **2.4.4 fastai**

fastai е библиотека с отворен код, създадена от изследователската институция fast.ai, която се фокусира върху улесняване на достъпа до най-новите техники за машинно самообучение и тяхното усъвършенстване. Библиотеката е изградена върху PyTorch и цели да направи процеса на разработка на дълбоки изкуствени невронни мрежи значително по-бърз и достъпен, като инкорпорира модерните най-добри практики. Факторите, които ме мотивираха да избира тази библиотека пред други подобни (например Keras), са отличната документация, наличието на готови решения за много проблеми и добрите резултати, които могат да се постигнат с относително малко количество код.

## 2.5 Структура на базата данни



Фигура 2.7 – UML диаграма на базата данни

- Таблица Examination – държи основна информация за всяко изследване

- id – колона от тип UUID, която уникално идентифицира всяко изследване
- created\_by – външен ключ към user, който е създал изследването
- image\_type – външен ключ към ред от таблицата ImageType, която пази всички възможни видове изображения. По тази колона се разбира дали изображението е на гръден кош, коляно, китка или друго.
- conducted\_on – дата на провеждане
- created\_on – дата на създаване
- image – път към изображението
- heatmap – път към изображение с визуализация на „горещите точки“ в медицинското изображение
- notes – записки на доктора
- pat\_name – име на пациента, за който се отнася изследването
- Таблица Finding – съдържа всички възможни аномалии във всеки вид изследване. Попълва се автоматично от миграционен файл.
  - id - автоматично генерирана от Django колона, която уникално идентифицира всяка аномалия
  - is\_no\_finding – указва дали дадената аномалия всъщност репрезентира липса на каквито и да е аномалии. Обикновено когато is\_no\_finding е True, label е ‘No Finding’.
  - label – название на аномалията
- Таблица InferredFinding – съдържа аномалиите, които са открити от невронните мрежи

- id - автоматично генерирана от Django колона, която уникално идентифицира всяка открита от невронна мрежа аномалия
- examination – външен ключ към изследването, в което е открита аномалията
- finding – външен ключ към аномалията
- certainty – колона от тип float, която съдържа активацията от последния слой на невронната мрежа за дадената аномалия. Колкото по-голямо е това число, толкова по-сигурна е мрежата, че аномалията присъства в изследването.
- Таблица ConfirmedFinding – съдържа потвърдени от докторите аномалии
  - id - автоматично генерирана от Django колона, която уникално идентифицира всяка потвърдена аномалия
  - examination – външен ключ към изследването, в което присъства аномалията
  - finding – външен ключ към аномалията
  - marked\_for\_training – дали предстои невронната мрежа да бъде тренирана върху тази аномалия
  - training\_complete – дали невронната мрежа вече е тренирана върху тази аномалия
- Таблица ImageType – съдържа всички възможни видове изображения (изследвания). Попълва се автоматично от миграционен файл.
  - id - автоматично генерирана от Django колона, която уникално идентифицира всеки вид изображение
  - human\_readable – название вида на изследването в по-четим вариант

- label – название на вида изследване
- Doctor – съдържа основна информация за всеки доктор. Уникално се идентифицира от връзката към потребителя.
  - user – връзка към потребител от тип „едно към едно“
  - hospital – външен ключ към болницата, от която е част докторът. Ако не е част от болница, има стойност NULL.
  - first\_name – първото име на доктора
  - last\_name – фамилията на доктора
- Hospital – съдържа основна информация за всяка болница. Уникално се идентифицира от връзката към потребителя.
  - user – връзка към потребител от тип „едно към едно“
  - subscription – външен ключ към абонамента на болницата
  - description – описание на болницата. Може да включва адреса, наличните отделения и друга важна информация.
  - email\_domain – имейл домейн адресът на болницата, по който се определя дали даден доктор е част от болницата. Ако имейл домейн адресът на болницата съвпада с този на доктора, докторът автоматично се добавя към болницата.
  - name – име на болницата
- Subscription – съдържа информация за абонамента на всяка болница и съответства на абонамент в Stripe
  - id – автоматично генерирана от Django колона, която уникално идентифицира всеки абонамент
  - plan – външен ключ към плана, за който е абонаментът



- `current_period_end` – датата, на която абонаментът преставва да бъде валиден
- `downgrade_at_period_end` – дали потребителят е поискал отказ от абонамента си. Ако е такъв случаят, в края на периода на валидност, планът към абонамента трябва да се смени с безплатен.
- `stripe_customer_id` – id на клиента, който притежава абонамента, в базата данни на Stripe
- `stripe_subscription_id` – id на абонамента в базата данни на Stripe
- **Plan** – съответства на създаден в Stripe план
  - `id` - автоматично генерирана от Django колона, която уникално идентифицира всеки план
  - `plan_type` – видът на плана – безплатен или платен
  - `price` – цена на плана
  - `stripe_plan_id` - id на плана в базата данни на Stripe

# Трета глава

## 3.1 Конфигуриране на Docker

За конфигуриране на Docker са използвани два файла – Dockerfile и docker-compose.yml. Dockerfile е текстов файл (Фигура 3.1), който съдържа командите за първоначалното създаване и конфигуриране на Docker image (Docker изображение).

```
FROM continuumio/miniconda3:latest

ENV PYTHONUNBUFFERED 1

WORKDIR /web

COPY ./conda_environment_linux.yml /web/conda_environment_linux.yml
COPY ./requirements.txt /web/requirements.txt

RUN conda create --name django-fastai
RUN conda env create -f /web/conda_environment_linux.yml --force

RUN apt-get update && apt-get install -y python-dev libpq-dev graphviz

RUN /bin/bash -c "source activate django-fastai && pip install -r /web/requirements.txt"

COPY . /web/
```

*Фигура 3.1 Съдържание на файла Dockerfile*

Първата команда указва кое е базовото изображение, върху което ще се надгражда. Ако базовото изображение не е налично локално, Docker автоматично проверява в хранилището (Docker Hub) и ако там изображението е налично, го сваля и създава контейнер от него. В официалното хранилище е достъпно голямо разнообразие от изображения, които могат да се пригодят към конкретната задача. Именно това в случая е направено – започнато е от последната версия на изображението miniconda3. Следващият ред задава променлива на средата, която

подсигурява терминалният изход от Docker да не е буфериран, тоест съобщенията за грешки ще бъдат извеждани мигновено за сметка на малки загуби в производителността. След това се създава работната директория, кръстена “web”, и се копират файловете, които съдържат изискванията съответно към conda средата и към пакети, които се инсталират през pip. Със следващите команди се създава нова conda среда, инсталират се в нея необходимите пакети, инсталират се Linux пакети чрез apt-get, които са необходими за правилното функциониране на базата данни, и се инсталират останалите пакети чрез pip. С последната команда се копират всички останали файлове от текущата директория, тоест самият проект. [11]

```
version: '3.7'
services:
  web:
    build: .
    command: bash -c "source activate django-fastai && python /web/manage.py runserver 0.0.0.0:8000"
    environment:
      - SECRET_KEY=0j*h)c9mw05oy26#8*c5w13aq@mmby0o0*@2+yju)$$_#pk2!)
      - DEBUG=True
      - STRIPE_TEST_PUBLISHABLE_KEY=pk_test_74YvyBqkakD4zxLzNlWwsobOX007KjEpmW0
      - STRIPE_TEST_SECRET_KEY=sk_test_NKc8tCg14LafsOIAhFI6GYrN00d9iWWIuZ
      - POSTGRES_HOST_AUTH_METHOD=trust
    volumes:
      - ./web
    ports:
      - 8000:8000
    depends_on:
      - db
  db:
    image: postgres:12
    volumes:
      - postgres_data:/var/lib/postgresql/data/
volumes:
  postgres_data:
```

Фигура 3.2 - Съдържание на файла *docker-compose.yml*

Compose е инструмент за създаване на Docker приложения, съставени от повече от един контейнер. В YAML файл се дефинират различните услуги (services) на приложението, които след това Docker изпълнява в изолирана среда. За всяка услуга могат да се задават променливи на средата.

На *Фигура 3.2* е показано създаването на услугите, необходими за функционирането на проекта – web и db. Дефиниран е и т.нар. “volume” – механизъм за съхранение на данни, създадени и използвани от Docker контейнери. Данните не се губят при спиране на контейнера. [12]

## 3.2 Създаване и трениране на дълбоките невронни мрежи

### 3.2.1 Невронна мрежа за разпознаване на типа снимка

Невронната мрежа за разпознаване на типа на снимката (основния обект в снимката) е тренирана върху изображения от 2 различни набора от данни (dataset) – MURA на Stanford и ChestX-ray на NIH. MURA съдържа голям брой рентгенови снимки на различни крайници – лакът, пръст, ръка, рамо, предмишница, раменна кост и китка, докато ChestX-ray съдържа рентгенови снимки, направени на гръден кош. [20, 21]

Първата стъпка е да се свалят всички данни, като тези от ChestX-ray се свалят чрез скрипт, предоставен за свободно ползване от създателите на самия dataset. За да останат данните сравнително балансирани, са използвани само 2000 изображения от ChestX-ray, които се прехвърлят в общата папка с данни. За сравнение, за всеки клас в MURA има между 727 и 3697 изображения или общо 14656 изображения (*Фигура 3.3*).

Study	Train		Validation		Total
	Normal	Abnormal	Normal	Abnormal	
Elbow	1094	660	92	66	1912
Finger	1280	655	92	83	2110
Hand	1497	521	101	66	2185
Humerus	321	271	68	67	727
Forearm	590	287	69	64	1010
Shoulder	1364	1457	99	95	3015
Wrist	2134	1326	140	97	3697
Total No. of Studies	8280	5177	661	538	14656

*Фигура 3.3 - Брой изображения в MURA за всеки тип снимка*

За по-лесно конструиране на цялостния набор от данни, изображенията за всеки тип снимка се преместват в папка с името на съответния тип. За целта от csv файл, който е част от MURA, се взима пътят, съдържащ наименованието на обекта в снимката, до всяко едно изображение. Спрямо пътя изображението се копира в съответната папка. Част от този процес е показан на Фигура 3.4.

```
label_dest = {
    'SHOULDER': 'data/shoulder',
    'HUMERUS': 'data/shoulder',
    'FINGER': 'data/finger',
    'ELBOW': 'data/elbow',
    'WRIST': 'data/wrist',
    'FOREARM': 'data/forearm',
    'HAND': 'data/hand',
}

for idx, name in enumerate(mura_paths):
    for label in label_dest:
        if label in name:
            dest = label_dest[label]

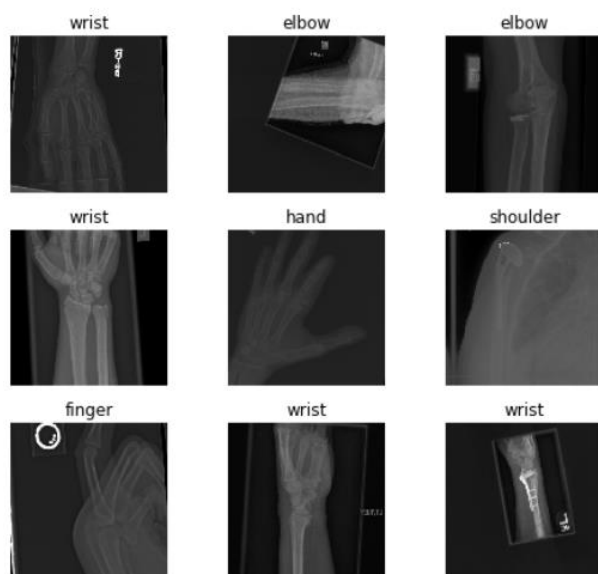
    destfname = dest + "image" + str(idx) + ".png"
    shutil.copy(name, destfname)
```

*Фигура 3.4 - Прехвърляне на всяко изображение от MURA в папка с името на обекта в снимката*

На Фигура 3.5 е показана следващата стъпка – зареждане на данните в обект от класа ImageDataBunch, който е част от библиотеката fastai. След това чрез този обект данните могат да се визуализират, валидират и да се предоставят на невронна мрежа за обучение. Създаването на ImageDataBunch обект се реализира чрез factory метода from\_folder, който взима желаната изходна стойност („ground truth”) за всяко изображение от името на папката, в която е изображението. Двадесет процента от данните се заделят като validation данни – като бъде тествана невронната мрежа върху тях, се отчита колко добре се справя тя. [19]

```
np.random.seed(2)
tfms = get_transforms()
data = ImageDataBunch.from_folder(path, ds_tfms=tfms, size=224, bs=64, valid_pct=0.2).normalize(imagenet_stats)
```

```
data.show_batch(rows=3, figsize=(7,6))
```



Фигура 3.5 - Зареждане на данните в *ImageDataBunch* обект и визуализиране на 9 изображения заедно с имената на обектите във всяко

```
learn = cnn_learner(data, models.resnet34, metrics=error_rate)
```

Фигура 3.6 - Създаване на невронна мрежа

След това се създава невронната мрежа. При обучението ѝ постигане на добри резултати върху данните е въпрос на експериментиране с хиперпараметрите и архитектурата. Изпробвани са архитектурите ResNet34 и ResNet50 поради тяхното добро представяне и широка употреба в практиката. Най-ниският процент на грешката (около 3%) се получава при трениране на невронна мрежа с архитектура ResNet50 (Фигура 3.7).

epoch	train_loss	valid_loss	error_rate	time
0	0.410887	0.188784	0.057982	04:27
1	0.250740	0.128860	0.038784	04:27
2	0.207341	0.108061	0.031697	04:25
3	0.169465	0.105474	0.030280	04:26

*Фигура 3.7 - Резултати след 4 епохи на обучение при архитектура ResNet50*

### **3.2.2 Невронни мрежи за разпознаване на аномалии в рентгенова снимка на гръден кош**

За осъществяване на задачата по откриване на аномалии са тренирани две невронни мрежи – първата открива дали има каквато и да е аномалия, като класифицира дадена снимка като ‘Finding’ (в снимката има поне една аномалия), или като ‘No Finding’ (в снимката не се забелязват аномалии). След това в зависимост от резултата от първата невронна мрежа, снимката може да премине през още една, която открива процентната вероятност за наличие на всяка аномалия.

За улеснение е използван csv файл (Data\_Entry\_2017.csv), предоставен от създателите на ChestX-ray, с данни за всяко изображение. Файлът се зарежда в dataframe – конструкция от библиотеката pandas, за удобно манипулиране на данните. Веднага след зареждането в dataframe биват премахнати всички колони, освен Image Index (име на изображението) и Finding Labels (изходни стойности за изображението), тъй като това са единствените необходими за тренирането на невронните мрежи колони. [13]

И за двете невронни мрежи обработката на данните е сходна. Първоначално всички изображения се разделят в две папки – train и test, а не на train и validation както при невронната мрежа за разпознаване на типа

снимка. Концептуалната разлика между test и validation данни е размита. Fastai дефинира test данни като данни, които не са обозначени с желана изходна стойност, докато в ChestX-ray test данните притежават такава. Тъй като fastai не позволява данните от test сета да имат предварително дефинирана изходна стойност, първоначално данните се разделят на train и test, както ги дефинира и ChestX-ray, а след това, при импортирането им в конструкция от fastai, се разделят на train и validation.

За целите на тренирането на първата невронна мрежа за всеки ред, в който колоната “Finding Labels” не съдържа “No Finding”, стойността на колоната “Finding Labels” се замества с “Finding” (Фигура 3.8). Също така е добавена допълнителна колона “is\_valid”, която указва дали изображението е част от данните за валидация, или е част от данните, върху които ще се тренира.

```
df_modified = pd.read_csv('Data_Entry_2017_modified.csv')
df_modified.head()
```

	Image Index	Finding Labels	is_valid
0	train/00000001_000.png	Cardiomegaly	False
1	train/00000001_001.png	Cardiomegaly Emphysema	False
2	train/00000001_002.png	Cardiomegaly Effusion	False
3	train/00000002_000.png	No Finding	False
4	test/00000003_000.png	Hernia	True

```
df_modified['Finding Labels'].replace('^(!No\sFinding).*$', 'Finding', regex=True, inplace=True)
df_modified.head()
```

	Image Index	Finding Labels	is_valid
0	train/00000001_000.png	Finding	False
1	train/00000001_001.png	Finding	False
2	train/00000001_002.png	Finding	False
3	train/00000002_000.png	No Finding	False
4	test/00000003_000.png	Finding	True

*Фигура 3.8 - Обработка на данните за трениране на първата невронна мрежа*



Този път за създаване на ImageDataBunch отговаря data block интерфейса на fastai, който осигурява по-голяма гъвкавост (Фигура 3.9). Първоначално се създава ImageList, от който се конструира ImageDataBunch обект. ImageList обектът се композира от dataframe обект, подаден на функцията from\_df. Вторият аргумент, path, указва базовата директория, в която fastai да търси изображенията. Чрез извикване на метода split\_from\_df данните автоматично се разделят на train и validation според стойността на колоната is\_valid в dataframe обекта. Методът label\_from\_df указва, че изходните стойности за всяко изображение трябва да бъдат взети от втората колона на dataframe обекта.

```
bs=64
np.random.seed(42)
src = (ImageList.from_df(df=df_any_findings, path=path)
      .split_from_df()
      .label_from_df())

tfms = get_transforms(flip_vert=False, max_warp=0.)

data = (src.transform(tfms, size=224)
      .databunch(bs=bs).normalize(imagenet_stats))
```

*Фигура 3.9 – Създаване на ImageDataBunch чрез data block API*

Отново се създава невронна мрежа. След четири епохи на трениране на ResNet50 е постигната стойност за AUC (area under the curve или зона под кривата, [18]) от 0.72. Това показва, че невронната мрежа не „гадае“ (в такъв случай резултатът би бил 0.5), а наистина разпознава релевантни особености в изображенията.

epoch	train_loss	valid_loss	error_rate	auroc	time
0	0.612127	0.604663	0.308837	0.697361	19:21
1	0.582859	0.607965	0.304930	0.705357	14:55
2	0.567903	0.591967	0.291022	0.720606	14:57
3	0.555679	0.589062	0.291139	0.720630	14:57

*Фигура 3.10 - Резултати за невронната мрежа, която разпознава дали в изображение има поне една аномалия*

При подготовката за трениране на втората невронна мрежа, тъй като са необходими само изображенията с поне една аномалия, се премахват всички редове от dataframe обекта, които имат стойност 'No Finding'. Създаването на ImageDataBunch обекта е много сходно с това при първата невронна мрежа (Фигура 3.11). Всъщност единствените разлики са, че на метода `from_df` се подава друг dataframe и че на метода `label_from_df` се подава аргумент `label_delim` със стойност '|'. Този аргумент указва, че е възможно за дадено изображение да е валидна повече от една изходна стойност (тъй като в едно изображение може да има една или повече аномалии), и също така указва с какъв символ са разделени изходните стойности във втората колона на dataframe обекта.

```

bs=64
np.random.seed(42)
src = (ImageList.from_df(df=df_findings, path=path)
      .split_from_df()
      .label_from_df(label_delim='|'))

tfms = get_transforms(flip_vert=False, max_warp=0.)

data = (src.transform(tfms, size=224)
      .databunch(bs=bs).normalize(imagenet_stats))
data.classes

['Atelectasis',
 'Cardiomegaly',
 'Consolidation',
 'Edema',
 'Effusion',
 'Emphysema',
 'Fibrosis',
 'Hernia',
 'Infiltration',
 'Mass',
 'Nodule',
 'Pleural_Thickening',
 'Pneumonia',
 'Pneumothorax']

```

*Фигура 3.11 - Създаване на ImageDataBunch за невронната мрежа за разпознаване на аномалии в снимките*

Процесът на създаване и трениране на невронната мрежа също е много сходен, като са използвани повече метрики за оценка на невронната мрежа (Фигура 3.12). По-подробно описание на всяка е налично в документацията на fastai [19]. След шест епохи на трениране невронната мрежа спира да се подобрява, което говори за достигане или поне доближаване до overfit – състояние, в което невронната мрежа се научава да разпознава само конкретните изображения в train данните и не генерализира добре. Затова тренирането се прекратява след шестата епоха (Фигура 3.13).

```
acc_02 = partial(accuracy_thresh, thresh=0.2)
acc_03 = partial(accuracy_thresh, thresh=0.3)
f_1_score = partial(fbeta, thresh=0.5, beta=1)
f_2_score = partial(fbeta, thresh=0.5, beta=2)

learn = cnn_learner(data, models.resnet50, metrics=[acc_02, acc_03, f_1_score, f_2_score])
```

*Фигура 3.12 - Създаване на невронната мрежа, която разпознава аномалии*

```
learn.fit_one_cycle(4, slice(1e-5, lr/10))
```

epoch	train_loss	valid_loss	accuracy_thresh	accuracy_thresh	fbeta	fbeta	time
0	0.240778	0.279846	0.839575	0.873685	0.300444	0.278917	06:21
1	0.238453	0.280524	0.844237	0.874043	0.296325	0.274481	06:25
2	0.236730	0.279646	0.840860	0.873208	0.307910	0.286143	06:26
3	0.235366	0.279189	0.838831	0.872400	0.314716	0.293011	06:25

```
learn.fit_one_cycle(2, slice(1e-5, lr/10))
```

epoch	train_loss	valid_loss	accuracy_thresh	accuracy_thresh	fbeta	fbeta	time
0	0.237020	0.279400	0.839139	0.872840	0.318262	0.296217	06:23
1	0.238706	0.279230	0.839207	0.872650	0.314547	0.292841	06:23

*Фигура 3.13 - Резултати след шест епохи на трениране на невронната мрежа за разпознаване на аномалии*

### 3.3 Описание на функционалността на приложенията в Django

Структурата на един Django проект се разделя на приложения (apps), всяко от което отговаря за специфична част от функционалността на сайта. Този начин на организация позволява едно приложение да се използва в много проекти. В повечето случаи едно приложение включва комбинация от модели, изгледи, шаблони, статични файлове и URL адреси. [14]

За да разпознае Django дадено приложение, то трябва да бъде включено в списъка `INSTALLED_APPS` в `settings.py` - файла с настройки на Django (Фигура 3.14). Там е необходимо да се добавят и всички инсталирани от трети страни приложения, които ще бъдат използвани в проекта.

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'django.contrib.sites',  
    'django_extensions',  
    'allauth',  
    'allauth.account',  
    'crispy_forms',  
    'braces',  
    'common.apps.CommonConfig',  
    'examinations.apps.ExaminationsConfig',  
    'subscriptions.apps.SubscriptionsConfig',  
]
```

*Фигура 3.14 – Добавени в проекта приложения*

### 3.3.1 Обяснение на някои концепции в Django

За целите на проекта са използвани единствено изгледи, базирани на класове (CBV, class-based views). Те позволяват чрез наследяване от вградени в Django класове бързо да се създават стандартни изгледи например за извличане на всички данни за даден запис от базата данни или за извличане на данни за група от записи. Освен това са лесно разширяеми чрез override на методи на класове, от които наследяват.

**Сигналите** (signals) в Django са съобщения, които се изпращат от подател на набор от получатели при настъпване на дадено събитие. Полезни са, когато много части на кода зависят от едно и също събитие. Най-често използваните сигнали са pre\_save, post\_save, pre\_delete и post\_delete. Те се изпращат от даден модел, когато обект от този модел съответно предстои да бъде запазен, вече е запазен, предстои да бъде изтрит или е изтрит. Възможно е създаване и на персонализирани сигнали, които да бъдат

изпращани само в специални случаи. На Фигура 3.15 е показан пример за това как с декоратора `receiver` се регистрира метод-получател, който да се изпълни след изтриване на обект от класа `Examination`.

```
@receiver(post_delete, sender=Examination)
def submission_delete(sender, instance, **kwargs):
    instance.image.delete(False)
```

*Фигура 3.15 - Регистриране на метод-получател*

### 3.3.2 Приложението “common”

Както подсказва името, приложението „common” помещава функционалност, която концептуално е обща и важна за функционирането на останалите приложения. В случая тази функционалност е създаването и управлението на потребителските профили и групи.

В проекта всеки потребител се приема за доктор. Опционално, след създаване на акаунта, може да бъде повишен и до представляващ болница. В частта на сайта, която е предвидена за потребители, могат да се създават само докторски акаунти. Причината за това е, че се очаква, че когато проектът започне да се използва в реална среда, ще има предварителна комуникация между всяка болница, която смята да започне да използва платформата, и администратор на сайта.

За да започне след нормална регистрация акаунтът да представлява и болница, трябва болницата да се създаде в администраторския панел от администратор на уебсайта. След това даденият акаунт на практика започва да представлява и доктор, и болница, тоест може да има свои изследвания, а също така може и да закупи абонамент, от който да се възползват той и всички доктори, които са част от болницата.

Дали доктор е част от дадена болница се определя по името на домейна на имейла. След създаване на акаунт, ако имейл домейнът отговаря на този на дадена болница, докторът автоматично се добавя към нея и може да се възползва от нейния професионален абонаментен план, ако такъв е закупен (Фигура 3.16, Фигура 3.17).

```
@receiver(pre_save, sender=Doctor)
def pre_save_doctor_add_to_hospital(sender, instance, **kwargs):
    user_email_domain = instance.user.email.partition("@")[2]
    hospital = Hospital.objects.filter(email_domain=user_email_domain)
    if hospital:
        instance.hospital = hospital.first()
```

*Фигура 3.16 - Добавяне на доктор към болница*

```
@receiver(post_save, sender=Doctor)
def post_save_doctor_create_and_add_groups(sender, instance, **kwargs):
    generate_doctor_groups_and_permissions()
    instance.user.groups.add(Group.objects.get(name='free_doctors_group'))
    if instance.hospital.user.groups.filter(name='pro_hospitals_group').exists():
        instance.user.groups.add(Group.objects.get(name='pro_doctors_group'))
```

*Фигура 3.17 - Добавяне на доктор към съответните групи*

В проекта съществуват четири потребителски групи – две за доктори и две за болници, като в бъдеще със сравнително малко усилия биха могли да се добавят и още. Всяка потребителска група носи права, съответстващи на безплатен или платен абонамент за съответния вид потребители (доктори или болници). Групите и правата за съответния вид акаунт се създават при създаването на първия акаунт от този вид (Фигура 3.18).

```

def generate_doctor_groups_and_permissions():
    ct = ContentType.objects.get_for_model(Doctor)

    pro_doctors_group, created = Group.objects.get_or_create(name='pro_doctors_group')
    free_doctors_group, created = Group.objects.get_or_create(name='free_doctors_group')

    is_doctor_perm, created = Permission.objects.get_or_create(codename='is_doctor',
name='The user is a doctor and so can access all doctor-specific functionality',
content_type=ct)
    max_examinations_perm, created = Permission.objects.get_or_create(
        codename='can_exceed_max_examinations',
        name='Can create more than the standard maximum for examinations',
        content_type=ct
    )

    free_doctors_group.permissions.add(is_doctor_perm)
    pro_doctors_group.permissions.add(is_doctor_perm)
    pro_doctors_group.permissions.add(max_examinations_perm)

def generate_hospital_groups_and_permissions():
    ct = ContentType.objects.get_for_model(Hospital)

    pro_hospitals_group, created = Group.objects.get_or_create(name='pro_hospitals_group')
    free_hospitals_group, created = Group.objects.get_or_create(name='free_hospitals_group')

    is_hospital_perm, created = Permission.objects.get_or_create(
        codename='is_hospital',
        name='The user is a hospital and so can access all hospital-specific functionality',
        content_type=ct
    )

    free_hospitals_group.permissions.add(is_hospital_perm)
    pro_hospitals_group.permissions.add(is_hospital_perm)

```

### *Фигура 3.18 - Създаване и конфигуриране на групите и правата*

При създаване на болница за улеснение се създават и необходимите данни за болницата в Stripe – клиент и безплатен абонамент за клиента. Създава се и нов безплатен абонамент, прикачен към болницата, в базата данни на проекта (Фигура 3.19). Така когато болницата реши да закупи платен абонамент, тази информация вече ще е налична и клиентът в базата данни на Stripe може да бъде прехвърлен към платен план, вместо да се създава изцяло нов абонамент.



```

@receiver(pre_save, sender=Hospital)
def pre_save_hospital_create_subscription(sender, instance, **kwargs):
    free_plan = Plan.objects.filter(plan_type='free').first()
    customer = stripe.Customer.create(email=instance.user.email,
    |     name=instance.user.doctor.first_name + " " + instance.user.doctor.last_name)
    stripe_sub = stripe.Subscription.create(customer=customer.id, items=[{
    |     "plan": free_plan.stripe_plan_id
    | }])
    sub = Subscription.objects.create(stripe_subscription_id=stripe_sub.id,
    |     stripe_customer_id=customer.id,
    |     plan=free_plan)
    instance.subscription = sub

```

*Фигура 3.19 – Създаване на безплатен абонамент за новосъздадена болница*

### 3.3.3 Приложението “examinations”

В това приложение се съхраняват обучените невронни мрежи, както и логиката за създаване, управление и обработка на изследвания.

Django по подразбиране за всяко приложение в проекта създава apps.py файл с клас, който наследява от AppConfig. В този клас могат да се зададат метаданни за приложението. Именно тук се зареждат невронните мрежи, тъй като това е сравнително вреемка операция. Поставянето на тази операция в класа ExaminationsConfig позволява да се изпълни само веднъж при инициализиране на приложението, а не всеки път при необходимост от използване на невронните мрежи (Фигура 3.20).

```

class ExaminationsConfig(AppConfig):
    name = 'examinations'
    path = Path(os.path.join(settings.ML_MODELS))
    path_image_type = path/'image_type_model'
    path_chest_xray = path/'chest_xray_model'

    learner_image_type = load_learner(path_image_type)
    learners_findings = {
        "chest": {
            'any_findings': load_learner(path_chest_xray, 'any_findings.pkl'),
            'findings': load_learner(path_chest_xray, 'findings.pkl')
        }
    }

    def ready(self):
        from .ml_models import trainer
        confirmed_findings_model = self.get_model('ConfirmedFinding')
        image_type_model = self.get_model('ImageType')
        examinations_model = self.get_model('Examination')
        trainer.start(self.learners_findings,
                      confirmed_findings_model, image_type_model, examinations_model)

```

*Фигура 3.20 - Класът ExaminationsConfig*

Освен това във файла `apps.py`, в метода `ready` на класа `ExaminationsConfig`, се задава интервала, през който да се тренират невронните мрежи на базата на обратната връзка, събрана от потребителите. Мястото на това парче код е именно в този метод, тъй като в него могат да се импортират модели от приложението. Това не е възможно на друго място в класа `ExaminationsConfig`, защото класът се зарежда от Django преди зареждането на моделите. [14]

Самото трениране е имплементирано във файла `trainer.py`. То се свежда до това всички записи от модела `ConfirmedFinding`, върху които съответната невронна мрежа не е тренирана, да бъдат записани в `DataFrame`. След това чрез `data block` интерфейса на `fastai` се конструира `dataset`, състоящ се само от новите данни, върху който се тренира невронната мрежа и се запазва като изцяло нова. Тя не се замества автоматично на мястото на

старата понеже би било добре след всеки цикъл на трениране да се валидира дали наистина има подобрене в резултатите.

```
class ExaminationCreateView(LoginRequiredMixin, UserPassesTestMixin, CreateView):
    def exceeded_max_examinations_redirect(self, request):
        messages.add_message(request, messages.INFO,
            '''You've exceeded the maximum number of examinations!
            Please delete an existing examination or get a higher tier subscription.'''
        )
        return HttpResponseRedirect(reverse_lazy('examination_list'))

    raise_exception = exceeded_max_examinations_redirect
    redirect_unauthenticated_users = True
    form_class = ExaminationUploadForm
    template_name = 'examinations/examination_new.html'

    def test_func(self, user):
        return user.doctor.can_create_more_examinations

    def form_valid(self, form):
        form.instance.created_by = self.request.user.doctor
        form.instance.created_on = datetime.date.today()

        img = open_image(form.instance.image.file)
        pred_class, pred_idx, outputs = ExaminationsConfig.learner_image_type.predict(img)
        form.instance.image_type = ImageType.objects.get(label=str(pred_class))

        return super().form_valid(form)
```

*Фигура 3.21 - Класът-изглед, който създава едно изследване*

При създаване на изследване се проверява максималният брой изследвания, които докторът може да създава (25 при безплатен абонамент, неограничен при платен), и ако този брой няма да бъде надвишен със създаване на изследването, то се създава и се открива видът на изследването от невронната мрежа, отговаряща за това. След това функция, която е регистрирана да се изпълнява след запазване на изследването, чрез друга невронна мрежа намира вероятностите за всеки вид аномалия, която може да бъде открита в дадения вид изследване, и ги записва в базата данни (Фигура 3.22). Тази функция също извиква функцията `generate_heatmap`,

която създава визуализация на „Горещите точки“ (heat map) в изображението.

```
@receiver(post_save, sender=Examination)
def infer_findings(sender, instance, created, **kwargs):
    if created:
        # delete all existing inferred findings
        inferred_findings = InferredFinding.objects.filter(examination=instance)
        inferred_findings.delete()

        # infer new findings
        img = open_image(instance.image.file)
        try:
            any_findings_learner = ExaminationsConfig.learners_findings[instance.image_type.label]['any_findings']
            pred_class, pred_idx, outputs = any_findings_learner.predict(img)
            inferred_finding = InferredFinding(examination=instance,
                                                finding=finding.objects.get(is_no_finding=True),
                                                certainty=outputs[1])
            inferred_finding.save()

            if inferred_finding.certainty < 0.5:
                generate_heatmap(instance, img, any_findings_learner, pred_idx)

            findings_learner = ExaminationsConfig.learners_findings[instance.image_type.label]['findings']
            pred_class, pred_idx, outputs = findings_learner.predict(img)
            for c, output in zip(findings_learner.data.classes, outputs):
                inferred_finding = InferredFinding(examination=instance,
                                                    finding=finding.objects.get(label=c),
                                                    certainty=output)
                inferred_finding.save()

        except KeyError:
            print('No model for this image type')
```

Фигура 3.22 – Намиране на аномалии в изследване

```
def generate_heatmap(examination_instance, img, learner, pred_idx):
    m = learner.model.eval()
    xb, _ = learner.data.one_item(img)
    xb_denormed = FAIImage(learner.data.denorm(xb)[0])

    def hooked_backward(pred_idx, x_batch):
        with hook_output(m[0]) as hook_a:
            with hook_output(m[0], grad=True) as hook_g:
                preds = m(x_batch)
                preds[0, int(pred_idx)].backward()
        return hook_a, hook_g

    hook_a, hook_g = hooked_backward(pred_idx, xb)
    heatmap = hook_a.stored[0].mean(0)

    # plot the heatmap on top of image
    _, ax = plt.subplots()
    xb_denormed.show(ax)
    ax.imshow(heatmap, alpha=0.6, extent=(0, xb.shape[2], xb.shape[3], 0), interpolation='bilinear', cmap='magma')

    buf = io.BytesIO()
    plt.savefig(buf, format='jpeg')
    examination_instance.heatmap.save(str(examination_instance.id) + "_heatmap.jpg", File(buf), save=True)
```

Фигура 3.23 - Създаване и запазване на визуализация на "горещите точки" в изображението

В функцията `generate_heatmap` (*Фигура 3.23*) още веднъж снимката преминава през невронната мрежа. За запазване на стойностите от последния конволюционен слой на невронната мрежа са използвани куки (`hooks`, механизъм в `PyTorch` и `fastai`, който позволява изпълнение на една или повече функции в който и да е момент от правото разпространение на данните или обратното разпространение на грешката в невронната мрежа). Тези стойности са решаващи за изхода на невронната мрежа и се съхраняват под формата на тензор от трети ранг с големина на първото измерение 512. Големината на останалите две измерения зависи от размера на изображението. Ако тя е 11 (т.е. тензорът е с форма `[512, 11, 11]`) и ако разгледаме този тензор като 512 матрици с размерност `11 x 11`, то числата във всяка матрица описват наличието или отсъствието на дадена особеност (*feature*) и позицията ѝ. Например, ако в изображението има сравнително голяма правоъгълна бяла област (така област често говори за натрупване на течност в белите дробове), и невронната мрежа е добре тренирана да разпознава такъв тип аномалия, то най-вероятно някоя от тези матрици отговаря за разпознаването ѝ и е силно активирана в областта, в която е аномалията. Ето защо, осреднявайки стойностите от всички тези матрици, се получава матрица с размерност `11 x 11`, от която става ясно в кои области са били засечени най-много особености на изображението. За визуализиране на матрицата върху изображението се извикват две функции – `show`, която визуализира чрез библиотеката `matplotlib` изображението, и `imshow`, също функция от `matplotlib`, която разтяга и налага матрицата от осреднени активации върху изображението. На *Фигура 3.24* е илюстрирана получената фигура. След това тя се запазва в поток от тип `BytesIO`, за да може после лесно да се запише в полето `heatmap` на изследването.

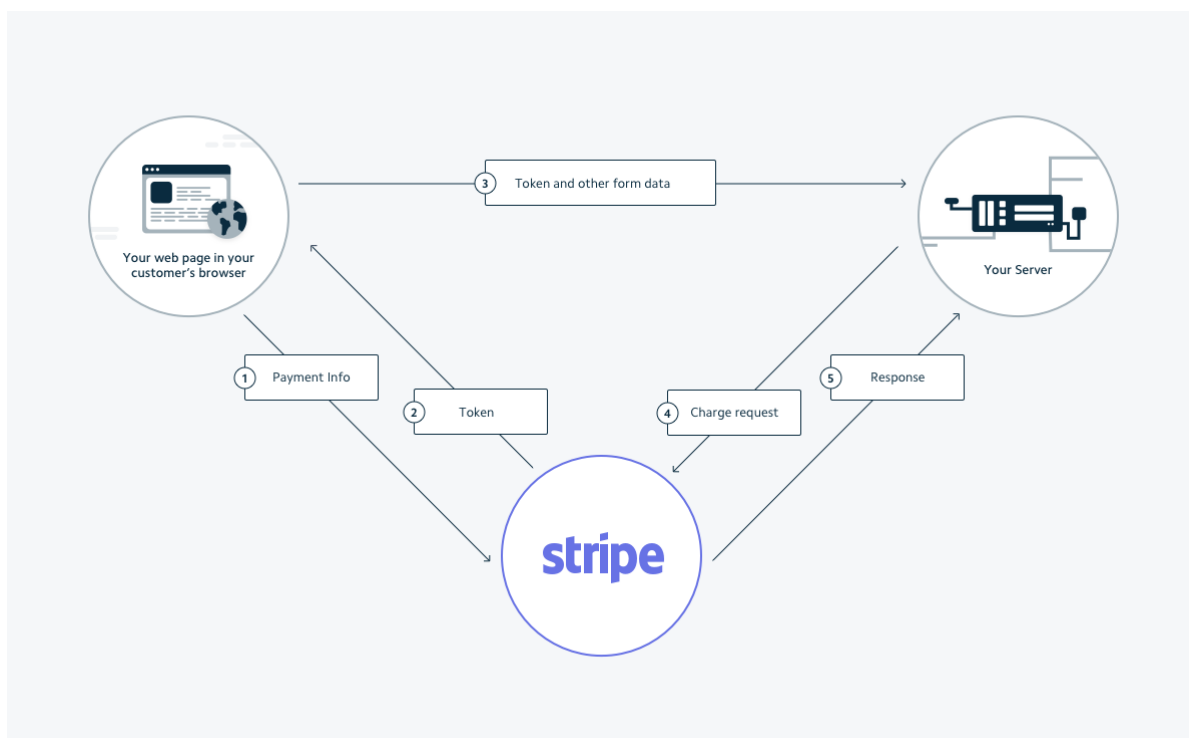


*Фигура 3.24 - Визуализация на "горещи точки" в изображението*

### **3.3.4 Приложението “subscriptions”**

В приложението “subscriptions” е имплементирана функционалността, която позволява на болниците да ползват приложението безплатно, или да закупят платен абонамент.

За реализиране на плащанията е използван програмният интерфейс на Stripe. На Фигура 3.25 е показана последователността за обработване на плащания през Stripe в общия случай. В случая плащанията се правят за закупуване на абонамент, който периодично се обновява, поради което има имплементационните разлики. Stripe връща на брауъра не token, а id на разплащателния метод (при стъпка 2). Уеб сървърът не прави директно заявка за плащане към Stripe, а Stripe прави заявката за плащане след като веб сървърът направи заявка за преминаване от безплатен към платен план.



*Фигура 3.25 – Последователност на стъпките при извършване на плащане през Stripe*

Когато потребителят избере плана, към който желае да се абонира, планът се записва в сесията и бива изтрит, когато процесът по плащането приключи успешно. Идеята е в бъдеще да могат лесно да се добавят още планове. Ако потребителят реши да прекъсне процеса по абониране, променливата за плана остава в сесията до следващия опит за абониране, при който тя се презаписва в случай, че потребителят е решил да се абонира към друг план.

При следващата стъпка се изисква от потребителя да въведе данни за своята дебитна или кредитна карта. Формата се конструира в браузъра изцяло от Stripe.js и Stripe Elements – библиотеките на Stripe, които позволяват никакви чувствителни данни да не се съхраняват в базата данни на проекта. Вместо това Stripe събира и пази тези данни. Единствената информация, която се получава на сървърната част след изпращане на формата, е id на разплащателния метод (PaymentMethod), въведен от

потребителя. В изгледа, който обработва POST заявката от формата, новосъздаденият `PaymentMethod` обект се прикача към клиента (`Customer` обекта) и планът в Stripe абонамента му се заменя с желанието от него. В този момент поради преминаването от безплатен към платен план Stripe прави опит да изтегли необходимата сума от разплащателния метод. В зависимост от новия статус на абонамента се разбира дали плащането е успешно, неуспешно или изисква допълнително потвърждение (Фигура 3.27).

```
let submitButton = document.getElementById('submitBtn');
const paymentIntentSecret = submitButton.dataset.secret;
stripe.confirmCardPayment(paymentIntentSecret).then(function(result) {
  if (result.error) {
    let modal = $('#messagesModal');
    modal.find('.messages').empty();
    modal.find('.messages').append('<li>Authentication has failed</li>');
    modal.modal('show');
  } else {
    // The payment has succeeded
    document.forms['authentication-form'].submit();
  }
});
```

*Фигура 3.26 - Потвърждаване на плащане в клиентската част*

В случай, че се изисква потвърждение, отново `Stripe.js` поема по-голямата част от работата. Единственото изискване към сървъра е да подаде на Stripe тайния ключ на клиента. След това `Stripe.js` визуализира модална форма, през която клиентът да потвърди плащането (Фигура 3.26). Детайлите по метода на потвърждаване зависят от банката, от която е издадена картата.



```

def post(self, request, *args, **kwargs):
    desired_plan = get_desired_plan(self.request)
    if desired_plan is None: return redirect(reverse('subscriptions:manage'))

    payment_method_id = request.POST['payment_method_id']
    customer_id = get_customer_id(self.request.user)
    stripe.PaymentMethod.attach(payment_method_id, customer=customer_id)
    customer = stripe.Customer.modify(customer_id,
        invoice_settings={
            'default_payment_method': payment_method_id,
        },
    )
    sub = get_subscription(self.request.user)
    stripe_sub = stripe.Subscription.modify(
        sub.id,
        cancel_at_period_end=False,
        items=[
            {
                'id': sub['items'] ['data'] [0].id,
                'plan': desired_plan.stripe_plan_id,
            }
        ],
        payment_behavior='allow_incomplete',
        expand=['latest_invoice.payment_intent'],
    )
    status = stripe_sub.status
    if status == "active" and stripe_sub.latest_invoice.payment_intent.status == "succeeded":
        # payment succeeds, provision goods
        provision_goods(self.request.user, desired_plan)
        messages.add_message(request, messages.SUCCESS, 'Payment successful!')
        return redirect(reverse('subscriptions:checkout_success'))
    elif status == "past_due" and stripe_sub.latest_invoice.payment_intent.status == "requires_action":
        # requires further action, ie authentication
        messages.add_message(request, messages.ERROR,
            'This payment method requires extra authentication.')
        request.session['client_secret'] = stripe_sub.latest_invoice.payment_intent.client_secret
        return redirect(reverse('subscriptions:checkout_authentication'))
    else:
        # card error usually, ie. payment has failed
        messages.add_message(request, messages.ERROR,
            'Payment method invalid! Please fill in another payment method!')
        return render(request, self.template_name)

```

*Фигура 3.27 - Обработка на плащане след POST заявка към изгледа,  
отговарящ за плащането*

Прекъсването на абонамент се реализира на няколко етапа. В изгледа първо се проверя дали потребителят вече не е прекъснал абонамента си, което е отразено в полето `downgrade_at_period_end` на модела `Subscription`. Ако полето е в стойност `False`, то се обръща на `True` и също така се вика методът `downgrade_stripe_sub` на абонамента, който прави заявка към Stripe за преминаване от платен план към безплатен такъв (Фигура 3.28). Тази

промяна се отразява в базата данни на приложението чак след като изтече времето, за което е платил потребителят, за да той може да се възползва от абонамента си докрай.

На всеки 24 часа се проверява дали са необходими промени в абонаментите на всички болници (Фигура 3.29). Ако болницата е подала заявка за прекъсване на абонамента, се проверява дали е изтекла валидността на абонамента и ако е, болницата се премества към безплатен абонамент и се премахват всички допълнителни права. Проверяват се и болниците с абонамент, чиято дата на изтичане (`current_period_end`) е в миналото. Ако в базата данни на Stripe абонаментът е с активен статус, то плащането за следващия месец е успешно и абонаментът автоматично се е подновил. Тогава срокът на абонамента в базата данни на проекта се удължава с месец и потребителят запазва разширените права. Ако пък абонаментът е със статус „unpaid“, „canceled“ или „past\_due“, то плащането е неуспешно (заради липса на достатъчно средства в разплащателния метод или поради друга причина). Тогава се преминава към безплатен план както в базата данни на Stripe, така и в базата данни на проекта.

```
class SubscriptionCancelView(LoginRequiredMixin, PermissionRequiredMixin, View):
    permission_required = "common.is_hospital"
    raise_exception = True
    redirect_unauthenticated_users = True

    def post(self, request, *args, **kwargs):
        if self.request.user.hospital.subscription.downgrade_at_period_end:
            messages.add_message(self.request, messages.INFO,
                                'Your subscription has already been canceled. You will not be charged at the end of your current billing period.')
            return redirect('common:home')

        self.request.user.hospital.subscription.downgrade_stripe_sub()

        self.request.user.hospital.subscription.downgrade_at_period_end = True
        self.request.user.hospital.subscription.save()

        messages.add_message(self.request, messages.INFO,
                              'Your subscription has been canceled. You will not be charged at the end of your current billing period.')
        return redirect('common:home')
```

*Фигура 3.28 – Изгледът, отговарящ за отказ от абонамент*

```

def update_subscriptions(hospital_model, plan_model):
    hospitals_with_canceled_subs = hospital_model.objects.filter(subscription__downgrade_at_period_end=True)
    for hospital in hospitals_with_canceled_subs:
        if hospital.subscription.current_period_end <= datetime.today().date():
            hospital.move_to_free_plan()

    hospitals_with_overdue_payments = hospital_model.objects.filter(
        subscription__current_period_end__lte=date.today() + relativedelta(days=-7)
    )
    for hospital in hospitals_with_overdue_payments:
        sub_status = hospital.subscription.get_status
        if sub_status == 'past_due' or sub_status == 'canceled' or sub_status == 'unpaid':
            # move to free subscription
            hospital.subscription.downgrade_stripe_sub()
            hospital.move_to_free_plan()
        elif sub_status == 'active':
            hospital.subscription.current_period_end = date.today() + relativedelta(days=30)
        pass

```

*Фигура 3.29 - Функция, периодически проверяющая за истекли абонаменти*

# Четвърта глава

## Ръководство за потребителя

### 4.1 Системни изисквания

Приложението е тествано и работи под Windows 10. Не е тествано под Mac и Ubuntu. Единственото допълнително изискване е да бъде инсталирано приложението Docker Desktop (съответно Docker Engine за Ubuntu) и то да бъде стартирано преди стартиране на проекта. Подробни инструкции за инсталиране на Docker Desktop/Engine могат да бъдат намерени на следните линкове:

- За Windows: <https://docs.docker.com/docker-for-windows/install/>
- За Mac: <https://docs.docker.com/docker-for-mac/install/>
- За Ubuntu: <https://docs.docker.com/install/linux/docker-ce/ubuntu/>

### 4.2 Инсталиране и стартиране на приложението

За да се свали и създаде контейнерът, в който работи приложението, в папката `med_analyser` трябва да се изпълни следната команда.

```
docker-compose up --build
```

Инсталацията отнема няколко минути поради необходимостта от сваляне и инсталиране на всички модули в контейнера. След завършване на инсталацията, в терминала ще се изпише текстът от Фигура 4.1, с настоящата дата и час. Тогава вече приложението може да се достъпи от браузър на адрес:

```
http://localhost:8000/
```

```

web_1 | Watching for file changes with StatReloader
web_1 | Performing system checks...
web_1 |
web_1 | System check identified no issues (0 silenced).
web_1 | February 29, 2020 - 14:58:47
web_1 | Django version 2.2.10, using settings 'med_analyser.settings'
web_1 | Starting development server at http://0.0.0.0:8000/
web_1 | Quit the server with CONTROL-C.

```

*Фигура 4.1 - Успешно стартиране на уеб сървъра*

### 4.3 Регистрация, вход и изход от потребителски акаунт

При посещение на началната страница на уебсайта потребителят може да избере да влезе в профила си или да се регистрира през бутоните в най-дясната част на навигационната лента в горния край на екрана (Фигура 4.2).



*Фигура 4.2 - Бутони за вход и регистриране на акаунт*

След натискане на бутона „Sign up”, се зарежда страница с полета, в които се въвеждат своите лични данни, за да му бъде създаден акаунт.

Ако регистрацията е успешна, потребителят автоматично бива логнат в новосъздадения си акаунт (Фигура 4.3).

След натискане на бутона „Log in”, се зарежда страница с полета за въвеждане на имейл и парола на създаден акаунт. Потребителят има избор дали да остане логнат дори след напускане на сайта (Фигура 4.4).

Medical Analysis Platform

Pricing Log in Sign up

### Sign up

E-mail\*

E-mail address

First name\*

Last name\*

Password\*

Password

Password (again)\*

Password (again)

Sign up

*Фигура 4.3 - Регистриране на акаунт*

Medical Analysis Platform

Pricing Log in Sign up

### Log In

E-mail\*

E-mail address

Password\*

Password

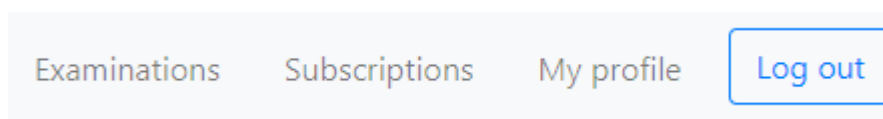
☐ Remember Me

Log In

*Фигура 4.4 - Вход в акаунт*

След като потребителят вече е в своя акаунт, в навигационната лента се появяват няколко нови възможности (Фигура 4.5). Чрез бутона „Log out”

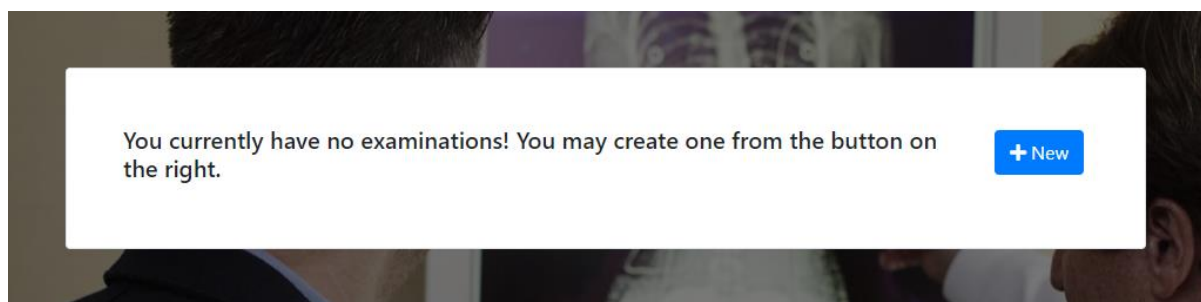
се излиза от настоящия акаунт, а “My profile” препраща към страница, в която е поместена информация за профила на потребителя и възможността да профилът да бъде изтрит.



*Фигура 4.5 - Навигационната лента за логнат потребител*

#### **4.4 Създаване и управление на изследвания**

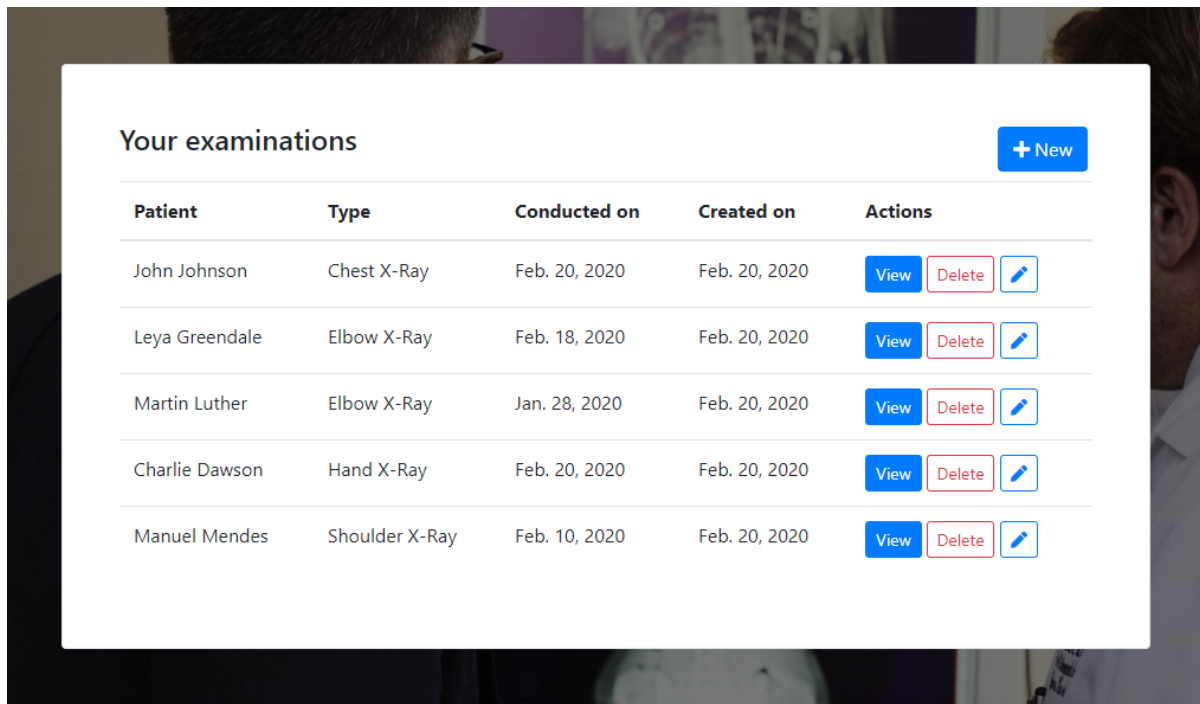
При избиране на „Examinations” от навигационната лента, ако потребителят няма нито едно създадено изследване, бива препратен към страница с информативно известие и бутон за създаване на ново изследване (Фигура 4.6).



*Фигура 4.6 – Съобщение при липса на създадени от потребителя изследвания*

Ако изследвания са налични, то ще се извежда списък с всички изследвания и част от информацията за всяко едно (Фигура 4.7). От този екран също чрез бутоните „View”, “Delete” и бутона с моливче може всяко

изследване съответно да се прегледа по-подробно, да се изтрие или да се редактира.



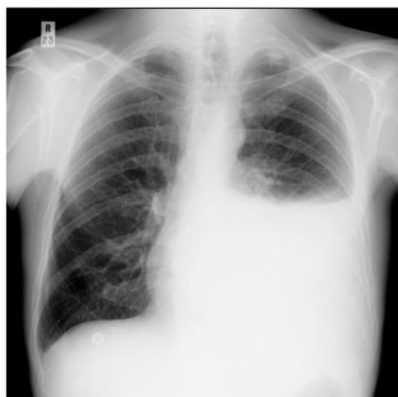
Your examinations					<a href="#">+ New</a>	
Patient	Type	Conducted on	Created on	Actions		
John Johnson	Chest X-Ray	Feb. 20, 2020	Feb. 20, 2020	<a href="#">View</a> <a href="#">Delete</a> <a href="#">Edit</a>		
Leya Greendale	Elbow X-Ray	Feb. 18, 2020	Feb. 20, 2020	<a href="#">View</a> <a href="#">Delete</a> <a href="#">Edit</a>		
Martin Luther	Elbow X-Ray	Jan. 28, 2020	Feb. 20, 2020	<a href="#">View</a> <a href="#">Delete</a> <a href="#">Edit</a>		
Charlie Dawson	Hand X-Ray	Feb. 20, 2020	Feb. 20, 2020	<a href="#">View</a> <a href="#">Delete</a> <a href="#">Edit</a>		
Manuel Mendes	Shoulder X-Ray	Feb. 10, 2020	Feb. 20, 2020	<a href="#">View</a> <a href="#">Delete</a> <a href="#">Edit</a>		

Фигура 4.7 – Списък с всички изследвания на потребителя

При преглед на изследване освен че на потребителя се показва цялата информация за изследването (име на пациента, медицинско изображение, визуализация на „горещите точки“ в изображението, вид на изследването, предположения за диагноза, бележки и дата на създаване; *Фигура 4.8*), има възможност и за отбелязване на правилна диагноза чрез натискане на бутона „Mark correct findings“ (*Фигура 4.9*). При натискането му потребителят бива препратен към две последователни страници, на първата от които може да отбележи дали в изследването има поне една аномалия. Ако отбележи, че има, той бива препратен към втората страница, на която може да избере всички аномалии, които е открил.



Peter Bellion

Image type: **Chest X-Ray**Our algorithms are 83.84% sure that there is **at least one** anomaly within the image.

The findings our algorithm predicted:

Effusion: 94.78%

Atelectasis: 16.78%

Infiltration: 14.54%

Consolidation: 6.22%

Mass: 5.49%

Pleural\_Thickening: 5.19%

Nodule: 4.54%

Pneumothorax: 2.42%

Emphysema: 0.95%

Fibrosis: 0.88%

Edema: 0.48%

Pneumonia: 0.34%

Cardiomegaly: 0.11%

Hernia: 0.02%

Do you see a problem with these findings?

[Mark correct findings](#)**Notes**

Might have to check his left lung. Possible signs of severe effusion.

Study created on March 23, 2020

*Фигура 4.8 - Изглед в детайли на едно изследване*

Do you see a problem with these findings?

[Mark correct findings](#)*Фигура 4.9 - Бутон за отбелязване на правилна диагноза*

## 4.5 Администраторски панел

За създаване на нова болница е необходимо да се достъпи администраторския панел от следния адрес:

<http://localhost:8000/admin>

При прилагане на миграционните файлове с цел по-лесно тестване автоматично се създава администраторски профил с потребителско име super1 и парола 123456superuser. След като се въведат тези данни, потребителят бива препратен към администраторския панел. От него могат да се преглеждат, добавят, редактират и премахват записи от повечето таблици в базата данни.

За да бъде добавена нова болница, трябва да бъде натиснат бутонът “Add” вдясно от “Hospitals” (Фигура 4.10).

COMMON	
Doctors	<a href="#">+ Add</a> <a href="#">✎ Change</a>
Hospitals	<a href="#">+ Add</a> <a href="#">✎ Change</a>

*Фигура 4.10 – Бутони за добавяне и редактиране на болници и доктори*

След това се показва форма, в която се въвежда цялата необходима информация за болницата – потребител, име на болницата, описание и имейл домейн адрес (Фигура 4.11). Важно е да се спомене, че е необходимо предварително потребителят, който желае да се регистрира като болница, да е създал акаунта си по стандартния начин за създаване на акаунт, описан в т. 4.3. След като е въведено всичко, се натиска бутонът “Save” в долния десен ъгъл. Така болницата е създадена и автоматично към нея са добавени всички доктори със съвпадащ имейл домейн адрес. Вече акаунтът, който е бил избран, може да достъпи функционалността, специфична за болници.

## Add hospital

The form is titled "Add hospital". It contains four input fields: "User:" with a dropdown menu, "Name:" with a text box, "Description:" with a large text area, and "Email domain:" with a text box. At the bottom, there are three buttons: "Save and add another", "Save and continue editing", and "SAVE".

*Фигура 4.11 - Форма за създаване на болница*

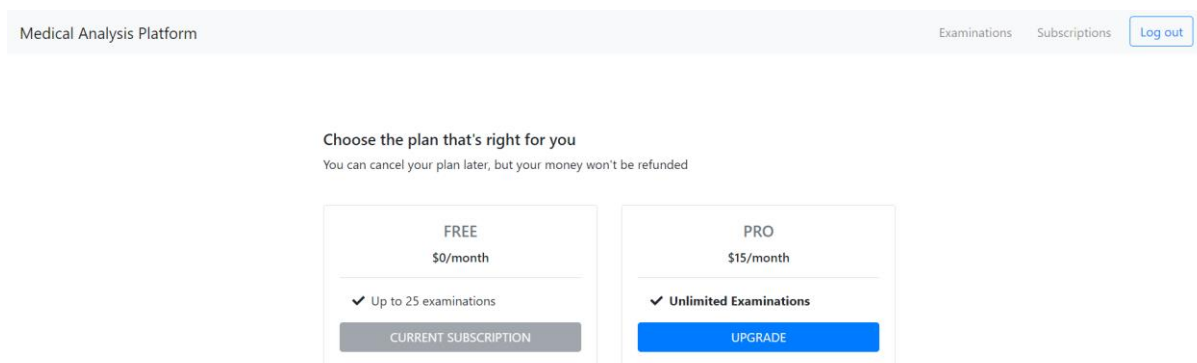
## 4.6 Закупуване на платен абонамент

Забележка: Тази функционалност е достъпна само за болници.

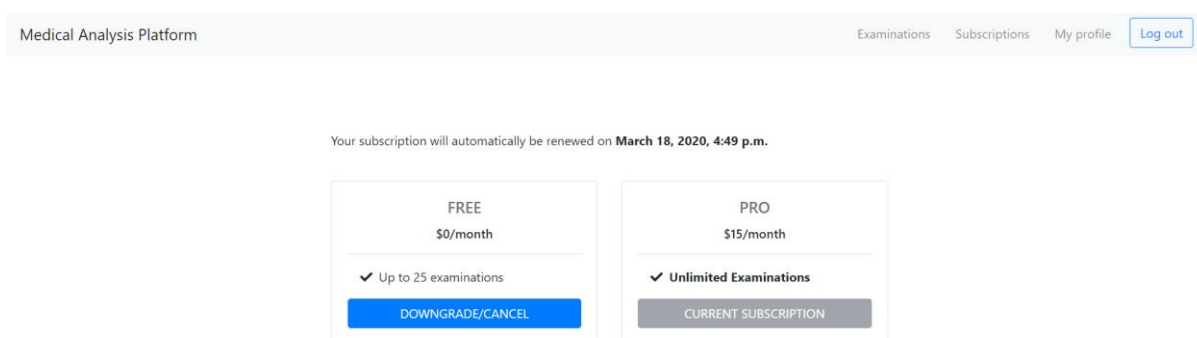
Линкът „Subscriptions” в основната навигационна лента води към страница, на която е представена основна информация за възможността за абониране (Фигура 4.12). Съдържанието на страницата варира в зависимост от това дали потребителят е абониран, дали е пожелал да прекъсне абонамента си и дали му остава време от абонамента, за който е заплатил. След закупуване на абонамент всички доктори, които са част от болницата, ще могат да създават и управляват неограничен брой изследвания. Ограничението за безплатен абонамент е 25 изследвания.

В случай, че потребителят е закупил абонамент, на страницата вижда информация за това кога неговият абонамент предстои да бъде автоматично подновен и ще има възможността да го прекъсне (Фигура 4.0.13). В случай,

че реши да го прекъсне, продължава да има всички привилегии, които му носи абонаментът, до месец след датата на закупуване, след което те му биват отнети и потребителят може да закупи наново абонамент.



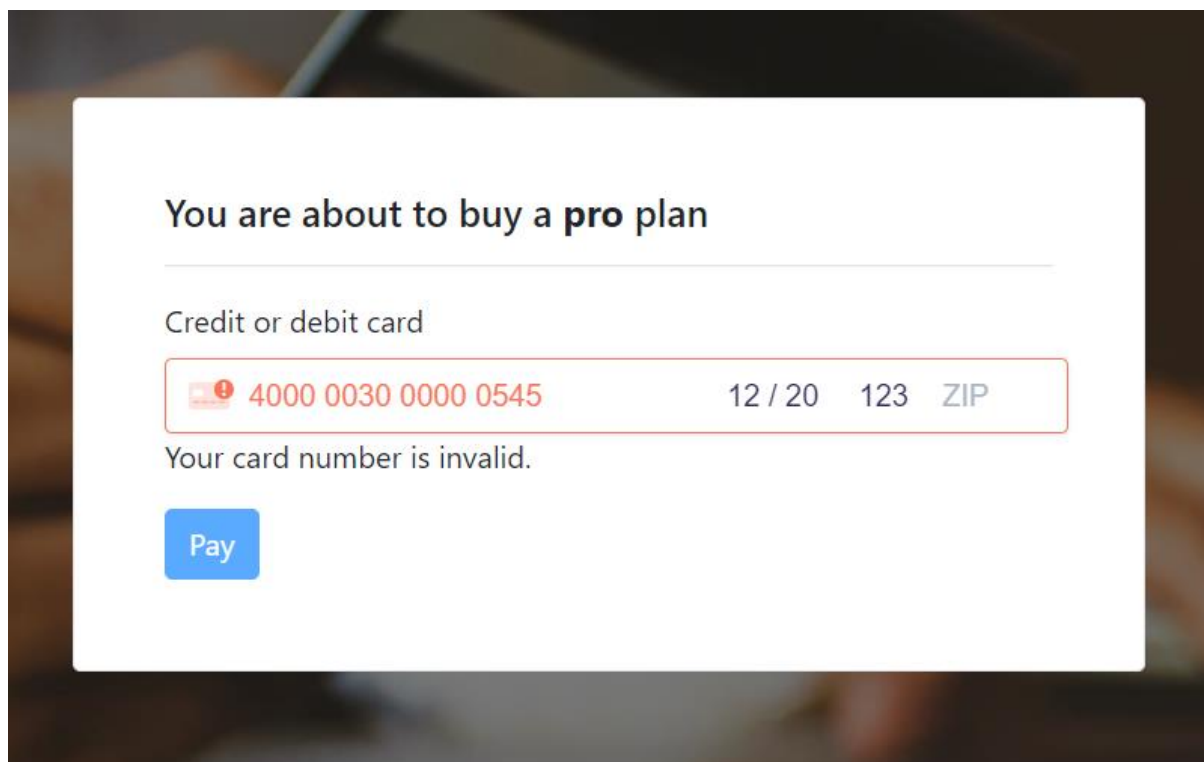
Фигура 4.12 - Екран за управление на абонамента преди закупуването му



Фигура 4.0.13 - Екран за управление на абонамента при закупен такъв

При закупуване на абонамент, потребителят въвежда данните за своята карта – номер на картата, месец и година на изтичане и CVC. За тестове цели могат да се използват картите, посочени в документацията на Stripe. Потребителят мигновено бива уведомен, ако някоя част от информацията е невалидна (Фигура 4.14). В случай, че картата изисква допълнителна автентикация чрез 3D Secure, на следващия екран плащането

трябва да се потвърди. При успешно приключване на цялата процедура, потребителят получава известие, че плащането е начислено и бива препратен към страницата, от която може да прекъсне новия си абонамент.



*Фигура 4.14 - Съобщение при невалиден номер на карта*

## Заклучение

Изискванията към дипломната работа бяха изпълнени. Създадено е уеб приложение, което може да бъде използвано от болници за улеснение на работата на техните радиолози чрез анализ и предложения за диагнози. Приложението има лесен за употреба интерфейс, с който потребителите биха свикнали лесно. Беше имплементиран безопасен и сигурен метод за закупуване на специален абонамент, който отключва разширена функционалност.

Проектът ще се подобрява след като започне реалното му използване от специалисти поради възможността на невронните мрежи автоматично да се дообучават и да поставят все по-точни диагнози.

Ядрото на проекта е така разработено, че да позволява надграждане. Посоките, в които това надграждане би могло да се случи, са няколко – подобряване на невронните мрежи, разпознаване на повече аномалии в повече видове изследвания, съхранение на повече информация за самите изследвания и допълнителни възможности за по-лесно управление на изследванията като например сортирането им и филтриране по избрани от потребителя параметри.

## Исползвана литература

1. <https://www.oreilly.com/content/machine-learning-a-quick-and-simple-definition/>
2. <https://www.ajronline.org/doi/full/10.2214/AJR.16.16963>
3. [https://developer.mozilla.org/en-US/docs/Learn/Server-side/First\\_steps/Web\\_frameworks](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks)
4. <https://insights.stackoverflow.com/survey/2019>
5. <https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fcd39170c>
6. <http://neuralnetworksanddeeplearning.com/>
7. <https://deeptai.org/machine-learning-glossary-and-terms/rectified-linear-units>
8. Maxwell Stinchcombe, Halber White. Multilayer Feedforward Networks are Universal Approximators.
9. <https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/>
10. <https://openai.com/blog/openai-pytorch/>
11. <https://docs.docker.com/engine/reference/builder/>
12. <https://docs.docker.com/compose/>
13. <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html>
14. <https://docs.djangoproject.com/en/2.2/ref/applications/>
15. <https://docs.djangoproject.com/en/2.2/>
16. <https://stripe.com/docs>
17. <https://stripe.com/docs/testing>
18. <https://www.dataschool.io/roc-curves-and-auc-explained/>
19. <https://docs.fast.ai/>

20. Wang X, Peng Y, Lu L, Lu Z, Bagheri M, Summers RM. ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases. IEEE CVPR 2017
21. Pranav Rajpurkar, Jeremy Irvin, Aarti Bagul, Daisy Ding, Tony Duan, Hershel Mehta, Brandon Yang, Kaylie Zhu, Dillon Laird, Robyn L. Ball, Curtis Langlotz, Katie Shpanskaya, Matthew P. Lungren, Andrew Y. Ng. MURA: Large Dataset for Abnormality Detection in Musculoskeletal Radiographs



## Съдържание

Първа глава .....	3
1.1 Обзор на подобни продукти.....	3
1.1.1 Системи за управление на медицински изследвания .....	3
1.1.1.1 Lify.....	3
1.1.1.2 AbbaDox Rad .....	4
1.1.1.3 QMENTA Trials .....	4
1.1.1.4 Merge PACS.....	4
1.1.2 Системи за анализ на изображения от медицински изследвания .....	5
1.1.2.1 Project InnerEye .....	5
1.1.2.2 ProFound AI .....	6
1.1.2.3 Arterys .....	6
1.2 Обзор на начини и технологии за реализиране на уеб приложение ....	7
1.3 Обзор на начини и технологии за реализиране на дълбоки невронни мрежи .....	10
1.3.1 Развойна среда и необходим хардуер .....	10
1.3.2 Програмни езици и библиотеки за създаване и трениране на дълбоки невронни мрежи.....	13
Втора глава .....	15
2.1 Функционални изисквания.....	15
2.2 Избор на среда за програмиране.....	16
2.2.1 Visual Studio Code .....	16
2.2.2 Jupyter Notebook.....	17
2.3 Избор на технологии за разработка на уеб приложението .....	18
2.3.1 Python .....	18

2.3.2 Miniconda.....	18
2.3.3 Django .....	18
2.3.4 PostgreSQL .....	20
2.3.5 Docker .....	20
2.3.6 Stripe .....	21
2.3.7 Bootstrap 4 .....	21
2.4 Избор на технологии за създаване на невронната мрежа.....	21
2.4.1 Принцип на работа на невронна мрежа .....	21
2.4.2 Google Cloud Platform (GCP) .....	25
2.4.3 PyTorch .....	25
2.4.4 fastai .....	26
2.5 Структура на базата данни .....	27
Трета глава .....	32
3.1 Конфигуриране на Docker .....	32
3.2 Създаване и трениране на дълбоките невронни мрежи .....	34
3.2.1 Невронна мрежа за разпознаване на типа снимка .....	34
3.2.2 Невронни мрежи за разпознаване на аномалии в рентгенова снимка на гръден кош .....	37
3.3 Описание на функционалността на приложенията в Django .....	42
3.3.1 Обяснение на някои концепции в Django .....	43
3.3.2 Приложението “common” .....	44
3.3.3 Приложението “examinations” .....	47
3.3.4 Приложението “subscriptions” .....	52
Четвърта глава.....	58

4.1 Системни изисквания .....	58
4.2 Инсталиране и стартиране на приложението .....	58
4.3 Регистрация, вход и изход от потребителски акаунт .....	59
4.4 Създаване и управление на изследвания .....	61
4.5 Администраторски панел .....	63
4.6 Закупуване на платен абонамент .....	65
Заклучение.....	68
Използвана литература .....	69