**Міністерство освіти і науки України**

**Харківський національний університет радіоелектроніки**

Факультет комп’ютерних наук

Кафедра Програмної інженерії

**МАГІСТЕРСЬКА АТЕСТАЦІЙНА РОБОТА**пояснювальна записка

|  |
| --- |
| Методи вилучення метаінформації з медіа-файлів |
| (Тема роботи) |

|  |  |
| --- | --- |
| Магістрант гр. ІПЗм-14-1 | Бікташев Т.І |
| Керівник роботи доц. | Каук В.І. |
| Допускається до захисту Зав. кафедри, проф. | Дудар З.В. |

2016 р.

*Харківський національний університет радіоелектроніки*

|  |  |
| --- | --- |
| Факультет *комп’ютерних наук.* | Кафедра *Програмної інженерії* |
| Спеціальність | *8.05010302 Інженерія програмного забезпечення* |
|  | ЗАТВЕРДЖУЮ |
|  | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_20\_\_\_року  В.о. Зав. кафедри проф. З.В.Дудар |

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ АТЕСТАЦІЙНУ РОБОТУ

магістрантові

***Бікташеву Тимуру Ільдаровичу***

1. Тема проекту (роботи) «Методи вилучення метаінформації з медіа-файлів» керівник проекту Каук Віктор Іванович, кандидат технічних наук, доцент, затверджені наказом вищого навчального закладу №\_1363\_ від «23»\_жовтня\_ 2015 р.
2. Строк подання студентом проекту «12» січня 2016 р.
3. Вихідні дані до проекту (роботи):дані предметної галузі, набори підписаних фото, середовище проектування Microsoft Visual Studio 2015, мова розробки C#.
4. Зміст розрахунково-пояснювальної записки:вступ, аналіз проблемної області, моделювання та аналіз, технології, програмна реалізація, висновки*.*
5. Перелік графічного матеріалу: титульне частина, мета роботи аналіз предметної області, постановка задачі, ієрархічна структура об'єктів, структура штучного нейрону, Relu функція, персептрон, архітектура згортальної нейронної мережі, шар масштабування, аргументні архітектури мереж, діаграма компонентів системи, діаграми бізнес сутностей, діаграма послідовностей для процесу завантаження файлів діаграма класів FilePreprocessor, використані технології, висновки*.*

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Назва етапів магістерської атестаційної роботи | Строк виконання етапів проекту | Примітка |
| 1 | Об'єктний аналіз поставленої задачі | 26.10.2015 - 28.10.2015 | виконано |
| 2 | Розробка моделі взаємодії даних | 28.10.2015 - 31.10.2015 | виконано |
| 3 | Розробка структури зберігання даних | 01.11.2015 - 03.11.2015 | виконано |
| 4 | Створення коду програми | 03.11.2015 - 26.11.2015 | виконано |
| 5 | Тестування і налагодження програми | 27.11.2015 - 13.12.2015 | виконано |
| 6 | Підготовка пояснювальної записки. | 14.12.2015 - 22.12.2015 | виконано |
| 7 | Підготовка презентації та доповіді | 23.12.2015 | виконано |
| 8 | Попередній захист | 29.12.2015 | виконано |
| 9 | Нормоконтроль, рецензування | 11.01.2016 | виконано |
| 10 | Занесення диплома в електронний архів | 12.01.2016 | виконано |
| 11 | Допуск до захисту у зав. кафедри | 12.01.2016 | виконано |

Дата видачі завдання «\_23\_\_» \_\_\_**\_**жовтня**\_\_\_\_\_** 2015 р.

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доц. Каук В.І.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Бікташев Т.І.

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до випускної кваліфікаційної роботи магістра містить: 95 с., 25 рис., 3 додатки, 20 джерел.

Об’єктом дослідження є великі бази медіафайлів, що не є класифікованими та не мають підписів та опису, як набору фотографій у соцмережах та відеозаписи із камер зовнішнього спостереження.

Метою роботи є розробка системи розпізнавання об'єктів, зображених на медіа-файлах.

Методи розробки базуються на мові програмування С#, .NET Framework, СУБД SQL Server, платформи ASP.NET MVC 4, WCF 4.5, з використанням інтегрованого середовища розробки Microsoft Visual Studio 2015.

У результаті роботи створено інформаційну систему, яка дає можливість виділяти об'єкти на зображеннях та класифікувати їх, для подальшого аналізу мета-інформації всього стеку медіа-файлів.

C#, VISUAL STUDIO, SQL, СУБД, MICROSOFT, .NET FRAMEWORK, ASP.NET MVC 4, WCF 4.5, HTML 5, CSS 3, JAVASCRIPT, НЕЙРОННА МЕРЕЖА, ЗОБРАЖЕННЯ, МЕТА-ІНФОРМАЦІЯ ЗОБРАЖЕНЬ.

Explanatory note to the final qualifying bachelor contains: 95 p., 25 fig., 3 appendix, 20 sources.

Object is a great base media, which is not classified and have no signatures and describing how a set of photos and videos in social networks of surveillance cameras.

The aim is to develop a system of recognition of objects depicted in the media.

Methods based on the development of the programming language C #, .NET Framework, DBMS SQL Server, ASP.NET MVC 4 platform, WCF 4.5, using an integrated development environment Visual Studio 2015 Microsoft.

As a result of an information system that makes it possible to select objects in images and classify them for further analysis meta information stack all media files.

C#, VISUAL STUDIO, SQL, СУБД, MICROSOFT, .NET FRAMEWORK, ASP.NET MVC 4, WCF 4.5, HTML 5, CSS 3, JAVASCRIPT, NEURAL NETWORKS, IMAGE, IMAGE LABEL.

ЗМІСТ

|  |  |
| --- | --- |
| Вступ ……………………………………………………………....................………........ | 5 |
| 1 Аналіз предметної області..……………………..........................……………............... | 7 |
| 1.1 Виявлення проблем………………………………....………………….................. | 7 |
| 1.2 Постановка задачі…………………………………………...................................... | 11 |
| 2 Моделювання та аналіз ……………………………………………................................ | 13 |
| 2.1 Загальна архітектура системи……....……....……....……....……....……....…….... | 13 |
| 2.2 Модуль даних……....……....……....……....……....……....……....……....…….... | 14 |
| 2.3 Нереляційне сховище даних……....……....……....……....……....……....…….... | 18 |
| 2.4 Модуль розпізнавання……....……....……....……....……....……....……....…….... | 20 |
| 2.4.1 Методи розпізнавання……....……....……....……....……....……....……....…….... | 20 |
| 2.4.2 Аналіз реальних об'єктів і біологічні нейронні мережі……....……....……....… | 25 |
| 2.4.3 Згортальна нейронна мережа……....……....……....……....……....……....…….... | 32 |
| 2.4.4 Похибка штучної нейронної мережі……....……....……....……....……....…….... | 33 |
| 2.4.5 Навчання штучної нейронної мережі .............……....……....……....……....…….... | 36 |
| 2.4.6 Доопрацювання ……....……....……....……....……....……....……....……....…….... | 40 |
| 2.4.7 Відео……....……....……....……....……....……....……....……....……....……....…… | 42 |
| 2.4.8 Масштабування……....……....……....……....……..……....……....……....…….... | 43 |
| 2.4.9 Аргументна архітектура……....……....……....……....……....……....……....…….... | 45 |
| 3 Програмна реалізація …………………....…………..............…………………………… | 48 |
| 3.1 WCF сервіс і розподіл спільної пам'яті…………………………………..................... | 48 |
| 3.2 Компонент FilePreprocessor………………………………………………..................... | 52 |
| 3.2 Нейрон штучної нейронної мережі ....…………………………………..................... | 53 |
| 3.3 Набори даних …………………………………………………….............................. | 54 |
| Висновки ..…………………………………………………………………........................ | 55 |
| Перелік посилань …………………………………..........................…………………..... | 56 |
| Додаток А Слайди презентації …………………………………………..…………....... | 57 |
| Додаток Б Програмний код .................................................................................................. | 75 |
| Додаток В Апробація результатів роботи............................................................................ | 91 |

ВСТУП

У сучасний час, в століття IT-технологій і швидко зростаючого обсягу даних, стає необхідною обробка величезних обсягів даних для отримання з них корисної інформації. Стало зрозуміло, що обробляти такий обсяг даних існуючих даних стало неможливим або це займає неприйнятно великий проміжок часу. Часто виникає потреба в пошуку конкретних об'єктів. Також у великих містах знаходиться величезна кількість відеокамер. Цілком логічно використовувати їх для вирішення виникаючих завдань.

Для вирішення цієї проблеми було проведено дослідження з розробки розподіленої системи з обробки великого обсягу медіа-матеріалів, який надається їй для вирішення конкретного завдання. Прикладом такого завдання є побудова статистики по вмісту відеозаписів і фото в соц. мережах та інших місцях зберігання вмісту і його надання для перегляду іншим користувачам. При цьому метою такого аналізу може бути визначення потенційно небезпечних акаунтів, які порушують політику даного соц. порталу або права інтелектуальної власності.

Обробка вхідних даних буде виконуватися на кластері, що складається з безлічі серверів. Кількість серверів масштабується в залежності від навантаження. Для задоволення цієї вимоги в якості платформи для розгортання використовується Microsoft Azure з сервісом HDInsight, реалізований на базі Hadoop. В якості моделі для розподілених обчислень був обраний MapReduce, що є одним з модулем в Hadoop.

Для побудови статистичної інформації за вхідними даними, на початку на кожному робочому вузлі кластера відбувається виділення метаінформації з кожного окремого зображення або кадру і передачі її у вигляді наборів тегів. На аналізованому зображенні виділяються прості геометричні об'єкти такі як лінії. Надалі використовуючи штучні багатошарові нейронні мережі з кожним наступному шаром виконується розпізнавання більш складних графічних об'єктів [1]. Дані отримані в результаті з вихідних шарів нейронних мереж у вигляді текстових тегів передаються майстер-вузлу для подальшого статистичного аналізу на етапі згортки. У рамках даної роботи виконано дослідження і визначення найбільш підходящих алгоритмів розподілу процесів обробки даних і розпізнавання метаданих в зображеннях і кадрах по визначеним користувачем правилах.

Ця робота є доцільною оскільки інтернет-портали, сервіси, подібні до Instagram, та особливо соцмережі, постійно наповнюються великим обсягом фотографій, що часто не підписані та не охарактеризовані. Ці осередки даних потребують автоматичного аналізу для покращення орієнтування користувачів в розподілених на ресурсі фотографіях та захисту сервісів від забороненого контенту.

Тема роботи відноситься до одного з наукових напрямів кафедри ПІ, а саме до напрямів інтелектуального аналізу даних та розпізнавання образів.

Метою роботи є створення веб-сервісу, що надає користувачам послугу з обробки великих об'ємів фотографій, розпізнаваннях зображених на них об'єктів та виконання поставлених користувачами запитів до цих наборів даних. Розробка WCF-сервісу та веб-додатку, як оболонки для нього, для пропозиції цього програмного забезпечення користувачеві за моделлю "програмне забезпечення як послуга".

Об'єктом дослідження є великі бази медіа-файлів, що потребують швидкої обробки та класифікації.

Предметом дослідження є виділення для окремих файлів мета-інформації, що описує зображення на фото- або відеоматеріалах, перелічуючи у текстовій формі об'єкти на ньому та їх характеристики.

Вибір алгоритму розпізнавання відбувався методом порівняння способу розпізнавання образів машиною із роботою мозку тварин [3], формалізацією алгоритму навчання штучної нейронної мережі і вичислення її похибки.

У цьому дослідженні дістав подальшого розвитку процес розпізнавання великих обсягів медіа-даних за допомогою архітектури нейронних мереж, базованої на згортальних шарах нейронів та глибокого графу мережі.

Результати цієї атестаційної роботи були опубліковані на 19-му Міжнародному молодіжному форумі "Радіоелектроніка та молодь у ХХІ столітті" на конференції "Комп'ютерні, програмні та інтернет-технології, програмування комп'ютерних мобільних систем" (див. додаток В).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Виявлення проблем

Для виявлення основних проблем та визначеннях напрямків для створення якісного продукту, було проведено наступний аналіз аналогів сервісів розпізнавання зображень, існуючих на ринку.

Google Goggles - це пошук по зображеннях для мобільного телефону (див. рис. 1.1). Користувач робить фотографію, телефон відсилає її в дата-центр Google, і, використовуючи спеціальний алгоритм, намагається розпізнати, що на ній зображено:

- якщо розпізнано текст, його можна перевести на будь-яку мову (використовуючи Перекладач Google), просто пошукати в Інтернеті, а також скопіювати в буфер обміну, щоб потім використати в іншому додатку;

- якщо сфотографували візитку, і Goggles розпізнає контактну інформацію, то можна зателефонувати за телефонним номером, написати е-мейл, перейти за посиланням сайту, знайти адресу на карті або додати людини в вашу адресну книгу;

- якщо розпізнана пам'ятка, можна знайти інформацію про неї в Інтернеті, а також знайти її на карті світу і дізнатися як до неї можна проїхати;

- якщо розпізнано витвір мистецтва, наприклад, картина, ви зможете дізнатися її назву, автора, музей, де вона знаходиться, і звичайно ж пошукати інформацію про неї в Інтернеті;

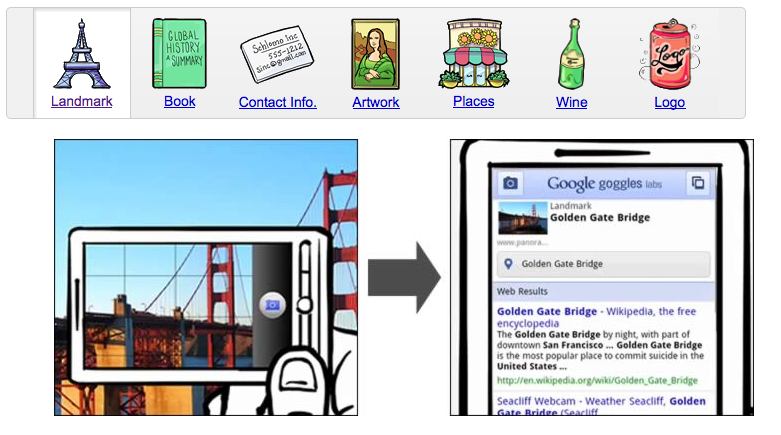
- якщо розпізнана книга, CD або DVD, можна побачити відгуки на них, а також дізнатися, де їх купити, використовуючи пошук Google;

- якщо сфотографувати штрих-код продукту, наприклад, лікарського препарату, то Goggles працює і як штрих-код сканер, і можна отримати додаткову інформацію про нього;

- якщо логотип розпізнано, можна дізнатися, якій організації він належить;

- якщо Goggles отримає на вхід розлінієну задачу судоку він розпізнає і вирішить її;

- якщо необхідно, система може просто шукати схожі зображення в Інтернеті.

Рисунок 1.1 - Google Goggles

Розробка з Wolfram Language (див. рис. 1.2). Для того, щоб розпізнати якесь зображення при роботі з Wolfram Language, потрібно просто застосувати функцію ImageIdentify до цього зображення:

На виході отримуємо деякий символьний об'єкт, з яким можна і далі працювати в Wolfram Language. Як у даному прикладі - з'ясувати, що це саме тварина, ссавець і так далі. Або просто запитати визначення. Такий запит ображено на рисунку нижче.

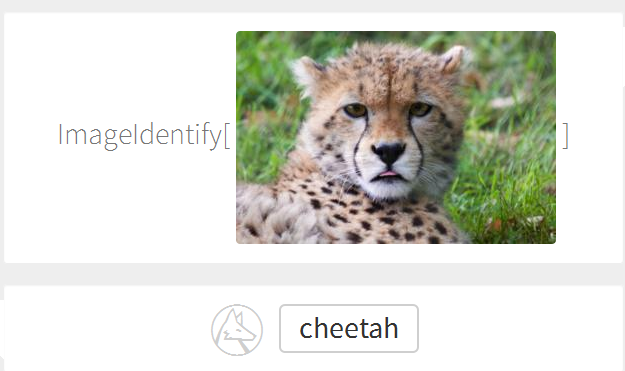


Рисунок 1.2 - Wolfram Language

За допомогою функції ImageIdentify, вбудованою безпосередньо в Wolfram Language (див. рис. 1.3), дуже легко створювати якісь API, додатки, в яких це використовується. А за допомогою Wolfram Cloud дуже легко створювати сайти - як, наприклад, сайт Wolfram Language Image Identification Project.



Рисунок 1.3 - Приклад запиту Wolfram Language

Недоліки системи: є особливості в навчанні, які дуже схожі на ті, що виникають для біологічних нейронних мереж. Приміром, можна допустити помилку, не помістивши зображення осіб людей в навчальну добірку. І якщо показати картинку, на якій був Індіана Джонс, то система взагалі не зрозуміє, що тут є якась особа і видає, що на картинці зображений капелюх.

Ймовірно, як і у мозку, у нейронної мережі ImageIdentify є безліч шарів, які містять нейрони різних типів (загальна структура, звичайно, добре описана символьним поданням Wolfram Language).

Важко сказати щось осмислене про те, що-ж відбувається всередині мережі. Але якщо заглянути у верхні шари, то можна буде виділити якісь риси, які розрізняє система. І, мабуть, вони вельми схожі з тими рисами, які розрізняються справжніми нейронами в первинної зорової корі.

Якщо користувачу потрібно знайти якийсь логотип, то одним із варіантів для нього буде пошукати в гуглі, але для цього треба було скласти якийсь опис побаченого логотипу і задати гуглу відповідні ключові слова. В іншому варіанті для пошуку за самою картинкою може бути використаний сервіс TinEye.

Сервіс TinEye - це «reverse image search engine». В нього користувачем завантажується картинку (або вказується URL до неї), а TinEye (див. рис. 1.4) говорить, де ж в мережі він її зустрічав. Розпізнавши картинку, він може виявити якісь фрагменти цієї картинки в інших картинках, або цю ж картинку, але іншого розміру або іншої якості.

Звичайно, він може не знайти картинку, навіть якщо вона є в мережі, з різних причин:

- відповідний веб-сайт ще не проіндексований;

- веб-сайт не проіндексовані, бо у файлі robots.txt сайту було заборонено залазити роботам;

- TinEye неправильно розпізнав картинку.

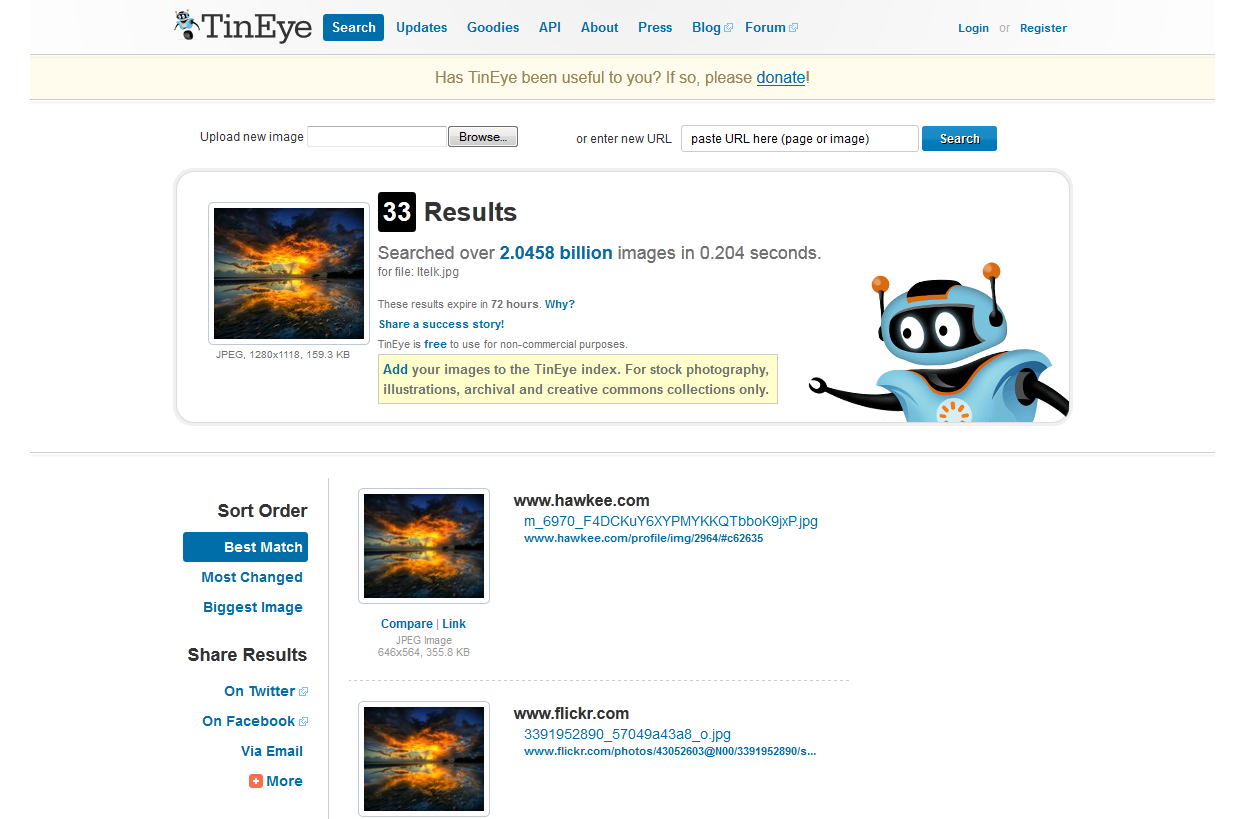


Рисунок 1.4 - TinEye

На даний момент проіндексовано близько 1,300 мільярда картинок, і ця кількість постійно зростає. Сервіс найбільше може бути використаний копірайтерам і власникам медіа (фото, зображення). Також є плагін для IE і Firefox'а та API.

Недоліком є відсутність додатку для iPhone'ов та інших мобільних пристроїв, які дозволяли б робити фотографію і відразу ж ініціювати з цього зображення пошук в TinEye.

1.2 Постановка задачі

У світі існує велика кількість найрізноманітніших речей, яким люди дали ім'я. Суть розпізнавання - визначити, які з цих речей представлені на цьому зображенні. Якщо більш формально - відобразити всі можливі зображення в деякий безліч імен об'єктів у символьній формі.

Тобто ми не маємо ніякого способу описати, наприклад, стілець. Але ми можемо навести безліч прикладів того, як виглядає стілець, ніби кажучи: «все, що схоже на ось це ось, нехай визначається системою як стілець». Тобто ми хочемо, щоб всі зображення, в яких міститься щось, схоже на стілець, мали відповідність зі словом стілець, а інші зображення не мали б такої відповідності.

Є безліч різних систем з подібним "аттракторним" поведінкою. Як приклад з фізичного світу можна навести схил гори. Краплі дощу можуть впасти на яку завгодно частина гори, однак (принаймні, в ідеальної моделі) вони будуть стікати вниз до максимально низько розташованим точкам. Краплі, які знаходяться поруч, будуть прагнути до одним і тим же точкам. Розташовані далеко один від одного краплі будуть текти до різних точок. Краплі дощу - як зображення, а точки підніжжя гори - як види об'єктів.

І точно таке ж аттракторное поведінку відбувається і в розпізнаванні. Приміром, є безліч клітинних автоматів, в якому кожен автомат може міняти кольори кількох сусідніх клітин, але все в будь-якому випадку закінчиться деяким стійким.

При обробці зображення, деякі звичні операції (на комп'ютерах або за допомогою візуального огляду людиною) є просто алгоритми двовимірних клітинних автоматів. Легко змусити клітинні автомати розрізнити деякі особливості зображення на зразок темних плям, наприклад. Однак для розпізнавання зображень потрібно щось більше. Якщо знову застосувати ту аналогію з горою, то нам потрібно створити схил гори з усіма його властивостями, щоб краплі з відповідної частини гори стікали на відповідну частину її підніжжя

Зараз, коли число штучних нейронів стало порівняно з числом нейронів у відповідних частинах нашого мозку. І справа не в тому, що кількість саме по собі щось означає. Скоріше справа в тому, що, якщо ми вирішуємо якісь завдання на зразок розпізнавання зображень, - ті завдання, які вирішує людський мозок - не дивно, що нам потрібна система відповідного масштабу.

Люди легко можуть розпізнавати тисячі самих різних об'єктів - приблизно стільки, скільки в мові є іменників, які можна якось зобразити. Інші тварини розрізняють набагато менше об'єктів. Але якщо ми спробуємо розпізнавати образи так, як це робить людина, ефективно відображаючи їх в слова, які є в людських мовах, то тоді ми зіткнемося з усім масштабом проблеми. Ключ до її вирішення - нейронна мережа масштабу людського мозку.

Безсумнівно, є відмінності між обчислювальними та біологічними нейронними мережами, хоча після того, як мережа навчена, процес отримання результату з образу виявляється досить схожий. Але методи, використовувані для навчання обчислювальних нейронних мереж, істотно відрізняються від тих, які мали б використовуватися в біологічних мережах.

Виходячи з аналізу предметної області та виявлених в ній проблем, задачею цього проекту становиться розробка системи обробки наборів фотографій за визначеними користувачем правилами. Зокрема розробка підсистеми розпізнавання об’єктів на фото та віднесення тегів до цих фото. В системі має бути реалізований веб-додаток для завантаження користувачем наборів медіа-файлів, модуль для аналізу зображення у вигляді веб-сервісу. Система повинна обробляти 1 млн. одиниць зображень за 1 годину та працювати не менш ніж з 100 класами зображень. Запит до бази даних об’ємом 100 ТБ повинен виконуватись не більш, ніж за 3 хвилини. Також повинна бути реалізована реплікація даних для забезпечення відмовостійкості системи.

2 МОДЕЛЮВАННЯ ТА АНАЛІЗ

2.1 Загальна архітектура системи

Для створення системи, що виконую необхідні, поставлені задачі проводиться моделювання системи та її окремих компонентів, з використанням UML діаграм [5]. У ході початкового аналізу було визначено основну структуру системи (див. рис. 2.1) і розподіл її на підсистеми.

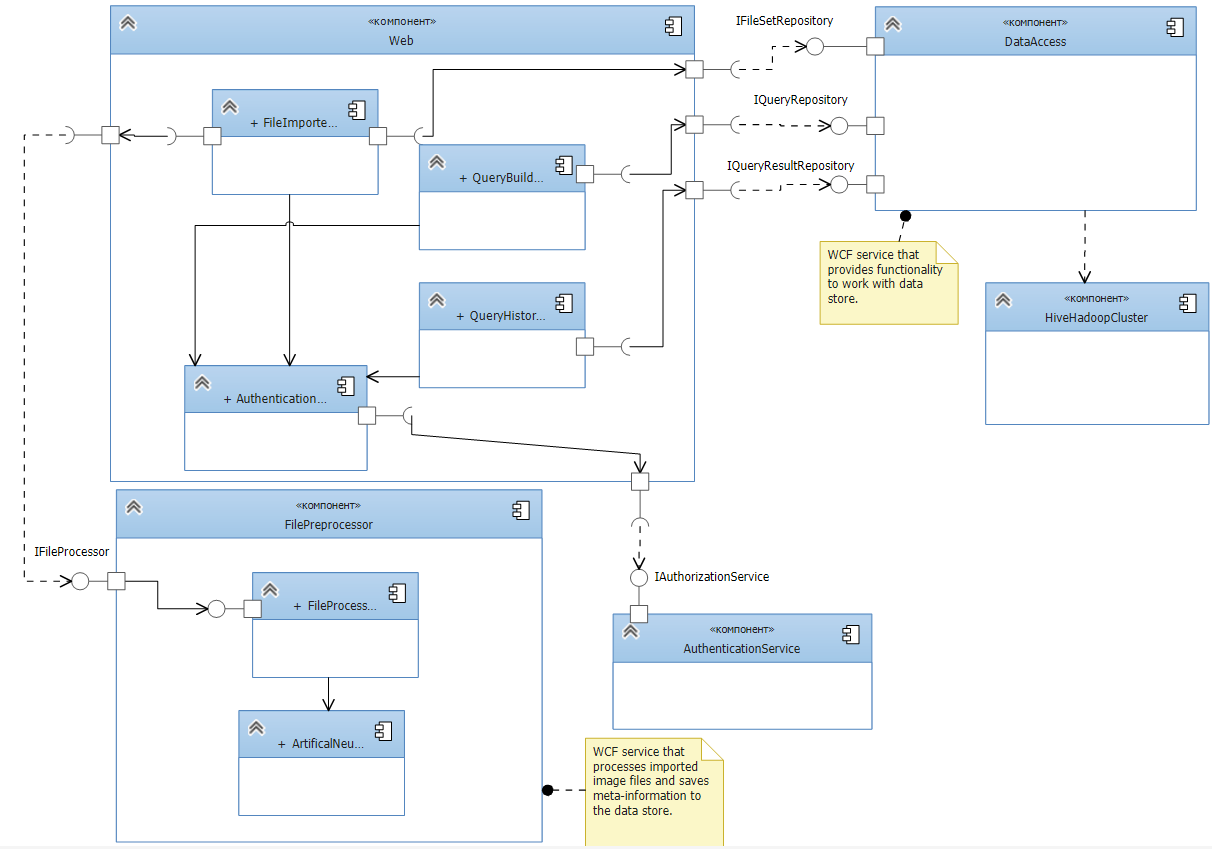


Рисунок 2.1 - Діаграма компонентів системи

Система розділяється на 4 основні компоненти:

- веб-додаток;

- модуль аутентифікації;

- модуль даних;

- модуль розпізнавання.

Кожен модуль представлений сервісом. Аутентифікація між модулями відбувається з допомогою цифрових сертифікатів SSL.

Модуль веб-додатку виконує роль основного, зв'язуючого компоненту системи. Взаємодія користувача із системою відбувається саме через цей модуль. Веб-додаток розробляється на фреймвоці ASP.NET MVC. Контролери додатку надають можливість реєструватися, входити у систему, завантажувати набори медіа-файлів, обробляти та аналізувати їх, виконувати оплати послуг системи. Має сторінки для роботи з конструктором запитів.

Аутентифікація в системі та модулях відбувається через модуль аутентифікації. Вона забезпечує можливість безпечного доступу до системи та розподілу ролей та прав.

2.2 Модуль даних

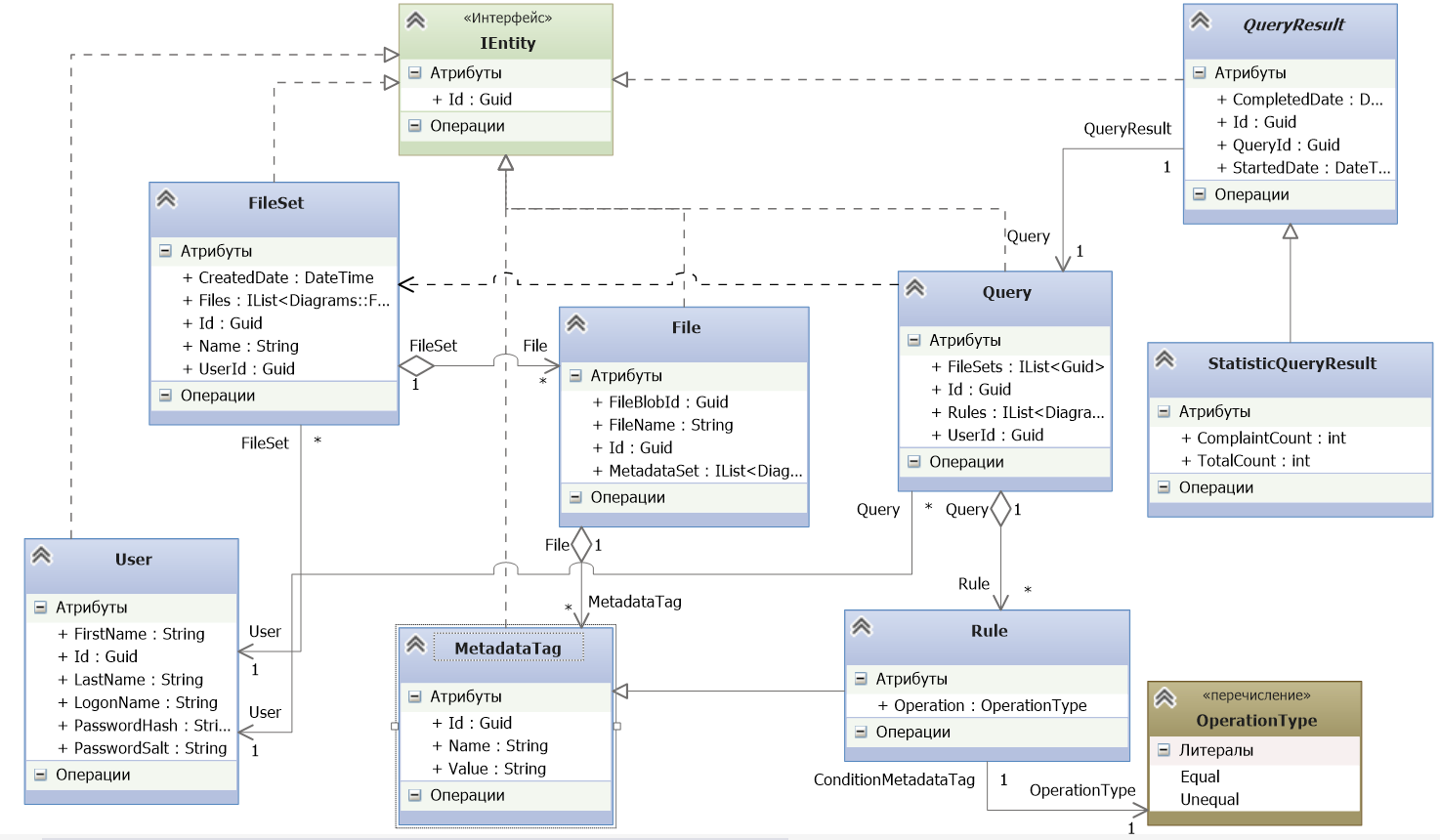
Модуль даних відповідає за зберігання даних основних сутностей системи. Для збереження даних використовується Hadoop та Microsoft SQL Server [7]. Далі наведена діаграма бізнес-сутностей (див. рис. 2.2), котрі контролює модуль.

Сутності, необхідні для роботи веб-додатка, так взаємодії користувача з системою зберігаються в звичайній реляційній SQL базі даних.

Основою є набори файлів FileSet, що потребують аналізу. Для кожного набору зберігається його ім'я Name та дата створення CreateUTC. Набор даних прив'язується до користувача, що завантажив його, через поле UserId. Файли набору зберігаються у окремому хранилищі.

Про користувача, що виступає власником наборів та тим що запрошую статистику з системи, інформація зберігається сутністю User.

Формулювання запитів користувачів зберігаються сутністю Query. Кожен запит зберігає інформацію про набори файлів FileSets, до яких він відбуватиметься, та про правила Rules, за якими він проходитиме.

Рисунок 2.2 - Діаграма бізнес-сутностей

Результати запиту зберігаються в абстрактній сутності QueryResult з фіксацією часу запуску запиту StartUTC та часу завершення виконання CompleteUTC. Конкретний вид запиту формує сутність StatisticQueryResult, що наслідує. абстрактну та зберігає інформацію про кількість файлів, що відповіли запиту ComplaintCount і загальну їх кількість TotalCount. При розширені потрібності у інших варіантів запитів, вони будуть створюватися додатково та наслідувати абстрактний клас.

Сам запит формулюється набором правил Rule, кожне з котрих зберігає тип MetadataTag.Name та значення MetadataTag.Value метатегу та типу операції Operation. Базовими операціями є прості "рівність" OperationType.Equal та "нерівність" OperationType.Unequal.

Сутності, котрі безпосередньо розбираються при запитах, зберігаються у Hadoop. Файл (File) його ім'я Name та ідентифікатор FileBlobId, що вказує на масив байтів цього файлу у Blob-об'єкті. Безпосередньо запити Hadoop відбуваються за властивістю файлу MetadataSet, у якому зберігаються тегами семантика відповідного фото-файлу. Це виконується через HiveHadoopCluster.

Для виконання запитів, користувач має спочатку завантажити свої файли до системи. Цей процес ілюстровано на наступній діаграмі (див. рис. 2.3).

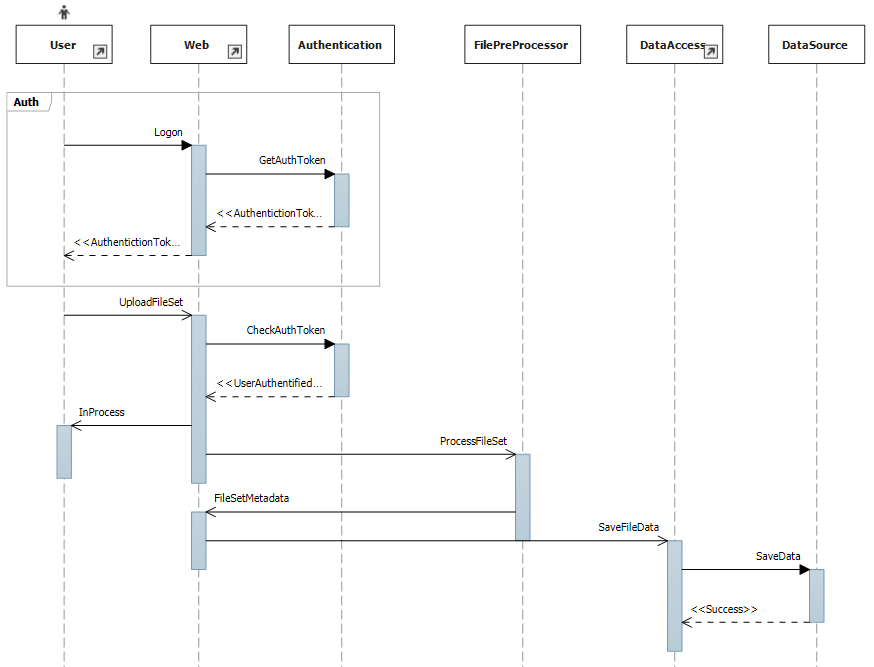


Рисунок 2.3 - Діаграма послідовностей для процесу завантаження файлів

Спочатку користувач завдяки веб-додатку передає в систему набір файлів. Перед цією операцією відбувається аутентифікація у веб-додатку через модуль аутентифікації та визначається дозвіл виконання цієї операції та отримується токен.

Далі данні файлів окремими потоками завантажуються через веб-додаток до модулю обробки медіа FilePreProcessor. При цьому при отриманні даних відбувається перевірка токену аутентифікації та відправка користувачу повідомлення про початок завантаження файлів. Модуль FilePreProcessor аналізую файли фотографій та виявляє в них семантику у вигляді тегів. Набори тегів до кожного файлу передаються через веб-додаток до модулю DataAccess та зберігаються у HiveHadoopCluster для подальших робіт та аналізів.

При виконанні користувачем запиту до завантажених наборів файлів (див. рис. 2.4) відбувається його аутентифікація за токеном у веб-додатку через модуль Authentication. Користувач набирає запит у конструкторі на веб-сторінці додатку, що відправляється в сервіс DataAccess. Де в Hadoop додається сформований запит до job.

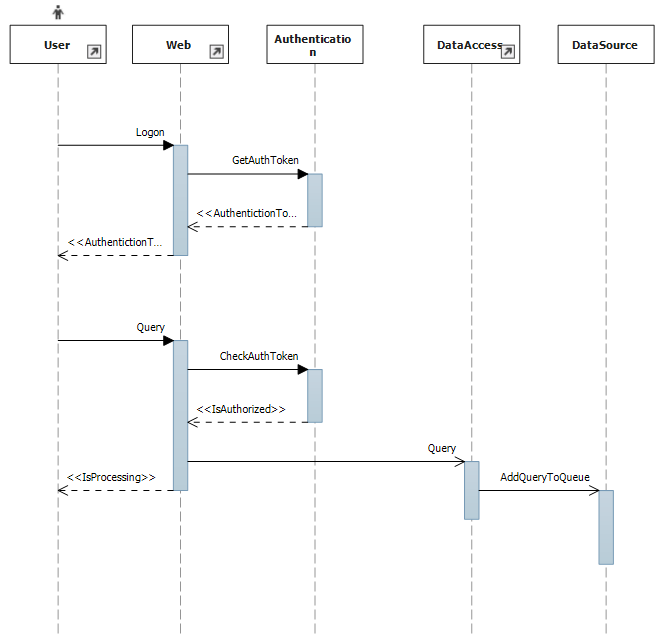


Рисунок 2.4 - Діаграма послідовностей для процесу запиту до системи

Результати виконання будуть бережені системою і при наступному запиті користувачем статусу виконання, його буде повідомлено про закінчення аналізу та надано можливість передивитися результати. Результати будуть взяті з даних і відображені на звичайним запитом MVC до даних та виведенням їх на сторінці.

2.3 Нереляційне сховище даних

В результаті аналізу вимог користувачів до розробляємої системи, було виявлено, що система буде зберігати та обробляти великий об’єм даних. Приблизно десять терабайт після перших трьох місяців використання системи. Іншою вимогою було те, що запит до даних повинен виконуватись у проміжку від нуля до двох хвилин. Виходячи з цих вимог було проаналізовано декілька рішень для Big Data. Серед них Google BigTable, Hadoop, Hive, Spark, HBase, Cassandra, MongoDB.

Hadoop - вільна програмна платформа і каркас для організації розподіленої обробки великих обсягів даних (що міряється у петабайтах) з використанням парадигми MapReduce, при якій завдання ділиться на безліч дрібніших відособлених фрагментів, кожен з яких може бути запущений на окремому вузлі [кластера](https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)). До складу Hadoop входить також реалізація розподіле[ної файлової системи](https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D0%BF%D0%BE%D0%B4%D1%96%D0%BB%D0%B5%D0%BD%D0%B0_%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) Hadoop Distributed Filesystem (HDFS), котра автоматично забезпечує резервування даних і оптимізована для роботи MapReduce-[застосунків](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA). Для спрощення доступу до даних в сховищі Hadoop розроблена [БД](https://uk.wikipedia.org/wiki/%D0%A1%D0%9A%D0%91%D0%94) [HBase](https://uk.wikipedia.org/w/index.php?title=HBase&action=edit&redlink=1) і[SQL](https://uk.wikipedia.org/wiki/SQL)-подібна [мова](https://uk.wikipedia.org/w/index.php?title=%D0%9C%D0%BE%D0%B2%D0%B0_%D0%B7%D0%B0%D0%BF%D0%B8%D1%82%D1%96%D0%B2&action=edit&redlink=1) Hive, яка є свого роду SQL для MapReduce і запити якої можуть бути розпаралелені і оброблені кількома Hadoop-платформами.

Google BigTable - розподілена система керування базами даних, що відноситься до класу [noSQL](https://uk.wikipedia.org/wiki/NoSQL)-систем і розрахована на створення високомасштабованих і надійних сховищ величезних масивів даних, представлених у вигляді хеша.

Apache Spark - високопродуктивний рушій для обробки даних, що зберігаються в [клас](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80)т[ері](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80) Hadoop. У порівнянні з наданим в Hadoop механізмом MapReduce, Spark забезпечує в 100 разів більшу продуктивність при обробці даних у пам'яті і 10 разів при розміщенні даних на дисках. Рушій може виконуватися на вузлах кластера Hadoop як за допомогою Hadoop YARN, так і у відокремленому режимі. Підтримується обробка даних у сховищах HDFS, HBase, Cassandra, Hive і будь-якому форматі введення Hadoop (InputFormat).

MongoDB - документо-орієнтована система керування базами даних (СКБД) з відкритим сирцевим кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

Для задоволення усіх вимог в якості платформи для розгортання була обрана Microsoft Azure з сервісом HDInsight, який реалізований на базі Hadoop. В якості мови роботи з кластерами Hadoop був обраний Hive. В якості моделі для розподілених обчислень був обраний MapReduce (див. рис. 2.5), що є одним з модулів в Hadoop. Суть цього алгоритму в поділі входять даних на рівні за розміром складові, і обробку кожної складової на окремому сервері за заданим алгоритмом. Результати обробки з кожного сервера будуть об'єднані в загальний результат і віддані клієнту.



Рисунок 2.5 - Алгоритм MapReduce

Для побудови статистичної інформації за вхідними даними, на початку на кожному робочому вузлі кластера відбувається виділення метаінформації з кожного окремого зображення або кадру і передачі її у вигляді наборів тегів. На аналізованому зображенні виділяються прості геометричні об'єкти такі, як лінії допомогою алгоритму Хафа. Надалі використовуючи штучні багатошарові нейронні мережі з кожним наступному шаром виконується розпізнавання більш складних графічних об'єктів. Дані отримані в результаті з вихідних шарів нейронних мереж у вигляді текстових тегів будуть передані майстер-вузлу для подальшого статистичного аналізу на етапі згортки. Для швидкого та зручного доступу до даних було вирішено використовувати нереляційну базу даних Apache HBase, що є частиною сімейства продуктів Hadoop, технологію Hive та SQL-подібну мову запитів HiveQL.

2.4 Модуль розпізнавання

Розпізнавання образів у медіа-матеріалах відбувається у модулі FilePreporcessor. Розпізнавання образів — це процес віднесення початкових даних до певного класу за допомогою виділення істотних ознак, що характеризують ці дані, із загальної маси несуттєвих даних.

2.4.1 Методи розпізнавання

Для виявлення класів об'єктів, розташованих на зображеннях, розглянемо існуючі алгоритми розпізнавання; далі згруповані за стилем навчання. Є різні алгоритми для моделювання задачі, засновані на його взаємодії з існуючим досвідом чи навколишнім середовищем або те, що ми хочемо називати вхідними даними. дарі розглянемо існуючі популярні алгоритми з машинним навчанням і штучним інтелектом. Спочатку розглянемо їх за видами навчання, що доступні для алгоритмів.

Є тільки кілька основних стилів навчання або навчальних моделей, що можуть бути в алгоритмів. Розглянемо декілька таких прикладів і проблему, що ними вирішуються.

Кероване навчання. Тут вхідні дані називаються навчальними даними і мають відомий відповідний тег або визначені такі, які відповідають цілі, і такі, що не відповідають. Модель підготовлюється через тренувальний процес, де це потрібно, щоб робити прогнози і коректується, коли ці прогнози невірні. Процес підготовки продовжується до тих пір, поки модель не досягне бажаного рівня точності на навчальних даних.

Приклад проблеми, що вирішується - класифікації і регресії. Приклади алгоритмів включають: логістичну регресії і нейронної мережі зворотнього поширення.

Неконтрольоване навчання. Дані не марковані і не мають відомий результат. Модель підготовлена автоматично виведеними структурами, присутніми у вихідних даних. Результатом може бути, щоб витяг загальних правил. Це можна виконати за допомогою математичного процесу систематично зменшуючи надлишковість, або це може бути організація даних за подібністю.

Приклад проблеми, що вирішується - кластеризації, зниження розмірності і правила асоціації навчання. Приклади алгоритми включають в себе: алгоритм Apriori і k-Means.

Полу-контрольоване навчання. Вхідні дані являють собою суміш маркованих і немаркованих прикладів. Існує проблема передбачення бажаних значень, але модель повинна дізнатися структур для організації даних, а також робити прогнози.

Приклад проблеми, що вирішується - класифікації і регресії. Прикладами алгоритми є розширеннями інших гнучких методів, які роблять припущення про те, як моделювати немарковані дані.

Коли розглядаються дані для моделювання бізнес-рішень, як правило, проблема вирішується за допомогою контрольованих і неконтрольованих методи навчання. Популярною темою в даний момент є напів-контрольовані методи навчання в таких областях, як класифікація зображень, де є великі набори даних з дуже небагатьма міченими прикладами.

Групування алгоритмів за схожістю. Алгоритми часто групуються за подібністю з точки зору їх функції, тим як вони працюють. Наприклад, на основі дерев і метода подібні до нейронних мереж. Це корисний метод групування, але він не є досконалим. Є ще алгоритми, які могли б так само легко вписуються в кілька категорій, таких як LVQ, який є одночасно нейронною мережею і метод на основі примірників. Є також категорії, які мають те ж ім'я, що описує проблему і клас алгоритму, таких як регресія і кластеризації. Ми могли б впоратися з цими випадках, перерахувавши алгоритми двічі або вибравши групу, яка суб'єктивно "краще" потрібним. Існує сильний ухил у бік алгоритмів, що використовуються для класифікації та регресії, два найпоширеніших контрольованих проблеми машинного навчання ви зустрінете.

Регресійні алгоритми. Стосується моделювання взаємозв'язку між змінними, які ітеративно очищено з використанням міру похибки в прогнозах, зроблених моделі. Методи регресії - це робоча конячка статистики і були виведені в статистичні машинне навчання. Це може ввести в оману, тому що ми можемо використовувати регресію для посилання до класу проблеми і класу алгоритму. Регресія це процес.

Найпопулярніші алгоритми регресії:

- звичайна регресія найменших квадратів;

- лінійна регресія;

- логістична регресія;

- покрокова регресія;

- багатомірна адаптивна сплавнова регресія;

- локальні розрахунки розкиду точок згладжування.

На основі шаблонів. Моделі навчання вирішення проблеми з шаблонами або прикладах навчальних даних, які вважаються важливим або необхідних для моделі. Такі методи, як правило, побудовують базу даних, наприклад, дані і порівняння нових даних в базі даних, використовуючи міру подібності для того, щоб знайти краще уподібнення і зробити прогноз. З цієї причини, методи шаблонів також називають "переможець отримує все" методи і навчання на основі запам'ятовування. Основна увага приділяється поданням збережених примірників і подібність структур, що застосовуються між екземплярами.

Найпопулярнішими алгоритмами на основі шаблонів є:

- k-Nearest Neighbour;

- Learning Vector Quantization (LVQ);

- самоорганізаційні карти

- локально-вагове навчання.

Регуляризаційні алгоритми. Як розширення іншого методу (зазвичай методи регресії), що передбачає моделі покарання, засновані на їх складності, на користь більш простих моделей, які також кращі в узагальненні. Алгоритми регуляризації дуже популярні, потужний і в цілому є простими змінами, внесеними в інших методах.

Найпопулярніші алгоритми регуляризації це:

- регуляризація Тихонова;

- Least Absolute Shrinkage and Selection Operato (Lasso method);

- еластична сітка;

- Least-Angle Regression (LARS).

Дерева прийняття рішень. Методи дерево будують модель рішень, прийнятих на основі фактичних значень атрибутів в даних. Вилка рішень в деревовидних структур до прийняття рішення прогнози зроблена для наданого запису. Дерева прийняття рішень навчаються на даних щодо класифікації та регресії проблем. Дерева прийняття рішень часто швидкий і точний і великий фаворит в машинному навчанні.

Найпопулярнішими алгоритмами дерев рішень є:

- класифікація і регресіїні дерева;

- Iterative Dichotomiser 3 (ID3) ;

- C4.5 і C5.0 (різні версії цих потужних підходів);

- Чи-квадрат Автоматична Взаємодія виявлення (CHAID);

- M5;

- Умовні Дерева прийняття рішення

Байєсовські алгоритми. Методи, які явно застосувати теорему Байєса для таких проблем, як класифікації і регресії.

Найпопулярніші байесовских алгоритми:

- наївний байесовский;

- гаусовий наївний байесовий;

- поліноміальний наївний байесовий;

- усереднені одно залежні оцінки;

- байесова довірена мережа;

- байесова мережа.

Алгоритми кластеризації. Кластеризація, як регресія, описує клас проблем і клас методів. Методи кластеризації, як правило, організовані підходів моделювання, таких як центр ваги на основі ієрархічних і. Всі методи пов'язані з використанням властиві структури в даних, щоб найкращим чином організувати дані в групи максимальної спільності.

Найпопулярніші алгоритми кластеризації:

- k-Means;

- k-Medians;

- Expectation Maximisation (EM) ;

- Hierarchical Clustering.

Навчання за асоціаційними правилами. Методи асоціаційні правила навчання, які витягують правила, які найкраще пояснюють спостережувані відносини між змінними в даних. Ці правила можуть виявити важливі і комерційно корисні асоціації у великих багатовимірних масивів даних, які можуть бути експлуатованих організацією.

Найпопулярніші алгоритми правило асоціація навчання є:

- Apriori algorithm;

- Eclat algorithm.

Штучні нейронні мережі. Штучні нейронні мережі - це моделі, які, натхненні структури та функції біологічних нейронних мереж. Вони являють собою клас пошуку за шаблоном, які зазвичай використовуються для регресії та класифікації проблем, але насправді величезна підполі, що складається з сотень алгоритмів і варіацій для всіх типів чином проблема. Відділений від Deep Learning нейронних мереж через масивного росту і популярності в цій галузі. Тут ми маємо справу з більш класичними методами.

Найпопулярніші штучні алгоритми нейронних мереж є:

- перцептрон;

- зворотного поширення;

- Хопфилда Мережа;

- Radial Basis Function Network.

Алгоритми глибоке навчання є сучасним оновленням штучних нейронних мереж, які використовують багатий дешеві обчислень. Вони пов'язані з будівлі набагато більш великі і складні нейронні мережі, і, як зазначив вище, багато методи пов'язані з підлозі під наглядом проблем навчання, де великі набори даних містять дуже мало мічених даних.

Найпопулярніші глибокі алгоритми навчання є:

- Deep BoltzmШНМ Machine (DBM);

- Deep Belief Networks (DBN);

- Convolutional Neural Network (згортальна нейронна мережа);

- Stacked Auto-Encoders.

Скорочення розмірностей. Зниження розмірності пошуку і використовувати властиву структуру даних, але в цьому випадку в неконтрольованій манері або для того, щоб узагальнити або описати дані, використовуючи менше інформації. Це може бути корисно для візуалізації двомірних даних або для спрощення дані, які потім можуть бути використані в способі обучени supervized. Багато з цих методів може бути пристосований для використання в класифікації і регресії:

- Principal Component Analysis (PCA);

- Principal Component Regression (PCR);

- Partial Least Squares Regression (PLSR);

- Sammon Mapping;

- Multidimensional Scaling (MDS);

- Projection Pursuit;

- Linear Discriminant Analysis (LDA);

- Mixture Discriminant Analysis (MDA);

- Quadratic Discriminant Analysis (QDA);

- Flexible Discriminant Analysis (FDA).

Ансамбльові Алгоритми. Моделі, що складаються з кількох слабших моделей, які незалежно один від одного, навчених і чиї прогнози об'єднані в якийсь спосіб зробити загальну прогноз. Багато зусиль поміщається в які типи слабких учнів поєднувати і способів, в яких, щоб об'єднати їх. Це дуже потужний клас методів і, таким чином, дуже популярні.

- Boosting;

- Bootstrapped Aggregation (Bagging);

- AdaBoost;

- Stacked Generalization (blending);

- Gradient Boosting Machines (GBM);

- Gradient Boosted Regression Trees (GBRT);

- Random Forest.

Інші алгоритми. Багато алгоритмів не були розподілено в групи.

- алгоритми вибору функцій;

- оцінка точності алгоритму;

- показники ефективності;

- обчислювальний інтелект (еволюційні алгоритми, і т.д.);

- обробки природної мови;

- рекомендаційні системи;

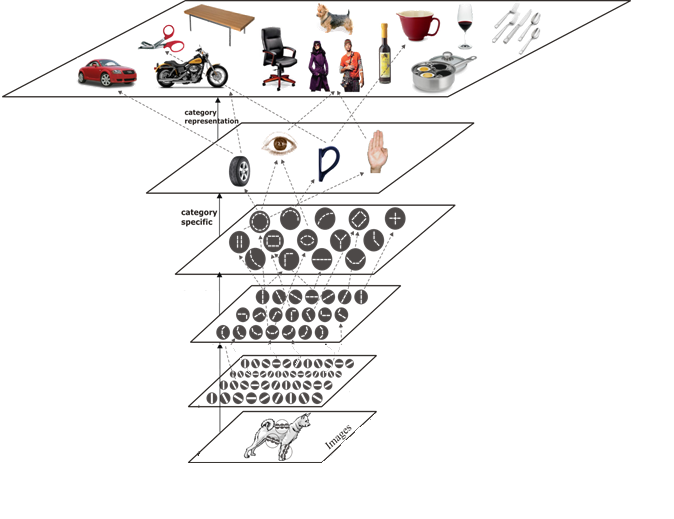
- підкріплене навчання;

- графічні моделі.

Для розпізнавання образів може бути застосовано декілька методів, зокрема, метод перебору вигляду об'єкта під різними кутами, масштабами, зсувами і т. д. Наступним методом є пошук контуру об'єкта й дослідження його властивості (зв’язність, наявність кутів і т. д.). Останнім методом є використання штучних нейронних мереж. Це вимагає створення, відповідно до специфіки задачі, нейронної мережі спеціальної структури, або великої кількості прикладів задачі розпізнавання з визначеними правильними виходами мережі.

2.4.2 Аналіз реальних об'єктів і біологічні нейронні мережі

Усі об’єкти світу можуть бути розбиті на гігантські ієрархії їх складових сегментів (див. рис. 2.6) . І ці самі об'єкти аналогічно є частинам інших більших ієрархії, створюючи таким чином структуру всього всесвіту. При необхідності знаходити та визначати ці обєкти в компютерах на проекціях реального світу - фотографіях та відео, треба звернутися до систем, що вже досконало це роблять. Цими системами є живі організми, зокрема їх найвищі представники - ссавці. Їх нервова система та мозок відповідають за пристосування та аналіз навколишнього світу. Головну роботу з розпізнавання та навчання виконує неокортекс - частина шару сірої речовини на поверхні півкуль головного мозку.

Рисунок 2.6 - Ієрархічна структури обєктів

Кору мозку складають мільйони нейронів. Кожен нейрон мозку являє собою клітину із двома типами відростків: дендрити та аксони (див. рис. 2.7). Через дендрити клітина отримує сигнал від іншої клітини. Сигнали отримані дендритами клітиною сумарно обробляються та при перевищені деякого загального рівня сигналу у сомі клітини створює свій сигнал і передає через аксон його дендритам інших клітин. Нейрони утворюються між собою безліч зв'язків для обробки усього потоку інформації. Якщо зв'язані клітини активуються одночасно, їх зв'язок посилюється полегшуючи їх подальшу взаємну активацію та забезпечуючи навчання мережі нейронів реакції на певний патерн вхідних сигналів.

Безпосередньо аналізом зображень отриманих очами займається зорова кора. Початково гангліозні клітини з біполярними виділяють межі та прості лінії та рухи і отриманого сітківкою зображення. Далі сигнал передаються до зорової кори для виявлення обєктів зовнішнього світу. Для цього її структура ієрархічно відповідає структурі оточуючих обєктів. Саме тим, що вона розбита на малі структури - колонки. Колонки формують між собою ієрархічні структури, що відповідають за різні рівні обєктів - від простих, як лінії та кути, до складних, як люди. Цією структурою забезпечується інваріантність розпізнання. Дозволяє виявити обєкти незалежно від зміни його масштабу, положення та куту перегляду.

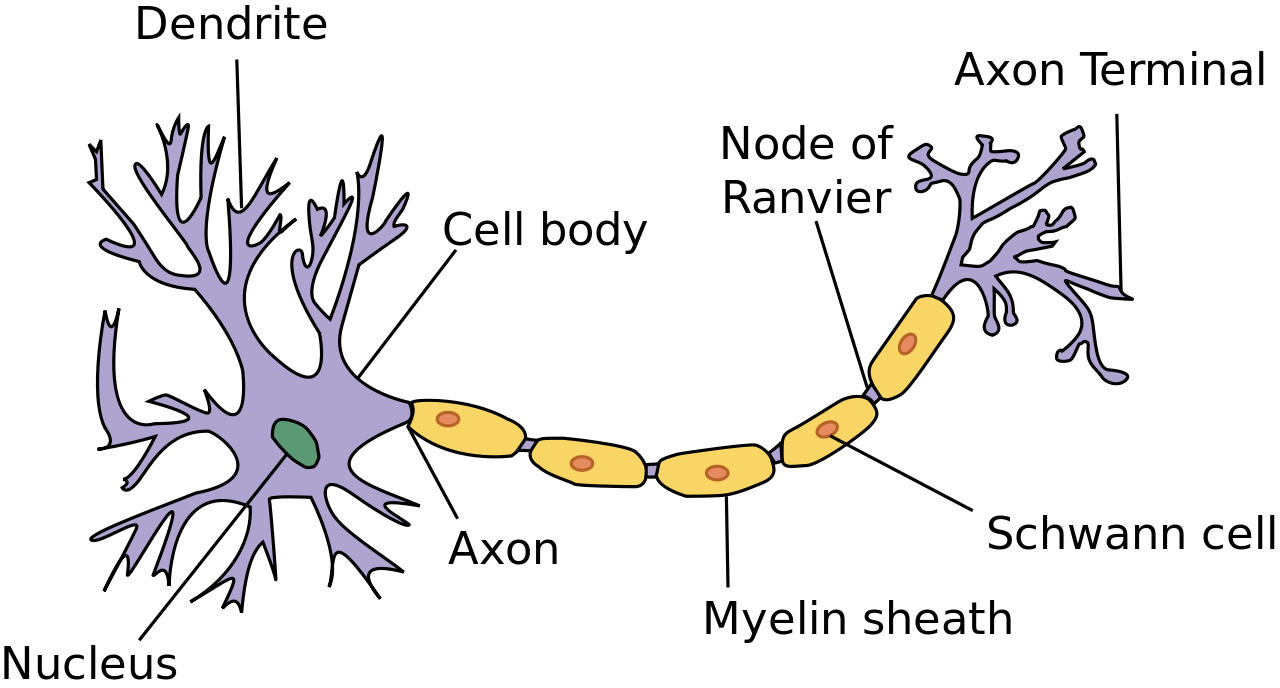


Рисунок 2.7 - Біологічний нейрон

Штучні нейроні мережі моделюють процеси, приближені до тих, що відбуваються у мозку тварин, зокрема процес аналізу сигналів зорових аналізаторів.

Першими, кто формалізували та зробили спробу розробити штучну нейрону мережу, були У. Маккалок та У. Піттс у 1943. Наступним кроком була винайдення Ф. Розенблатом у 1958 році одношарового персептрону для виконання задач класифікації та прогнозування і нейронного компьютеру «Марк-1» у 1960 році, що міг розпізнавати літери англійського алфавіту.

Простий нейрон представляє з функцію, що видає на декілька вхідних сигналів власний відповідний сигнал. Вона складається з сумуючої та активаційної (див. рис. 2.8) функції.

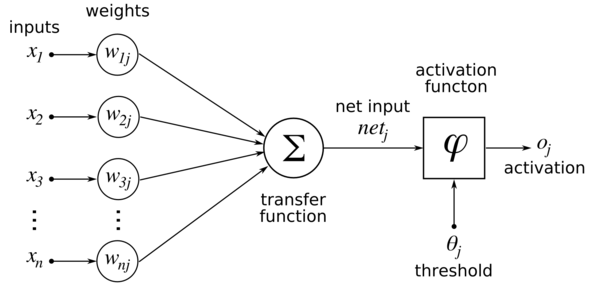


Рисунок 2.8 - Структура штучного нейрону

Сумарна функція (2.1) помножує кожен вхідний Xi сигнал нейрона на відповідний йому ваговий коефіцієнт Wi і сумує їх, формуючи спільний вхідний сигнал net, аналогічно біологічній сомі. У ході навчання нейрона значення вагового коефіцієнта змінюються.

|  |  |
| --- | --- |
| https://lh5.googleusercontent.com/OvFkBCc3Wqlh7YsDTkVRz8JRrwUwIg3JsAy2oKZGY4RrdU_4Xcu-jqiQo3WxLWL6P7Lzj6pYx7Some0ReOKB_jFrpZalV6ndJi-QLAsPHkBY-td9QIAQkIQzBynsUGrbMqvRVdXd | (2.1) |

Далі функція активації (2.2) в залежності від спільного сигналу net дає вихідний сигнал нейрону Out. У найпростішому варіанті, для порогової функції (див. рис 2.9), ця функції видає вихідний сигнал 1, якщо сумарний сигнал перевищує деяке визначене константне порогове значення θ. Інакше вона повератає значення 0.

|  |  |
| --- | --- |
| Формула функции активации | (2.2) |

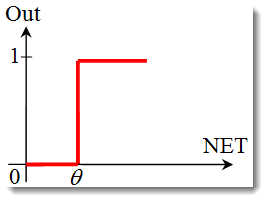


Рисунок 2.9 - Жорстко порогова функція

Також застосовуєтсья часто застосовується у многошарових мережах та мережах із безперервним сигналом сигмоідальна функція (див. рис. 2.10), через такі якості як безперервність та плавкість. І гіперболічний тангенс (див. рис. 2.11), через його можливість повертати відємні значення.

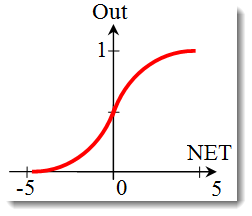


Рисунок 2.10 - Сигмоідальна функція

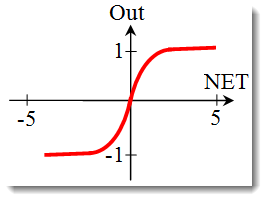


Рисунок 2.11 - Гіперболічний тангенс

Один нейрон не зможе виконати весь спектр задач розпізнавання та прогнозування. Тому створюються структури - мережі із багатої кількості нейронів. Найпростішою нейронною мережею є персептрон (див. рис. 2.12). Елементарний персептрон складається із слою сенсорних вхідних нейронів-S, слою прихованих ассоціційних А-нейронів та вихідного слою із R-нейрона.

Сенсорні нейрони (аналогами котрих є клітини сітківки тварин) збуджуються, якщо він освітлюється (відповідає світлій точці зображення) і передаються цей сигнал наступному шару. Далі асоціативні нейрони приймають сигнали від декількох асоційованих їм сенсорних, сумують їх виходи та при перевищені кількості активованих сенсорних нейронів деякого порогу, передають на виході сигнал. Таким чином асоціативні нейрони можуть активуватися та визначати наявність, наприклад, частини деякої цифри у відповідній їм частині зображення. Виходи асоціативних нейронів передаються R-нейрону. Він сумує сигнали враховуючи вагові коефіцієнти і обробляє пороговою функцією. І тоді його вихідне значення буде визначати наявність на зображенні обєкту певного класу. При навчанні, змінюються ваги звязків A та R нейронів, звязки S-A визначаються на початку і далі залишаються незмінними.

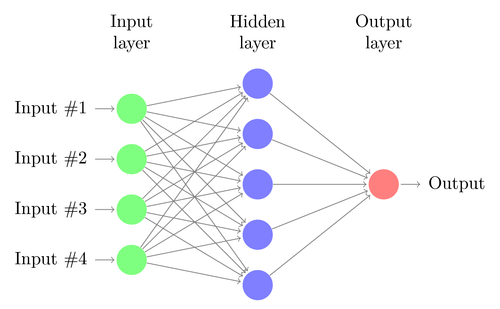


Рисунок 2.12 - Персептрон

Існують і більш складні структури нейронних мереж. Деякі нейрони пов'язані з "зовнішнім світом", деякі тільки з іншими нейронами (приховані нейрони). Існує два основних типи топології мереж: feed-forward (з прямим зв'язком) і feedback (зі зворотним зв'язком). У feed-forward мережі сигнал поширюється тільки в одну сторону, тобто мережа не має петель. У feedback-нейромережі вихідний сигнал від нейрона передається на вхід іншого нейрона, на тому ж або попередньому рівні.

У feed-forward нейронних мережах сигнал поширюється тільки в одному напрямку - від входів до виходів. Лінійні feed-forward нейронні мережі були створені одними з перших (персептрон). В даний час найчастіше використовують нелінійні feed-forward мережі. Feed-forward мережі вважаються менш придатними для реалізації асоціативної пам'яті, ніж feedback мережі, хоча вони можуть виконувати такі ж завдання.

В даний час нейронні мережі з архітектурою feed-forward працюють краще, ніж мережі з архітектурою feedback з кількох причин. Ємність feedback виявилася не дуже великою. Так, наприклад кількість даних, яка може бути збережена без помилок у сережі Хопфілда обчислюється за формулою: M <= N / (2 \* log2 N), де N - кількість нейронів. Це означає, що для зберігання 1000 записів потрібно 30000 нейронів і 300000000 унікальних зв'язків. Для порівняння, NetTalk додаток використовує feed-forward модель, використовуючи 300 нейронів і 20,000 унікальних сполук, може вивчити правильне написання 1000 слів і близько 7000 асоціацій буква-звук.

Крім того, feed-forward мережі працюють швидше, ніж feedback, тому для того щоб знайти рішення їм потрібно зробити тільки один прохід. Feedback мережі повинні повторювати цикл знову і знову, до тих пір, поки виходи не перестануть змінюватися. Зазвичай для цього потрібно від 3 до 1000 циклів. Feed-forward нейронні мережі можуть бути контрольовані (supervised) або неконтрольовані (unsupervised). Контрольовані нейронні мережі під час навчання порівнюють свій вихід із заздалегідь відомим, в той час як неконтрольовані не роблять цього.

Мережами прямого поширення є, наприклад, найпростіший персептрон (розроблений Розенблаттом) і багатошаровий персептрон.

Рекурентні нейронні мережі - сигнал з вихідних нейронів або нейронів прихованого шару частково передається назад на входи нейронів вхідного шару.

Радіально базисні функції - вид нейронної мережі, що має прихований шар з радіальних елементів і вихідний шар з лінійних елементів. Мережі цього типу досить компактні і швидко навчаються. Запропоновано в роботах Broomhead and Lowe (1988) і Moody and Darkin (1989). Радіально базисна мережа володіє наступними особливостями: один прихований шар, тільки нейрони прихованого шару мають нелінійну активаційну функцію і синаптичні ваги вхідного і прихованого шарів дорівнюють одиниці.

Самоорганізаційні карти або мережі Кохонена - такий клас мереж, як правило, навчається без вчителя і успішно застосовується в задачах розпізнавання. Мережі такого класу здатні виявляти новизну у вхідних даних: якщо після навчання мережа зустрінеться з набором даних, несхожим ні на один з відомих зразків, то вона не зможе класифікувати такий набір і тим самим виявить його новизну. Мережа Кохонена має всього два прошарки: вхідний і вихідний, складений з радіальних елементів. Для роботи штучної нейронної мережі вона спочатку має бути навчена своїй задачі. Нейронні мережі відрізняються за способом їх навчання:

- нейронні мережі, що використовують навчання з учителем;

- нейронні мережі, що використовують навчання без учителя.

Навчання з учителем припускає, що для кожного вхідного вектора існує цільовий вектор, що представляє собою необхідний вихід. Разом вони називаються навчальною парою. Зазвичай мережа навчається на деякому числі таких навчальних пар. Пред'являється вихідний вектор, обчислюється вихід мережі і порівнюється з відповідним цільовим вектором. Далі ваги змінюються відповідно до алгоритму, що прагнуть мінімізувати помилку. Вектори навчальної множини пред'являються послідовно, обчислюються помилки і ваги підлаштовуються для кожного вектора доти, доки помилка по всьому навчальному масиву не досягне прийнятного рівня.

Нейронні мережі, що використовують навчання без учителя. Навчання без вчителя є набагато більш правдоподібною моделлю навчання з погляду біологічних коренів штучних нейронних мереж. Розвинена Кохоненом і багатьма іншими, вона не потребує цільового векторі для виходів і, отже, не вимагає порівняння з зумовленими ідеальними відповідями. Навчальна множина складається лише з вхідних векторів. Навчальний алгоритм підлаштовує ваги мережі так, щоб виходили узгоджені вихідні вектори, тобто щоб пред'ява досить близьких вхідних векторів давала однакові виходи. Процес навчання, таким чином, виділяє статистичні властивості навчальної множини і групує подібні вектори в класи.

Використання класичних нейронних мереж для розпізнавання зображень ускладнене, як правило, великою розмірністю вектора вхідних значень нейронної мережі, великою кількістю нейронів у проміжних шарах і, як наслідок, великими витратами обчислювальних ресурсів на навчання і обчислення мережі. Згортальні нейронних мереж в меншій мірі притаманні описані вище недоліки.

2.4.3 Згортальна нейронна мережа

Згортальна нейронна мережа (англ. Convolutional neural network, CNN) - спеціальна архітектура штучних нейронних мереж, запропонована Яном Лекуном і націлена на ефективне розпізнавання зображень, входить до складу технологій глибокого навчання (англ. Deep leaning). Ця технологія побудована за аналогією з принципами роботи зорової кори головного мозку, в якій були відкриті так звані прості клітини, що реагують на прямі лінії під різними кутами, і складні клітини, реакція яких пов'язана з активацією певного набору простих клітин.

Таким чином, ідея згортальних нейронних мереж полягає в чергуванні згортальні них шарів (англ. Convolution layers) і субдескритезуючих шарів (англ. Subsampling layers, шарів підвибірки) (див. рис. 2.13).

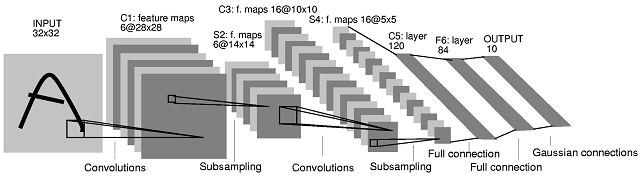


Рисунок 2.13 - Архітектура згортальної нейронної мережі

Ключовим моментом у розумінні згортальних нейронних мереж є поняття так званих «поділюваних» ваг, тобто частина нейронів деякого розглянутого шару нейронної мережі може використовувати одні й ті ж вагові коефіцієнти. Нейрони, що використовують одні й ті ж ваги, об'єднуються в карти ознак (feature maps), а кожен нейрон карти ознак пов'язаний з частиною нейронів попереднього шару. При обчисленні мережі виходить, що кожен нейрон виконує згортку (операцію конволюції) деякої області попереднього шару (обумовленої безліччю нейронів, пов'язаних з даним нейроном). Шари нейронної мережі, побудовані описаним чином, називаються згортальне шарами. Крім, згортальних шарів у згортальної нейронної мережі можуть бути шари субдескритезації (виконують функції зменшення розмірності простору карт ознак) і повнозв'язні шари (вихідний шар, як правило, завжди повнозв'язний). Усі три види шарів можуть чергуватися в довільному порядку, що дозволяє складати карти ознак з карт ознак, а це на практиці означає можливість розпізнавання складних ієрархій ознак.

2.4.4 Похибка штучної нейронної мережі

Градієнт Спуск, будучи безперервною функцією, один з найбільших переваг лінійною функцією активації через блок ступінчастої функції є те, що вона диференційовна. Ця властивість дозволяє визначити функцію вартості J (W) (2.3), що ми можемо мінімізувати для того, щоб оновити наші ваг. У разі лінійної функції активації, ми можемо визначити функцію вартості J (W) у вигляді суми квадратів помилок (SSE), яка схожа на функцію витрат, що мінімізується в звичайних найменших квадратів (МНК) лінійна регресія.

|  |  |
| --- | --- |
|  | (2.3) |

Для того, щоб мінімізувати функцію вартості SSE, ми будемо використовувати градієнтний спуск, простий, але корисний алгоритм оптимізації, який часто використовується в машинному навчанні, щоб знайти локальний мінімум лінійних систем. Розглянемо опуклу функцію витрат однієї ваги. Принцип її роботи у тому, що вона лежить у градієнтному спуску в "не злазити з пагорба", поки локальний або глобальний мінімум досягається. На кожному кроці, ми зробити крок у напрямку, протилежному напрямку градієнта, і розмір кроку визначається значенням швидкості навчання (див. рис. 2.14), а також нахил градієнта. Виведенні навчання правила Adaline - як уже згадувалося вище, кожне оновлення оновлюється, приймаючи крок у протилежному напрямку градієнта Δw = -η∇J (w), таким чином, ми повинні обчислити приватну похідну від вартості функція для кожної ваги у ваговій вектор: Δwj = -η. Приватної похідною функції витрат ГСП для конкретного ваги можна розрахувати наступним чином:

|  |  |
| --- | --- |
| Δ = - η= − η(t(i) − o(i)) ( −)= η(t(i) − o(i)) () | (2.4) |

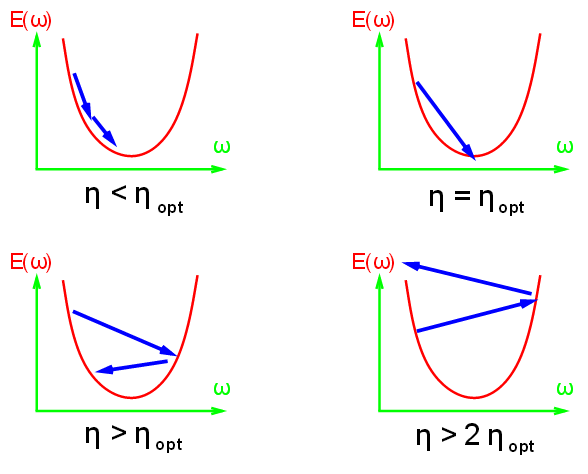
(t = мета, о = вихід)

І якщо ми підключити результати назад в навчанні правило, ми отримуємо:

|  |  |
| --- | --- |
| Δ = - η= − η(t(i) − o(i)) ( −)= η(t(i) − o(i)) () | (2.5) |

Зрештою, ми можемо застосувати одночасно поновлення ваги, схожий на правила персептрона: W: = W + Δw. Хоча, вище правило навчання виглядає ідентично правилом персептрона, відзначимо дві основні відмінності: Тут вихід "O" є дійсним числом, а не клас етикетки, як правило в персептрон навчання. Оновлення ваги розраховується на основі всіх зразків у повчальному наборі (замість оновлення ваг поступово після кожного зразка), тому цей підхід називають також "пакетної" градієнтного спуску.

Для навчання нейронної мережі можна використовувати метод навчання за допомогою стохастичного градієнтного спуску. Функція вартості мінімізується на основі повного набору даних навчання. Якщо ми згадаємо правилом персептрона, ми пам'ятаємо, що вона виконувала оновлення ваги поступово після кожного зразка навчання. Цей підхід також називають "онлайн" навчання, а насправді, це також як Adaline був вперше описаний Бернард Уїдроу ін. Процес додаткового поновлення ваг також називається "стохастичним" градієнтного спуску, оскільки вона наближається до мінімізації функції вартості.

Рисунок 2.14 - Коефіцієнт похибки

Хоча стохастичний градієнтний спуск підхід може здатися гірше градієнтного спуску належне його "стохастичного" природи і "приблизною" напрямку (градієнт), вона може мати певні переваги на практиці. Часто, стохастичний градієнтний спуск сходиться набагато швидше, ніж градієнтного спуску, оскільки поновлення застосовуються відразу після кожного тренувального зразка; стохастичний градієнтний спуск в обчислювальному більш ефективним, особливо для дуже великих наборів даних. Ще одна перевага онлайн-навчання є те, що класифікатор може бути негайно оновлюється надходження нових даних навчання, наприклад, у веб-додатках, і старі дані навчання можуть бути відкинуті, якщо пам'ять є проблемою. У великомасштабних систем машинного навчання, це також звичайна практика, щоб використовувати так звані "міні-партій», компроміс з гладкою збіжності, ніж стохастичною градієнтного спуску. В інтересах повноти давайте також здійснювати стохастичний градієнтний спуск Adaline і переконайтеся, що він сходиться на лінійно поділеній діафрагмі даних. Перш ніж дозволити Adaline дізнатися за допомогою стохастичного градієнтного спуску, є перетасування підготовки набору даних для перебору навчальних вибірок у випадковому порядку.

Відзначимо, що "стандартний" алгоритм стохастичного градієнтного спуску використовує вибірку "з заміною", що означає, що на кожній ітерації, навчальний зразок вибирається випадковим чином з усього набору навчання. На противагу цьому, відбір проб "без заміни", що означає, що кожен зразок навчання оцінюється тільки один раз в кожному епохи, не тільки легше реалізувати, але також показує більш високу продуктивність у порівнянні емпіричних.

|  |  |
| --- | --- |
|  | (2.6) |

Більш детальне обговорення на цю тему можна знайти в Бенджамін Recht і паперу Крістофера Ре Під долині некомутативної арифметичного середнє геометричне нерівність: домисли, тематичні дослідження, і наслідки.

2.4.5 Навчання штучної нейронної мережі

Многошарові персептрони застосовуються для рішення різноманітного кола задач, навчання такої нейронної мережі часто виконується за допомогою такого популярного алгоритму як алгоритм зворотного поширення помилки. Алгоритм заснований на корекції помилок нетренованої мережі. Навчання методом зворотного поширення похибки припускає два проходи по всім верствам мережі: прямий і зворотний. При прямому проході вхідний вектор з навчального набору подається на входи мережі і сигнал поширюється в прямому напрямку , що призводить до генерації на виході мережі якогось то вихідного вектора. При зворотньому ж проході вихідний вектор віднімається з бажаного і наявного у навчальному наборі, у результаті чого формується сигнал помилки, який поширюється в напрямку зворотньому напрямку синаптичних зв'язків. При прямому проході синаптичні ваги фіксовані, а при зворотньому здійснюється корекція синаптичних ваг відповідно з яким-небудь правилом корекції. Синаптичні ваги налаштовуються з метою максимального наближення фактичного виходу мережі до бажаного в статистичному сенсі.

Метод градієнтного спуску. Розглянемо функцію f, вважаючи для визначеності, що вона залежить від трьох змінних x, y, z. Обчислимо її приватні похідні f/х, f/у, f/z і утворюємо з їх допомогою вектор, який називають градієнтом функції:

|  |  |
| --- | --- |
|  | (2.7) |

Тут i, j, k - одиничні вектори, паралельні координатним осям. Приватні похідні характеризують зміна функції f по кожній незалежної змінної окремо. Утворений з їх допомогою вектор градієнта дає загальне уявлення про поведінку функції в околі точки (х, у, z). Напрямок цього вектора є напрямом найбільш швидкого зростання функції в даній точці. Протилежне йому напрям, який часто називають антіградіентним, являє собою напрямок найбільш швидкого спадання функції.

|  |  |
| --- | --- |
|  | ((2.8) |

Модуль градієнта визначає швидкість зростання та спадання функції в напрямку градієнта і антіградіента. Для всіх інших напрямків швидкість зміни функції в точці (х, у, z) менше модуля градієнта. При переході від однієї точки до іншої як напрям градієнта, так і його модуль, взагалі кажучи, міняються. Поняття градієнта природним чином переноситься на функції будь-якого числа змінних [15].

Перейдемо до опису методу градієнтного спуску. Основна його ідея полягає в тому, щоб рухатися до мінімуму в напрямку найбільш швидкого спадання функції, яке визначається антіградіентом. Ця ідея реалізується наступним чином.

Виберемо яким-небудь способом початкову точку, обчислимо в ній градієнт аналізованої функції і зробимо невеликий крок у зворотному, антіградіентном напрямку. У результаті ми прийдемо в точку, в якій значення функції буде менше первинного. У новій точці повторимо процедуру: знову обчислимо градієнт функції і зробимо крок у зворотному напрямку. Продовжуючи цей процес, ми будемо рухатися в бік спадання функції. Спеціальний вибір напрямку руху на кожному кроці дозволяє сподіватися на те, що в даному випадку наближення до найменшому значенню функції буде більш швидким, ніж у методі покоордінатного спуску.

Метод градієнтного спуску (див. рис. 2.15) вимагає обчислення градієнта цільової функції на кожному кроці [16]. Якщо вона задана аналітично, то це, як правило, не проблема: для приватних похідних, що визначають градієнт, можна отримати явні формули. В іншому випадку приватні похідні в потрібних точках доводиться обчислювати наближено, замінюючи їх відповідними різницевими відносинами. На рис. 2.15 зображені лінії рівня тієї ж функції двох змінних u = f (х, у).

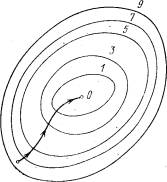


Рисунок 2.15 - Пошук найменшого значення функції методом градієнтного спуску.

Алгоритм Левенберга-Марквардта призначений для оптимізації параметрів нелінійних регресійних моделей. Передбачається, що в якості критерію оптимізації використовується середньоквадратична помилка моделі на навчальній вибірці. Алгоритм полягає в послідовному наближенні заданих початкових значень параметрів до шуканого локального оптимуму.

Задана навчальна вибірка - безліч пар вільної змінної http://neuronus.com/images/theory/algo_L-M/0.jpg (входи мережі) і залежною змінною http://neuronus.com/images/theory/algo_L-M/1.jpg. Задана функціональна залежність, що представляє собою регресійну модель http://neuronus.com/images/theory/algo_L-M/2.jpg, що безперервно диференційованоо в області W · X. Параметр w является вектором вагових коефіцієнтів. Потрібно знайти таке значення вектора w, яке б приносило локальний мінімум функції помилки.

|  |  |
| --- | --- |
|  | (2.9) |

Перед початком роботи алгоритму задається початковий вектор вагових коефіцієнтів w.

На кожному кроці ітерації цей вектор http://neuronus.com/images/theory/algo_L-M/4.jpg замінюється на векторhttp://neuronus.com/images/theory/algo_L-M/5.jpg.

Для оцінки приросту http://neuronus.com/images/theory/algo_L-M/6.jpg використовується лінійне наближення функції http://neuronus.com/images/theory/algo_L-M/7.jpg де J - якобіан функції f (w, xn) у точці w.

Матрицю J наочно можна представити у вигляді

|  |  |
| --- | --- |
|  | (2.10) |

Тут вектор вагових коефіцієнтів http://neuronus.com/images/theory/algo_L-M/9.jpg.

Приріст http://neuronus.com/images/theory/algo_L-M/6.jpg в точці w, що доставляє мінімум Ed равно нулю. Тому для знаходження подальшого збільшення http://neuronus.com/images/theory/algo_L-M/6.jpg прирівнується нулю вектор приватних похідних Ed по w.

|  |  |
| --- | --- |
| *,* | (2.11) |

Таким чином, щоб знайти значення потрібно вирішити систему лінійних рівнянь. Так як число обумовленості матриці JTJ є квадрат числа обумовленості матриці J, то матриця JTJ може виявитися істотно виродженої. Тому Марквардта введений параметр регуляризації.

|  |  |
| --- | --- |
| де I - одинична матриця. | (2.12) |

Цей параметр призначається на кожній ітерації алгоритму. Якщо значення помилки ED убуває швидко, мале значення http://neuronus.com/images/theory/algo_L-M/18.jpg зводить цей алгоритм до алгоритму Гаусса-Ньютона.

Алгоритм зупиняється в тому випадку, якщо приріст http://neuronus.com/images/theory/algo_L-M/6.jpg в подальшій ітерації менше заданого значення, або якщо вектор вагових коефіцієнтів доставляє помилку Ed, меншу заданої величини, або якщо вичерпано число циклів навчання НС. Значення вектора w на останній ітерації вважається шуканим.

Недолік алгоритму - значне зменшення швидкості апроксимації при збільшенні параметра http://neuronus.com/images/theory/algo_L-M/18.jpg.

2.4.6 Доопрацювання

Дослідження, проведені в неврології знайшли два шляхи в система розпізнавання, черевний ("те, що") шлях і Спинний ("де") шлях [13]. На вентральній шлях займає крупний частина сигналів з P клітин в сітківці через оператори Р каналу в LGN, і проходить через коркових областей V1, V2, V4, PIT, CIT, AIT.. Спинний шлях займає крупний частина сигналів від М-клітин в сітківці через оператори М каналу в LGN і проходить через коркових областей V1, V2, MT, LIP, MST, VIP і далі. Це було запропонував (наприклад, Кандель та ін.), що черевний шлях є в основному відповідає за форму і колір об'єктів і Спинний шлях в основному обробляє інформацію про рух і позиції. Внутрішній зв'язок "де" і "що" шляху показує, що увагу і визнання відбувається одночасно, Відомо, як проблема курки і яйця. Без уваги, визнання не може робити добре: розпізнавання вимагає участь AR- EAS для подальшої обробки.

Без визнання, уваги обмежена: увагу не тільки потрібно знизу вгору saliency- засновані київ, але також зверху вниз об'єктно-орієнтована і позиційно засновані сигнали. Знизу-вверх увагу, також званий помітності на основі уваги, використовує різні властивості сенсорних входів, наприклад, кольору, форми, Zhengping Джі з Центром по нейронної основі пізнання, Університет Карнегі-Меллона, Піттсбург, Пенсільванія 15213 США і Juyang Вен є з кафедрою обчислювальної науки і техніки, штат Мічиган Державний університет, Іст-Лансінг, Мічиган 48824. і освітленість для вилучення явнополюсность. Основні відмінності висхідних моделі уваги лежать в різноманітті функції видалення зубів, стратегії художнього комбінації, і правила в знайти найбільш помітну місце. Перше явне обчислювальна модель знизу вгору уваги було запропоновано Кохом і Ульман 1985, в якому "помітності карту" використовується для кодування стимули помітності на кожному лактації у візуальній сцени. Більш Нещодавно, Іти & Koch та ін. 1998 інтегрований колір, інтенсивність, і орієнтація в основних рисах, і екстрагують інтенсивність.

Інформація в шести шкал для контролю уваги. Бакер в 2001 застосував подібну стратегію активної системи бачення, називається NAVIS (Neural Active Vision), підкреслюючи VI- СУАЛ вибір увагу в динамічному візуальної сцени. Замість безпосередньо, використовуючи деякі низькорівневі функції, як орієнтації і інтенсивність, вони розміщені додаткову обробку для знайти риси середньої ланки, наприклад, симетрію і ексцентричність, побудувати функцію карту. Вольові зрушення уваги також вважається виконується зверху вниз, через положення визначено і особливість- залежить зважування різних карт об'єктів. На початку TOP- вниз уваги моделі вибрали помітні позиції незалежно від того, зайнята об'єктів чи ні, а названий Положення основі управління зверху вниз (наприклад, Olshausen ін., 1993, Tsotsos ін. 1995 та ін Schill. 2001 і т.д.). Нещодавно розширення наукових досліджень були зроблені зусилля по розмістити здатності розпізнавання об'єктів, щоб підвищити TOP- вниз контролю уваги, зокрема, називається об'єктно-орієнтована TOP- вниз уваги (наприклад, Сонце і Фішер +2003 і Вальтер та ін. 2 006 і т.д.). Вищезазначена робота за умови, обчислювальні моделі увагу (знизу вгору і зверху вниз, відповідно) і їх аспекти можливостей розпізнавання об'єкта обмежується, однак, робота стосується питання про їхні загальні взаємодій і інтеграція. Тим більше, що обчислювальна причинності обліку для одночасного розвитку мозку "Де" і "те, що" шляху, використовуючи як знизу вгору, і управління зверху вниз. На жаль, запропонований архітектура не була продемонстрована в масштабованої мережі для інженерна продуктивність уваги і визнання. Крім того, з'єднання зверху вниз були використані для розповсюдження TOP- тільки вниз сигнали, без будь-якої внутрішнього розвитку через нейронні обчислення.

2.4.7 Відео

Розберемо часову інтеграцію інформації у згортальній нейронній мережі. Розглянемо кілька підходів до вилучення інформації по тимчасовій області: Злиття може бути зроблено на початку мережі шляхом зміни першого шару згортки ниє фільтри, щоб розширити в часі, або це може бути зроблено пізно розміщення двох окремих покадрового мережі на деяку відстань часу один від одного і пізніше в процесуальна злиття їх виходи [16]. Ми спочатку описати базове однокадрову згортальну нейронну мережу, а потім обговорити свої розширення в часі відповідно різних типів Fusion. Покадрова , використовуємо однокадрову базову архітектуру Туре зрозуміти внесок статичного зовнішності точність класифікації. Ця мережа схожа на ImageNet завдання виграш модель, Але приймає не ставить розміру 170 х 170 х 3 пікселі замість оригіналу 224 х 224 х 3. Подається на вхід з кроком s , FC(n) є повністю підключений шар з N вузли. Всі шари об'єднують Р Басейн просторово 2 х 2 регіони, що не пересікаються, й усі нормалізаційні шари N визначені. І використовувати ті ж самі параметри: k=2;n= 5;d=10.

В першому шарі повністю підключений. Таким чином, жодна рамка вежа поодинці може виявити будь-який рух, але перша повністю підключений шар може обчислити глобальні характеристики руху шляхом порівняння виходів обох веж. Низька Fusion , Повільний модель Fusion є збалансованим змішувати між цими двома підходами, які повільно запобіжники тимчасова Інформація по всій мережі, наприклад, що вище lay- ERS отримати доступ до все більш глобальний інформація обидва просторові і тимчасові розміри. Це реалізується розширюючи можливості підключення всіх верств згортальних в часі і проведення тимчасових витків на додаток просторовим звивин, щоб обчислити активації, як видно на. У моделі ми використовуємо, перший шар згортального поширити і кожен фільтр тимчасової ступеня Т = 4 на вхід кліп 10 кадрів через дійсний згортки з крокувати 2 і виробляє 4 відповіді в часі. Другий і третій шари вище ітерації цей процес з фільтрами temporal ступеня Т = 2 і крок 2.

Таким чином, третій згортальний шар має доступ до інформації по всіх 10 вхідних кадрів. Так згортальна нейронна мережа зазвичай займає по замовленнях тижнів тренуватися на великомасштабні набори даних, навіть на найшвидших доступних графічних, то продуктивність виконання є критично важливим компонентом для нашої здатності експериментувати з різною архітектурою і hyperparame- тер настройки. Це мотивує підходи для прискорення моделі, зберігаючи їх продуктивність. Існує кілька фронтів цих починаннях, у тому числі поліпшення в обладнанні, схеми квантування ваги, краще оптимі- алгоритми та стратегії Тіоне ініціалізації, але в цій роботі ми орієнтуємося на зміни в архітектурі, які дозволяють швидше працює разів без шкоди для продуктивності. Один з підходів до прискорення мереж полягає у зменшенні кількість шарів і нейронів у кожному шарі, але Аналогічним Лар Ми виявили, що це послідовно знижує продуктивності таких. Замість того, щоб зменшення розміру мережі, ми провели подальші експерименти з навчання із зображеннями низький дозвіл. Тим не менш, у той час як це поліпшило Рен Нін час в мережі, докладно високої частоти в Зображення довів вирішальне значення для досягнення гарної точності. Fovea і контекст потоків. Пропонований multiresolu- архітектура ня прагне знайти компроміс, маючи дві окремі потоки обробки більше двох просторових дозволах. 178 на 178 кадрів відео кліп утворює вхід до мережі. Контекст потік отримує downsam- PLED кадрів на половину початкового просторового дозволу ( 89 на 89 пікселів), а потік ямки отримує центр 89 на 89 область в оригінальному дозволі. Таким чином, те спільне вхід розмірність вдвічі. Примітно, що це займає дизайн Перевага зміщення камери присутня в багатьох онлайн-відео, так як об'єкт інтересу часто займає центральну область. Архітектура зміни. Обидва потоки обробляються ідентична мережі, а в повній моделей кадрів, але, починаючи з 89 на 89 кліпи відео. Оскільки вхід тільки половина просторовий розмір як моделі повнокадрових, виймаємо останній об'єднання шар для того, щоб обидва потоку ще закінчуються шар розміром 7 х 7 х 256 , У активацій з обох потоків об'єднуються і подають в перший повністю підключений шару з щільними з'єднаннями.

2.4.8 Масштабування

Для адаптації архітектури слабо контрольованого навчання можна ввести наступні три модифікації [14]. По-перше, повністю зв'язані шари замінені на згортальні, що дозволяє мати справу з майже довільної розміру зображень в якості вхідних. По-друге, ми маємо явно шукати високому виграші позиції об'єкта за образом, додавши один глобальний шар MaxPool на вихід. По-третє, ми використовуємо функцію витрат, які можуть в явному моделі кілька об'єктів, присутніх у зображенні.

Мережа архітектури (див. рис. 2.16) передбачає фіксованого розміру зображення ділянку 224 × 224 RGB пікселі в якості вхідних і виводить 1 × 1 × D вектор переоцінки класу в якості висновку, де D це число класів. Мета полягає в застосуванні до великих мережу зображень в ползункового ING вікно чином розширюючи тим самим свій вихід до N × м × D де N і м Позначимо число вікна переходи в х - І Y - Напрямок в зображенні, відповідно, обчислюючи D в класі оцінки на всіх вхідних вікон RO-переходи. У той час як цей тип ковзаючого проводили в із застосуванням мережі, щоб витягти зображення незалежно патчі, тут ми досягти того ж ефекту шляхом обробки повністю підключений адаптації шарів, як згорток. Для враховуючи розмір вхідного зображення, повністю підключені шар може бути розглядається як окремий випадок згортки шару, де розмір ядра дорівнює розміру вхідний шар. З Ця процедура вихід кінцевого рівня адаптації FC7 стає 2 × 2 × D Вихід на мапі оцінка для 256 × 256 RGB зображення ввід. Як світовий крок в мережі 32×1 пікселів, додавши, 32 пікселів по ширині або висоті зображення збільшує ширину або висоту вихідного оцінка карті по одним.

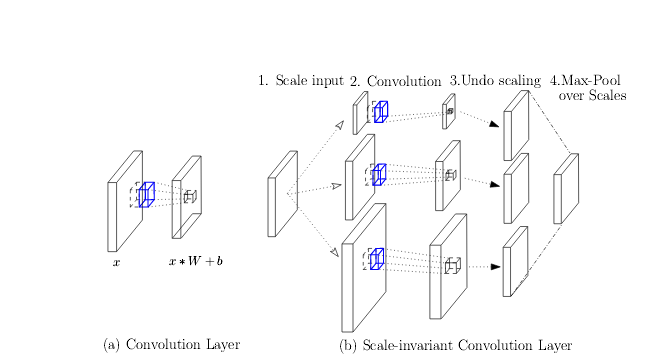


Рисунок 2.16 - Шар масштабування

Отже, наприклад, 2048 × 1024 вхід пікселів буде призвести до 58 × 26. Вихід оцінка мапи, що містить оцінку мережу для всіх класів для різних місцях вхід 224 × 224 вікно з ходу в 32 пікселів. У той час як ця архітектура зазвичай використовується для ефективного класифікації під час випробувань, дивись, наприклад,, Тут ми також використовувати його в навчальний час Ефективно досліджувати всю зображення для можливих місць розташування об'єкта під час слабо Су pervised навчання. Явний пошук положення об'єкта через об'єднання. Метою є виведення оцінка одного зображення рівня для кожного класів об'єктів незалежно від розміру вхідного зображення. Це досягається шляхом об'єднання N × м × До матриця Вихідні показники для N × м різні позиції введення Вікно з використанням глобальної maxpooling роботи в sin- GLE 1 × 1 × D вектор, де D це число класів.

Зауважимо, що операція MaxPool ефективно шукає кращий варіант позиції кандидата об'єкт на зображенні, яка має вирішальне значення для слабо контрольованого навчання, де Точне положення об'єкта в зображенні не дано в навчанні. Крім того, через MaxPool операції вихід мережі стає незалежною від розміру вхідне зображення,

2.4.9 Аргументна архітектура

Обробка інформації за допомогою поділу атрибутів. Використання атрибутів дозволяє передавати інформацію між класами об'єктів. Нехай навчальні зразки, де Х - це довільна функція простору і складається з дискретних K класів.

Завдання навчитися класифікатор для набору тегів , що не перетинається з y. Очевидно, це завдання не може бути вирішена за допомогою звичайного мультикласового класифікатора. Типові класифікатори вивчають один параметр вектор , для кожного тренувального класу . Оскільки класи не були присутніми при кроці навчання, жоден вектор параметрів не може бути отримані для них, і неможливо робити прогнози щодо цих класів для майбутніх зразків. Для того, щоб робити прогнози по класах, для яких немає даних, доступних під час навчання, ми повинні ввести зв'язок між класами в Y і Z. Так як дані для підготовки для майбутніх класів не доступні, цей зв'язок не можна витягти із зразків, і повинен бути вставлений в систему людськими зусиллями. Це вводить деякі серйозні труднощі, при рішенні котрих треба враховувати:

- кількість людських зусиль, щоб вказати нові класи, повинні бути мінімальними, оскільки в іншому випадку збору і маркування додаткових навчальних зразків буде простішим рішенням;

- дані про зв'язки тегів, що потребують тільки загальні знання, є кращими, ніж спеціалізовані експертні знання, тому що останні часто важко і дорого отримати.

Шляхом введення невеликого набору високорівневих семантичних покласових атрибутів буде досягнуто одразо декілька цілей. Ними можуть бути, наприклад, колір і форма для довільних об'єктів, або природне середовище існування для тварини. Люди, як правило, здатні забезпечити гарне знання таких атрибутів, і, отже, можна зібрати необхідну інформацію без багатьох накладних витрат. Оскільки атрибути призначаються на основі кожного класу замість кожного зображення; витрачаються мінімальні зусилля, щоб додати новий клас об'єкту. Для ситуації, коли дані атрибутів цього виду доступні, то введемо класифікацію на основі атрибутів.

В ситуації зі слабкою навчальною підготовкою тестових класи. Якщо для кожного класу і існує пов'язаний атрибут , то ми можемо дізнатися нетривіальний класифікатор шляхом передачі інформацією між Y і Z через А.

Класифікація на основі атрибутів (див. рис. 2.17) дійсно є рішенням проблеми навчання з використанням незвичайних навчальних і тестових класів, і це може бути практично використано для класифікації об'єктів. Для цього, розглянемо два загальних методів для інтегрування атрибутів у мульти-класовій класифікації [12].

Пряме прогнозування атрибутів, як зображено на малюнку нижче, побудовано так, що шар атрибутів розташовано між відокремленими шаром вхідного зображення і шаром тегів класів. Під час навчання, тег вихідного нейрону класу для кожного зразка відповідає деякій активності нейронів шару атрибутів. Отже, будь-який контрольований метод навчання може бути використаний, щоб налаштувати ваги зв'язків атрибутів. Під час тестування, навчені зв'язки дозволяють прогнозувати зі значень атрибутів для кожного випробуваного зразка, визначити який тестовий клас передано на вхід мережі. Під час роботи мережі тестові класи можуть відрізнятися від класів, використовуваних для навчання, а їх зв'язки із шаром атрибутів виставляються вручну, і не вимагають фази навчання.

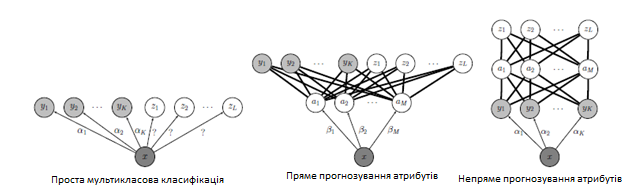


Рисунок 2.17 - Аргументні архітектури мереж

Непряме прогнозування атрибуту також використовує атрибути для передачі знань між класами, але атрибути утворюють сполучний шар між двома шарами тегів класів, один для класів, які, як відомі під час навчання і один для класів, які є невідомими під час навчання. Під час навчальної фази відбувається звичайна мульти-класова класифікація. У режимі тестування, за прогнозами для всіх навчальних класів відбувається маркування виходів нейронів шару атрибутів, за котрим можуть вже бути визначена відповідність тестових класів.

Основна відмінність цих двох підходів лежить в відносинах між навчальними класами та тестовими класами. Безпосередньо навчання атрибути призводить до мережі, де все класи ставляться однаково. Коли клас мітки виводяться під час тестового, рішення для всіх класів засновані тільки на атрибут шар. Ми можемо очікувати, що вона, отже, також обробляти Ситуація, коли підготовка і тестові класи не перетинаються. На противагу цьому, при прогнозуванні значення атрибутів побічно класи підготовки відбувається також час тестування в якості проміжного Функція шар. З одного боку, це може ввести зсув, якщо навчальні класи також є потенційними класи вихідні під час Тестування. З іншого боку, можна стверджувати, що виведення атрибутів шар з шару ярлика, а не із зразків буде виступати в якості регуляризації крок, який створює єдиний розумний атрибутів комбінацій і, отже, робить систему більш надійний. Надалі ми будемо розвивати реалізації для обох методів і порівнювати їх.

В даному розділі проведено аналіз та моделювання основних модулів системи. Розглянуто які класи необхідно застосовувати при роботі з даними сервісу та те, як система буде розгорнута.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 WCF сервіс і розподіл спільної пам'яті

У колишніх версіях HTML не було можливості завантажувати на сайт відразу декілька файлів, для цього потрібно було використовувати сторонні компоненти, написані на Flash, Silverlight і т.п., які дозволяли виводити діалогове вікно вибору декількох файлів. У HTML5 елемент input отримав новий атрибут multiple, а з ним і нову можливість вибирати в діалозі відразу декілька файлів. Крім того, для вибору файлів тільки певного типу, введено ще один новий атрибут accept, який розпізнає MIME media types (стандарт описує передачу різних типів даних в інтернеті). Але потрібно пояснити, що множинна завантаження HTML5 дозволяє відправляти на сервер відразу декілька файлів, але сам процес відправки не сильно змінився. Поки ще немає можливості показу стану завантаження та інших подібних речей, які б автоматично робили завантаження зручніше, без використання додаткових бібліотек або написання допоміжного коду. У HTML5 так може виглядати код розмітки для відправки файлів на сервер (див. рис. 3.1).

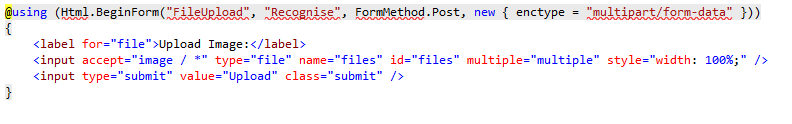


Рисунок 3.1 - Код розмітки для відправки файлів на сервер

Після натискання на кнопку вибору файлів з'явиться діалогове вікно, в якому можна вибрати відразу декілька файлів і завантажити їх одним кліком. Прийом файлів на стороні сервера. Коли файли завантажуються на сервер c ASP.NET MVC, то потрібно пам'ятати про наступні речі (див. рис. 3.2). Якщо завантаження йде з одного елемента input, то файлам присвоюється одне і те ж ім'я. Тобто не можна отримати файл по імені керуючого елемента input. 1 HttpPostedFileBase file = Request.Files ["File1"]; Замість цього можна вибрати всі ключі файлів і через них же запитувати файли у вигляді, наведеному на рисунку 3.2. Більш докладний код контролеру надано в додатку В.

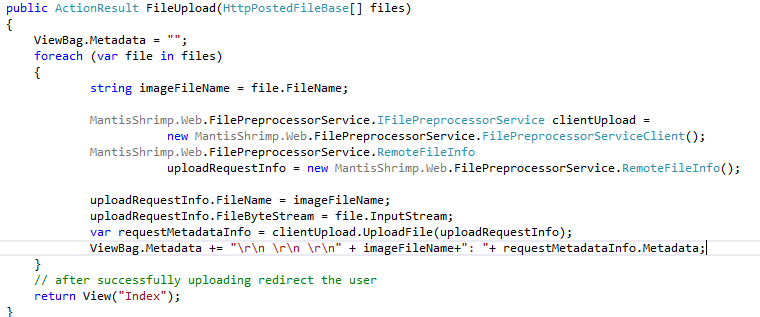


Рисунок 3.2 - Код контролера для прийняття файлів на сервері

На початку розпізнавання WCF сервіс, налаштований на роботу із потоковою передачею зображень отримує із веб-додатку поточну картинку із набору користувача. Для забезпечення безпеки роботи сервісу, доступ до нього відбувається через SSL сертифікати. Таким чином уся відповідальність та логіка надання прав користувачам на доступ до функціоналу розпізнавання лягає на веб-додаток.

В свою чергу веб-додаток оперує ролями і правами, та аутентифікацією користувачів через AuthenticationService.

Також реалізований на базі SOAP і повертає дані в XML-форматі. Обрана WCF технологія має наступні характеристики:

- це удосконалений веб-сервіс (ASMX) підтримуючий різні протоколи, такі як TCP, HTTP, HTTPS, Named Pipes, MSMQ;

- основна проблема WCF, це важкі і складні налаштування;

- це не "open source" рішення але можна використовувати на будь-якому клієнті який вміє працювати з XML;

- може працювати тільки на сервері IIS.

Із веб-інтерфейсу кожна окрема картинка передається на сервісний метод Recognise. Може бути кілька кінцевих точок в сервіс. Кінцеві точки визначають параметри зв'язку.

Кінцева точка є частиною інформації, яка говорить WCF, як побудувати канали зв'язку для виконання передачі і прийому повідомлень. Кінцева точка складається з адреси, зв'язування і контракту.

Адреса - куди відправляти повідомлення. Адреси є унікальним Uniform Resource Locator (URI), яка ідентифікує місцезнаходження служби. Він визначає мережеву адресу для відправки та отримання повідомлень.

Binding - як відправляти повідомлення. Зв'язування визначає, який транспортний протокол використовувати, які повідомлення і формат, який будь-який з WS \* протоколи, які ми хочемо використовувати для конкретної кінцевої точки.

BasicHttpBinding. Заміна для раніше веб-служби на основі ASMX (Методи Active Server). Підтримує:

- HTTP - Hypertext Transfer Protocol;

- HTTPS - Hypertext Transfer Protocol over SSL;

- MTOM - Message Transmission Optimization Mechanism encoding methods.

WsHttpBinding. Використає SOAP поверх HTTP і підтримує надійність, транзакції і безпеку в Інтернеті. Підтримує також:

- HTTP - Hypertext Transfer Protocol;

- HTTPS - Hypertext Transfer Protocol over SSL;

- MTOM - Message Transmission Optimization Mechanism encoding methods.

WSDualHttpBinding. Використовується для договору дуплекс служби, тому що він підтримує двосторонній зв'язок.

WebHttpBinding. Передає інформацію безпосередньо над HTTP або HTTPS без створення SOAP конверт. Це хороший вибір, коли мило не вимагається від клієнта.

NetTcpBinding. Використовується для відправки SOAP повідомлення двійковим кодом з одного комп'ютера на інший. Використовує протокол TCP (Transmission Control Protocol) і включає в себе підтримку надійності, безпеки угод.

NetPeerTcpBinding. Використовується для однолітків-рівному зв'язку на TCP • Зв'язок повинна відбуватися між двома або декількома комп'ютерами.

netNamedPipeBinding. Бінарні закодовані повідомлення SOAP передаються через іменовані канали • Використовується на одному комп'ютері WCF.

netMSMQBinding. У черзі зв'язування використовується для відправки SOAP повідомлення двійковим кодом над MSMQ. Комунікація повинна відбуватися між двома комп'ютерами.

Contract. Контракт забезпечує додаткові деталі про структуруванні змісту різних повідомлень, які будуть використовуватися різних операцій, схильних конкретних кінцевих точок. Наприклад, ми створюємо службу "TopupService", інтерфейс, використовуваний для визначення служби є "ITopupService" і назва проекту є "Утиліта". Контракт на цієї послуги буде Utility.ITopupService.

Три типи WCF паралелізму. Є три шляхи, по яких ви можете впоратися паралелізм в WCF: один, кілька, і поворотний. Щоб вказати WCF паралелізм, ми повинні використовувати тег ServiceBehavior, як показано нижче, з відповідним значенням власність '' ConcurrencyMode.

**Single** - один запит має доступ до об'єкта служби WCF на даний момент часу. Таким чином, тільки один запит буде оброблений в будь-який момент часу. Інші запити, доведеться чекати, поки запит обробляється службою WCF не буде завершено процес.

**Multiple** - у цьому випадку, декілька запитів можуть бути оброблені об'єктом сервісу WCF в будь-який даний момент часу. Іншими словами, запити обробляються одночасно породжуючи декілька потоків на об'єкті ФОС сервера. Так у вас є великою пропускною тут, але ви повинні переконатися, проблеми паралельної, пов'язані з серверними об'єктами WCF.

**Reentrant** - один потік запиту має доступ до об'єкта служби WCF, але нитка може вийти з служби WCF, щоб викликати іншої служби WCF або також можете робити виклик клієнту WCF через зворотний виклик і повторно без безвихідь.

У нормальному запитом WCF та зв'язку відгуку, наступна послідовність дій відбувається:

- клієнт WCF робить запит до об'єкта служби WCF;

- об'єкт служби WCF ініціюється;

- екземпляр служби WCF опрацьовує запит і відправляє відповідь клієнту WCF.

Нижче наведені різні способи, за допомогою яких можна створювати екземпляри WCF:

- створення нового екземпляра служби WCF на кожному WCF методу клієнтському виклику;

- тільки один екземпляр служби WCF повинен бути створений для кожного сеансу клієнта WCF;

- тільки один глобальний екземпляр служби WCF повинен бути створений для всіх клієнтів WCF. Для вирішення зазначених вище сценаріїв, WCF забезпечило три шляхи, по яким можна управляти екземплярами WCF послуг:

- Per call;

- Per session;

- Single instance.

Для роботи в цій системі було обрано режими InstanceContextMode = InstanceContextMode.Single, ConcurrencyMode = ConcurrencyMode.Single, що скорочує час на кожний запит. Оскільки вибірка із файлу набору ваг та будування структури нейронної мережі відбувається лише один раз. У подальшому планується ConcurrencyMode.Multiple - тоді буде використано налаштований масив ваг зв'язків нейронів, киортий буде лише прочитано кожним екземпляром виклику сервісу.

3.2 Компонент FilePreprocessor

У компоненті FilePreprocessor (див. рис. 3.3) виконується основна робота по розпізнаванню. Класи об'єктів комоненту FilePreprocessor зображено на малюнку нижче.

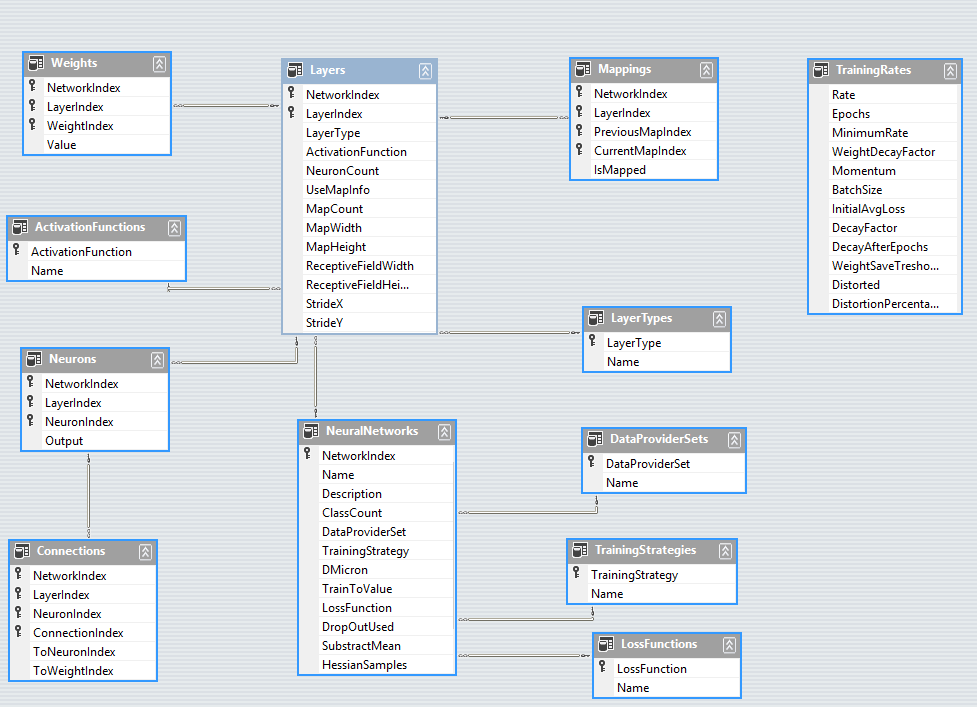


Рисунок 3.3 - Діаграма класів FilePreprocessor

NeuralNetworks клас є основними класом нейронної мережі системи і наразі створюється у одному екземплярі у сервісі. Її поля визначають спільні для всіх нейронів та системи у цілому характеристик, як обрана функція похибки, стратегія тренування та набір для навчання. У цьому клас знаходяться методи, що використовуються для виконання навчання мережі із прямим та зворотнім проходом, та методи визначення класу поточного зображення.

Клас Layers визначає окремий шар мережі, характеризує усі нейрони цього шару. Визначає функції активації цих нейронів їх тип і характер розташування нейронів в мережі. До одного шару відноситься множина екземплярів класу Neurons, кожний з котрих поєднаний з іншим через Connections. Для кожного зв'язку між нейронами зберігається вага зв'язку Weights, що є ключовим для роботи та навчання штучної нейронної мережі. TrainingRates зберігає дані на час навчання мережі, його темп, епоху та поточну стадію.

3.2 Нейрон штучної нейронної мережі

У контексті штучних нейронних мереж, випрямляч є активація функції визначається як де х вхід до нейрона. Це також відомо як функція рампи (див. рис. 3.4), і це аналогічно напівхвильового ректифікації в електротехніці. Ця функція активації Стверджувалося, більш біологічно правдоподібним, ніж широко використовуваний логістичної сигмовидної (який натхненний теорії ймовірностей, см логістичної регресії) і його більш практичним колегу гіперболічний тангенс. Випрямляч, від 2015 року, найпопулярнішою функцією активації глибоких нейронних мереж.

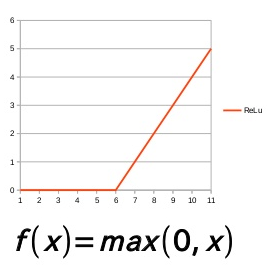


Рисунок 3.4 - Relu функція

Блок використовуючи випрямляч називається також виправлено лінійний блок (Relu). Гладка апроксимація випрямляча є аналітичною функцією ., Які похідна Softplus IS F називається функцією Softplus , тобто логістична функція , Ректифікованого лінійні підрозділи знайти застосування в комп'ютерному, і розпізнавання мови за допомогою нейронних мереж глибокі.

3.3 Набори даних

Застосування набору: Object Recognition. Атрибути: 64 типів довічних атрибутів анотованих для кожного об'єкта вибірки поїзда aPascan і тестових наборів, і випробування aYahoo набору. Типи об'єктів в aPascal: 20 типів об'єктів для Паскаля VOC2008 виклик, тобто люди, птахи, кішки, корови, собаки, коні, вівці, літак, велосипед, човен, автобус, автомобіль, мотоцикл, поїзд, пляшки, стілець, обідній стіл, рослина в горщику, диван і телевізор чи монітор. Типи об'єктів в aYahoo: вовк, зебра, кіз, ослів, мавп, статуя людей, кентавр, сумки, будівлі, водні лижі, перевезення, і кухоль.

CIFAR-10 і CIFAR-100 помічені підмножини 80000000 крихітних зображень даних. Вони були зібрані Алексом Крижевський, Вінод Наїр, і Джеффрі Хінтоном. CIFAR-10 набір даних CIFAR-10 набір даних складається з 60000 32x32 кольорових зображень в 10-х класів, з 6000 зображень в класі. Є 50000 навчальних зображень і 10000 тестові зображення. Набір даних складається з п'яти навчальних партій і одного пробної партії, кожна з яких 10000 зображень. Тест партія містить рівно 1000 випадково вибраних зображень з кожного класу. Навчальні партії містять залишилися зображень у випадковому порядку, але деякі навчальні партії може містити кілька зображень з одного класу, ніж інший. Між ними, навчальні партії містять рівно +5000 зображень з кожного класу. Ось класи в наборі даних, а також 10 випадкових зображень з кожного: Класи повністю взаємовиключними. Там немає перекриття між легкових і вантажних автомобілів. "Автомобілі" включає в себе седани, позашляховики, речі такого роду. "Вантажівка" включає в себе тільки великі вантажівки. Ні включає пікапи.

ВИСНОВКИ

Була спроектована інформаційна система, котра вирішує задачу обробки великого об'єму інформації, вилученої із медіа-файлів через Інтернет. Система була спланована з багаторівневою архітектурою. Була досягнута висока горизонтальна масштабованість системи завдяки реалізації додатку на базі архітектури типу «Microservices». Було спроектовано розподіл модулів системи по сервісам доступу даних, аутентифікації та розпізнавання. Шляхом аналізу у якості платформи для зберігання та обробки даних було вибрано Hadoop. Система була протестована на кластері з 60 вузлів, що розгорнутий на базі хмарної платформи Microsoft Azure. Сервіс для розробки за використанням Hadoop зветься HDInsight. Також було виявлено, що архітектура платформи Hadoop дозволяє масштабувати рівень даних до 4000 серверів (дані надані організацією Apache Software Foundation), що повністю покриває потреби системи. Експерементально було доведено, ща Hadoop та HBase гарантують 99.9999% выдмовостійкість завдяки реплікації даних між вузлами кластера.

Виведено структуру нейронної мережі, що підходить в цій системі для проведення розпізнавання масиву зображень. Знайдені шляхи доопрацювання системи. Реалізовано систему розпізнавання зображень у формі WCF-сервісу із використанням багатошарової нейронної мережі із згортальними шарами. Розроблено веб-додаток для завантаження на сервіс зображень. Сервіс, ща обробляє вхідні зображення також розгорнутий з використанням платформи Microsoft Azure, що гарантує його практично необмежену горизонтальну масштабованість.

Докладніше про процесс розробки системи описано у презентаціі (див. додаток А).

Фрагменти програмного коду розробленої системи наведено у додатку Б.

Результати цієї атестаційної роботи були опубліковані на 19-му Міжнародному молодіжному форумі "Радіоелектроніка та молодь у ХХІ столітті" на конференції "Комп'ютерні, програмні та інтернет-технології, програмування комп'ютерних мобільних систем" (див. додаток В).

ПЕРЕЛІК ПОСИЛАНЬ

1. Big Data Analytics Using Neural networks [Текст] / C. Sharma, 2014. – 100 p.

2. Крис Боуэн. Основы Windows Communication Foundation для .NET Framework 3.5 [Текст] / К. Боуэн, С. Резник, Р. Крейн, 2008. – 478 с.

3. On Intelligence [Текст] / J. Hawkins, S. Blakeslee, 2004. – 115 p.

4. Дейт, К. Дж. Введение в системы баз данных [Текст] : пер. з англ. / К.Дж.Дейт. - 8-е издание. – М.: Издательский дом «Вильямс», 2008. – 1328 с.

5. Steve Sanderson and Adam Freeman «Pro ASP.NET MVC 4» [Текст] / S. Sanderson, A.Freeman. - Business Media New York, 233 Spring street, 6th Floor, 2012. – 729 p.

6. Фаулер М. UML Основы [Текст] : пер. з англ.. / М. Фаулер, К. Скотт. – СПб: Символ-Плюс, 2002. – 192 с.

7. Microsoft developers network [Електронний ресурс]: форум з програмування.– Режим доступу: http://msdn.microsoft.com.

8. ASP.NET [Електронний ресурс]: портал для розробників на платформі ASP.NET. – Режим доступу: http://asp.net.

9. ASP.NET MVC 4 [Електронний ресурс]: книга. – Режим доступу: http://metanit.com/sharp/mvc.

10. ASP.NET MVC 3 [Електронний ресурс]: книга. – Режим доступу: http://unmail22.narod.ru/Books/ASPNET\_MVC3/ASPNET\_MVC3.html.

11. Герберт Шилдт. C# 3.0 [Текст] : повне керівництво / Герберт Шилдт; пер. з англ. – М.: ООО «И.Д. Вильямс», 2010. – 992 с.

12. Нейроні мережі [Електронний ресурс]: книга. – Режим доступу: http://logic.pdmi.ras.ru/~sergey/teaching/mlaptu11/02-ann.pdf.

13. WWN-2: A Biologically Inspired Neural Network for Concurrent Visual Attention and Recognition [Електронний ресурс]: книга. – Режим доступу: https://web.cse.msu.edu/~jizhengp/WWN2\_Ji.pdf.

14. Weakly-supervised learning with convolutional neural networks [Електронний ресурс]: книга. – Режим доступу: http://www.di.ens.fr/~josef/publications/Oquab15.pdf.

15. Алгоритм Левенберга-Марквардта [Електронний ресурс]: книга. – Режим доступу: http://neuronus.com/nn/38-theory/246-algoritm-levenberga-markvardta.html**.**

16. The Hessian Matrix [Електронний ресурс]: книга. – Режим доступу: http://www.cedar.buffalo.edu/~srihari/CSE574/Chap5/Chap5.4-Hessian.pdf.

17. Efficient learning and second−order methods [Електронний ресурс]: книга. – Режим доступу: http://www-labs.iro. umontreal.ca/~vincentp/ift3390/lectures/ YannNipsTutorial.pdf.

18. Large-scale Video Classification with Convolutional Neural Networks [Електронний ресурс]: книга. – Режим доступу: http://vision.stanford.edu/pdf/karpathy14.pdf.

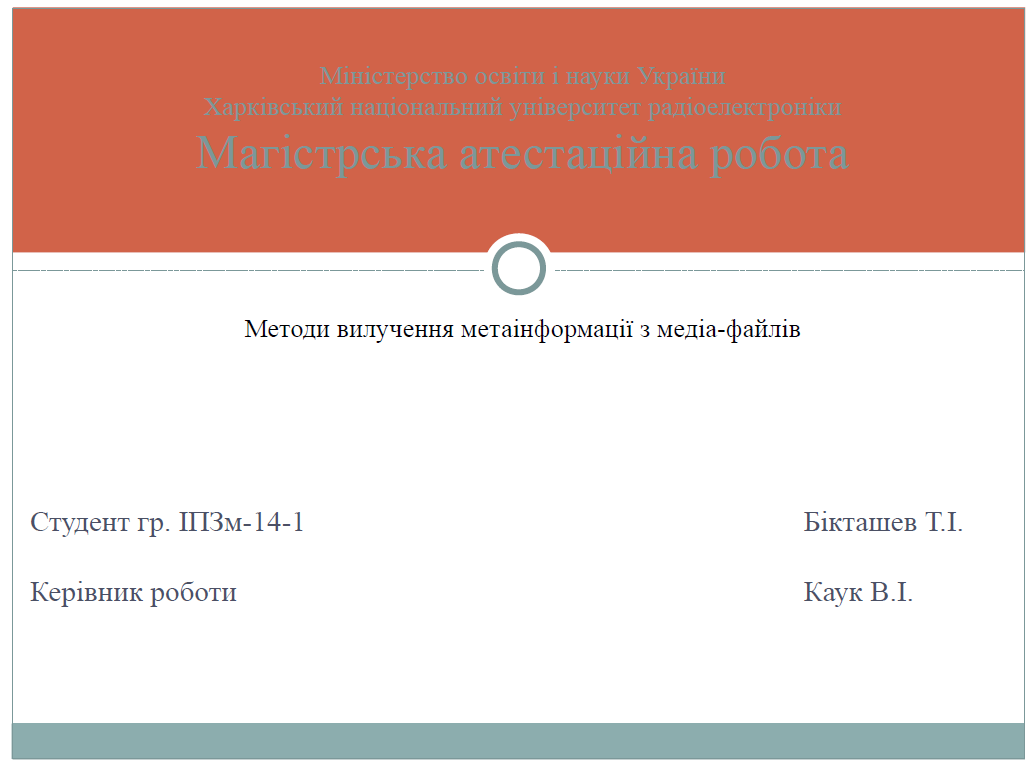
19. Learning To Detect Unseen Object Classes by Between-Class Attribute Transfer [Електронний ресурс]: книга. – Режим доступу: http://unmail22.narod.ru /Books/ASPNET\_MVC3/ASPNET\_MVC3.html.

20. Describing Objects by their Attributes [Електронний ресурс]: книга. – Режим доступу: http://web.engr.illinois.edu/~iendres2/publications/cvpr2009\_att.pdf.

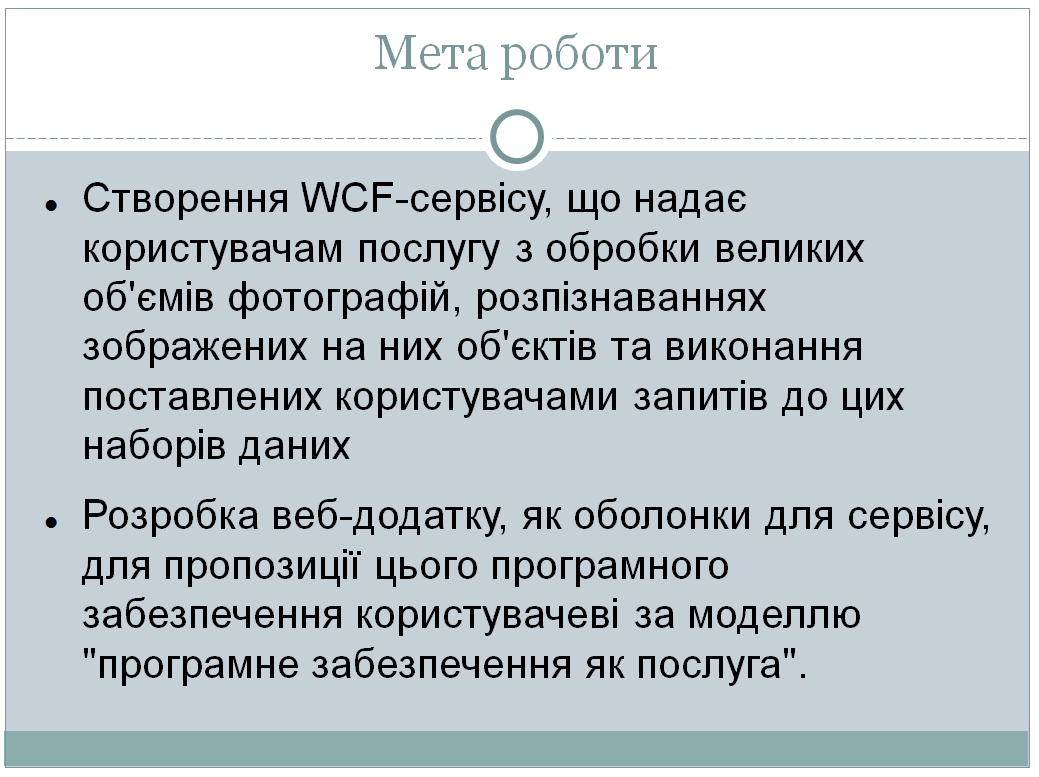
Додаток А

Слайди презентації

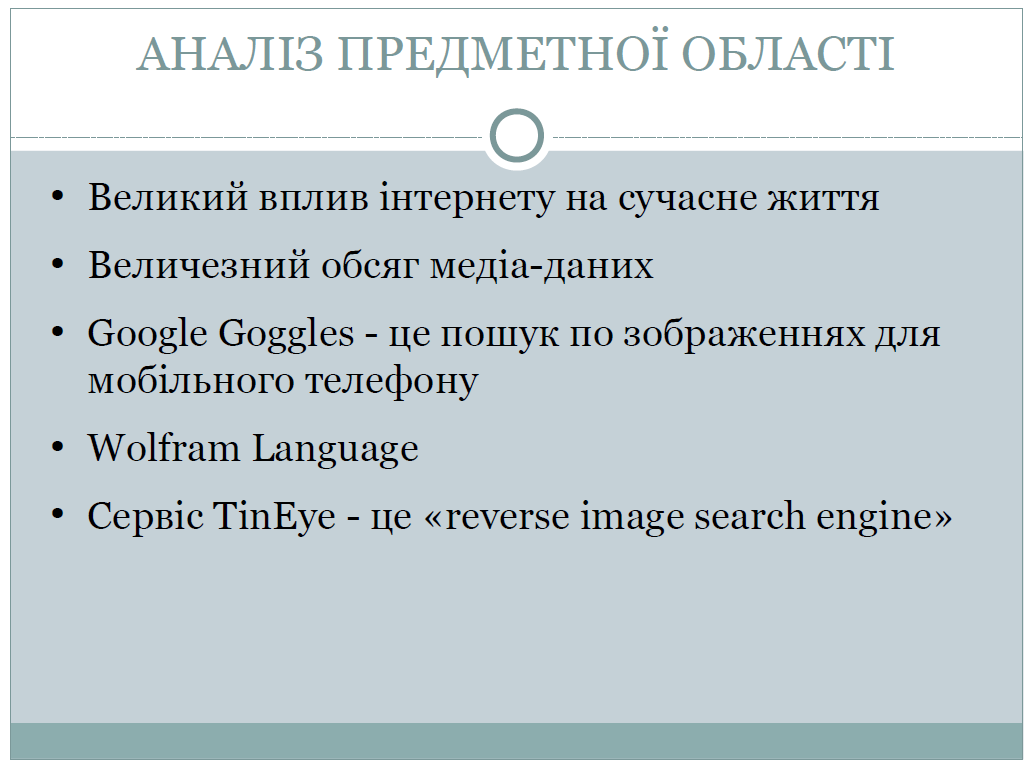
Слайд 1



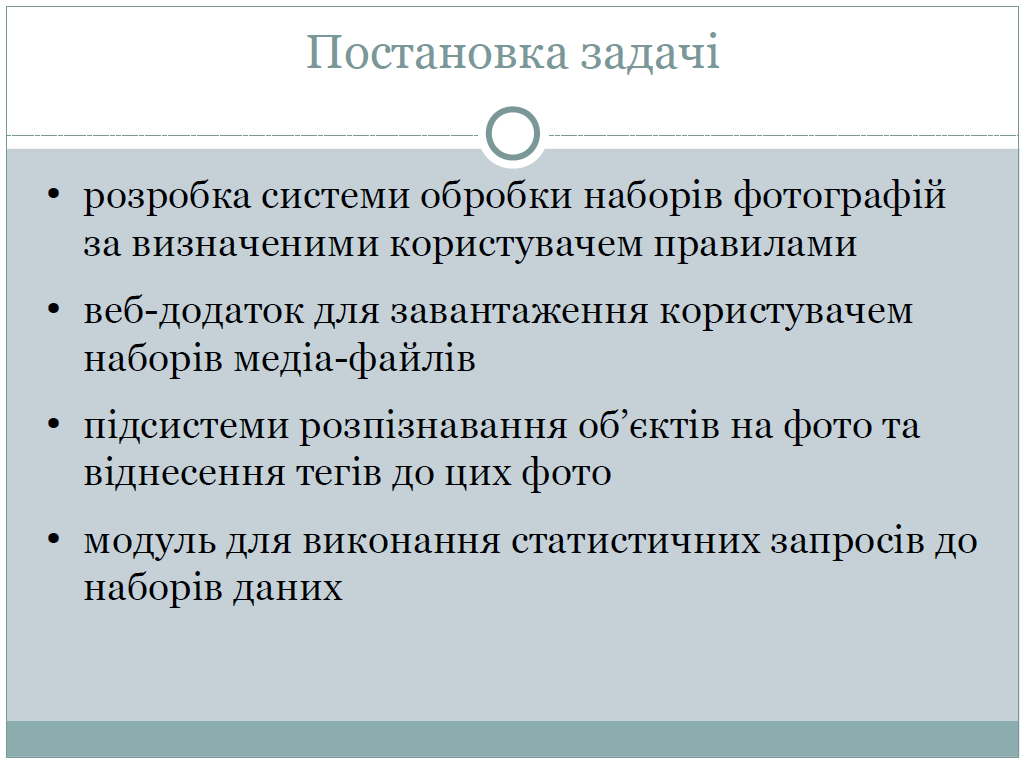
Слайд 2



Слайд 3



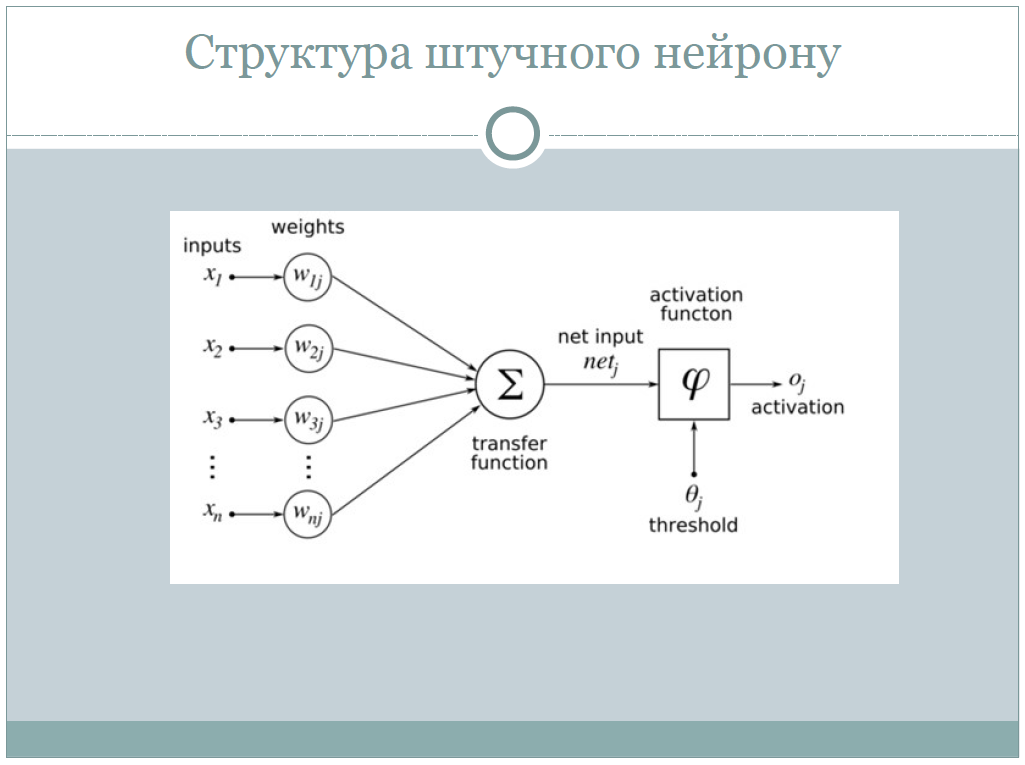
Слайд 4



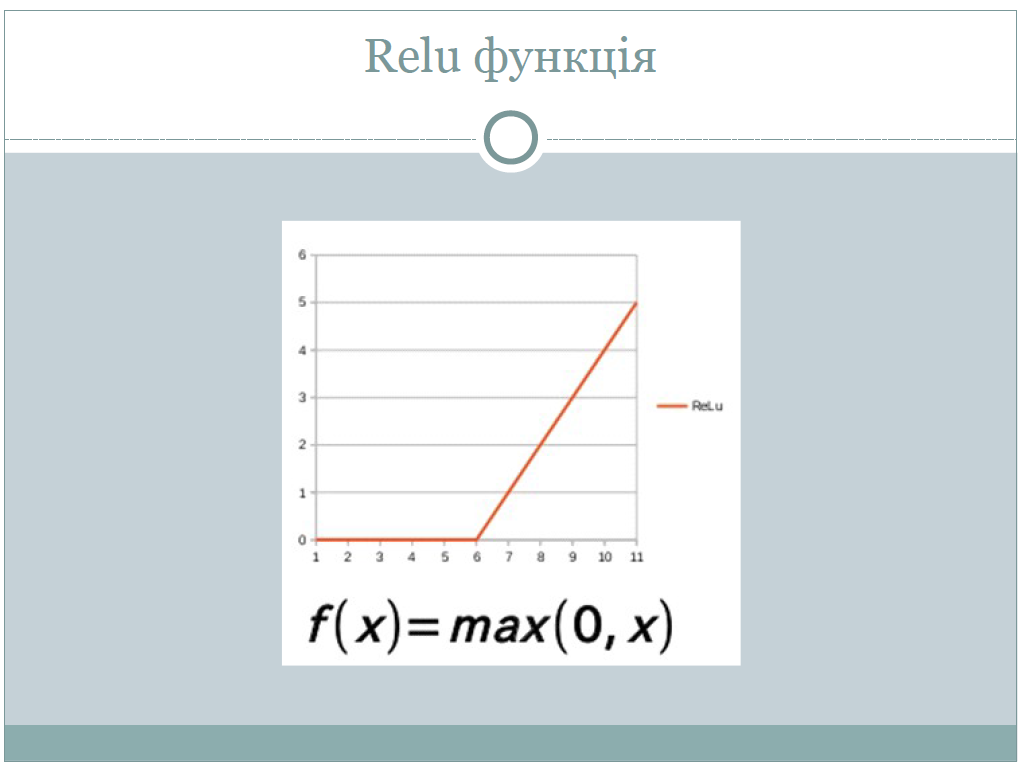
Слайд 5



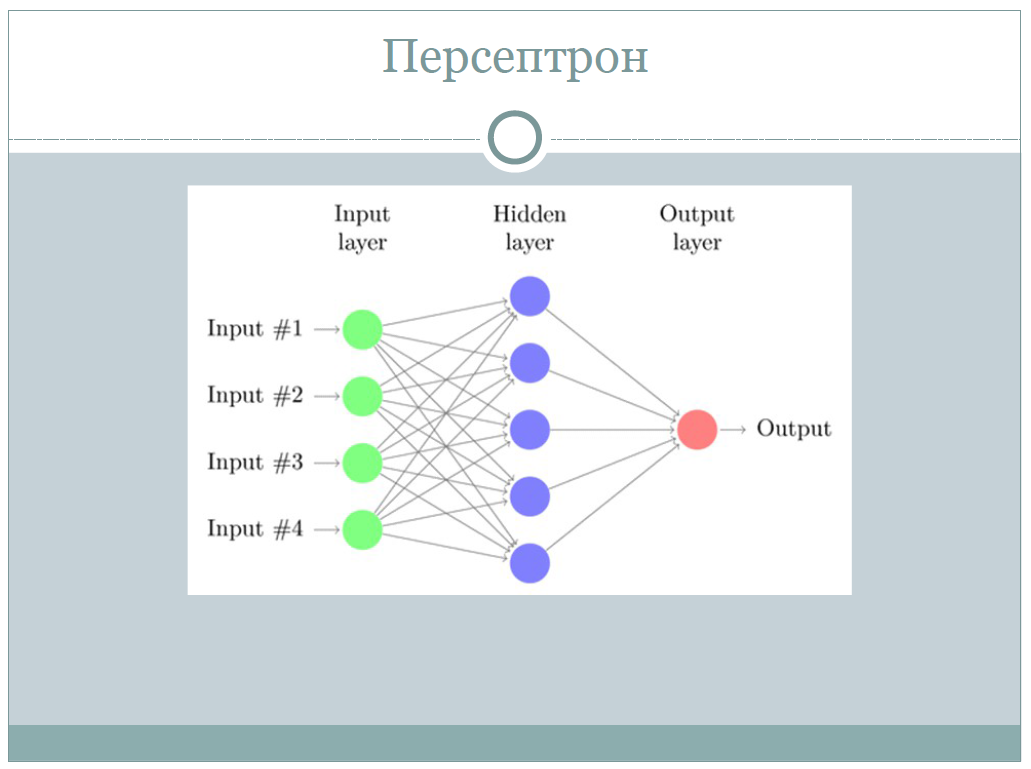
Слайд 6



Слайд 7



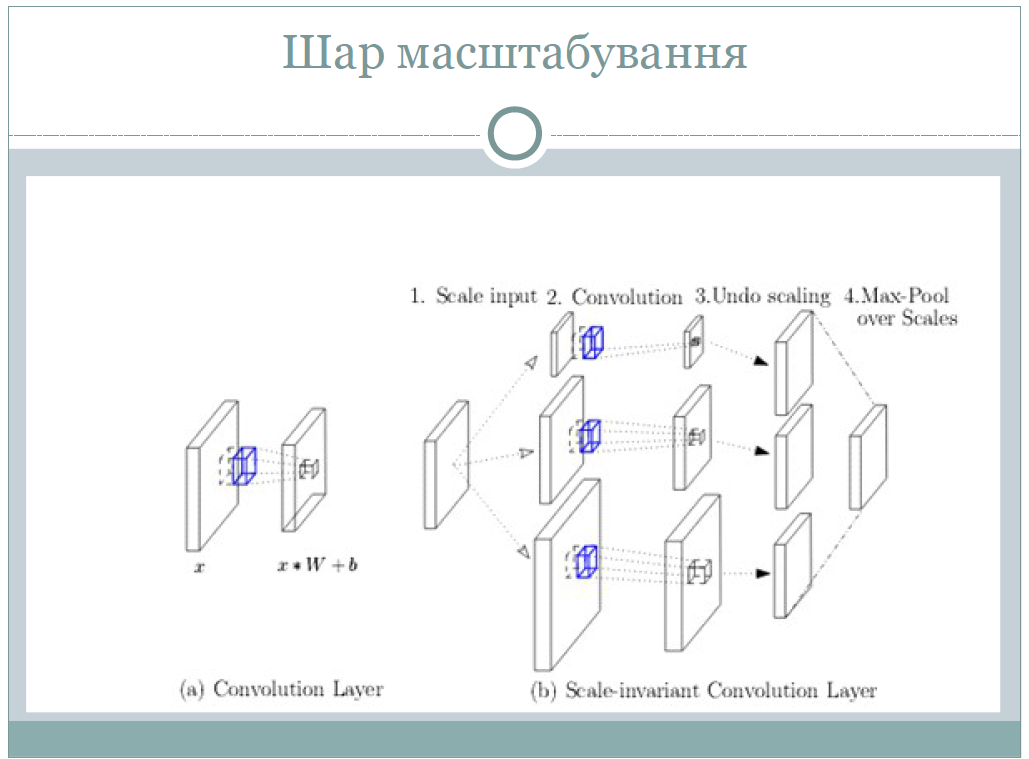
Слайд 8



Слайд 9



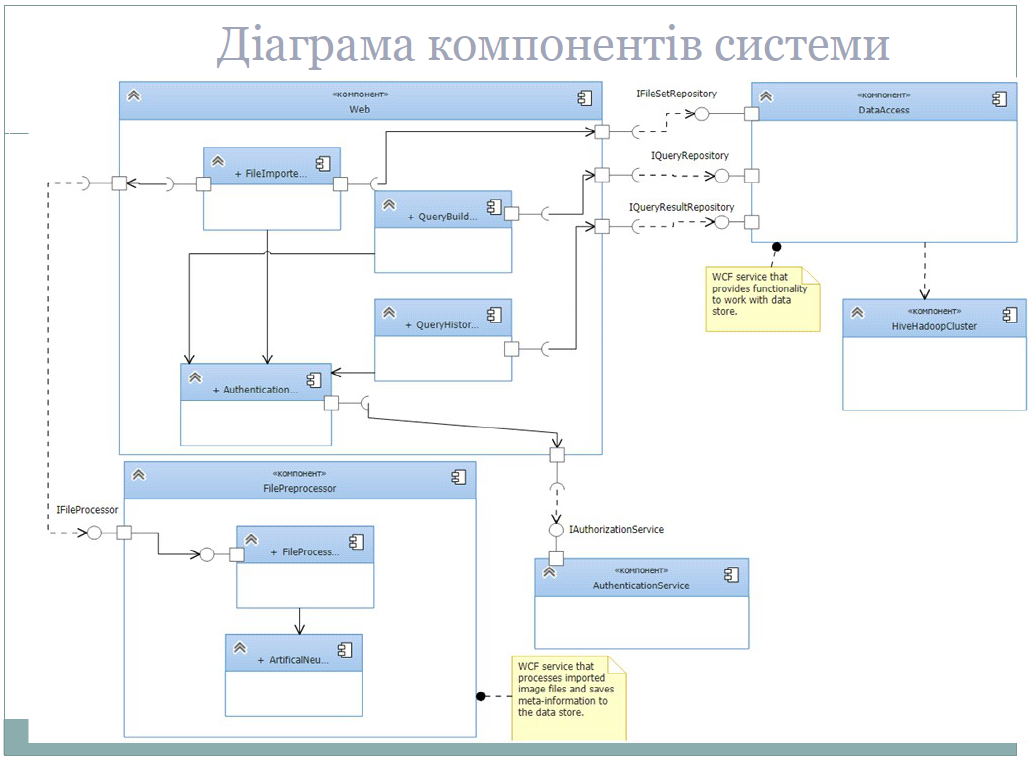
Слайд 10



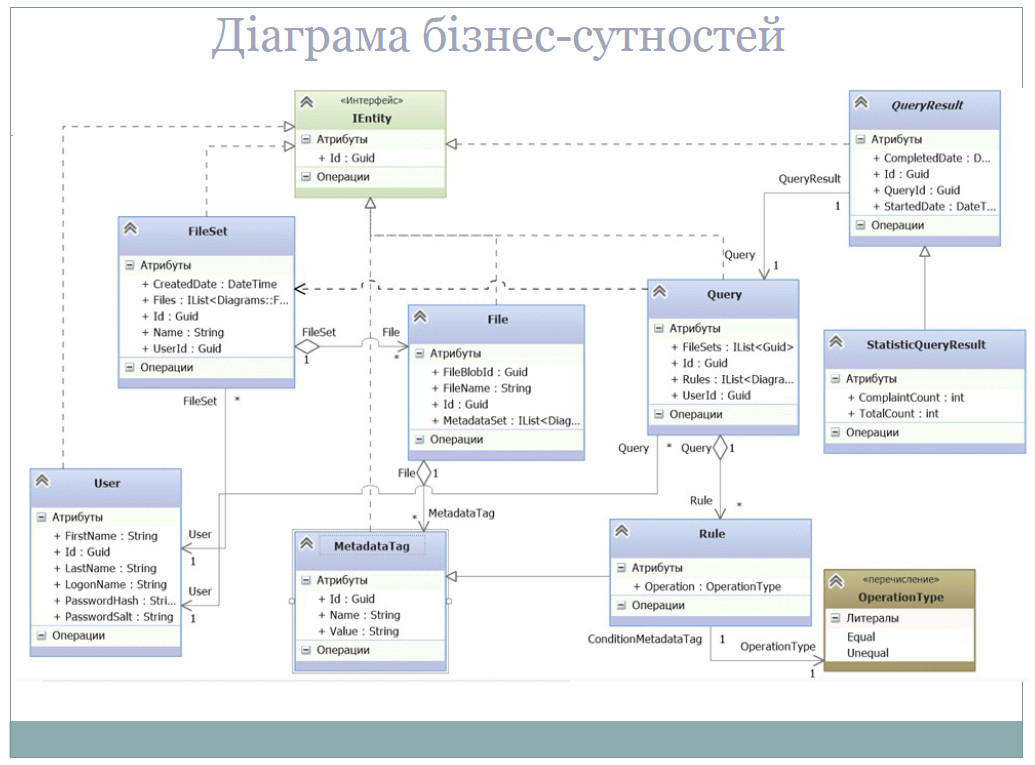
Слайд 11



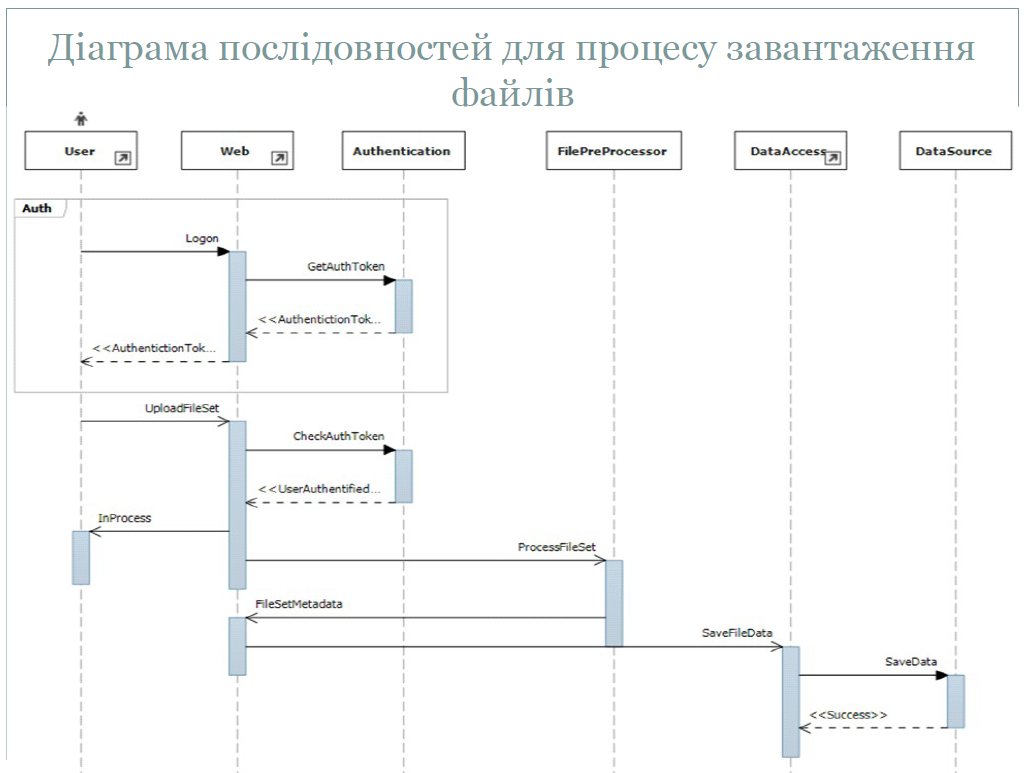
Слайд 12



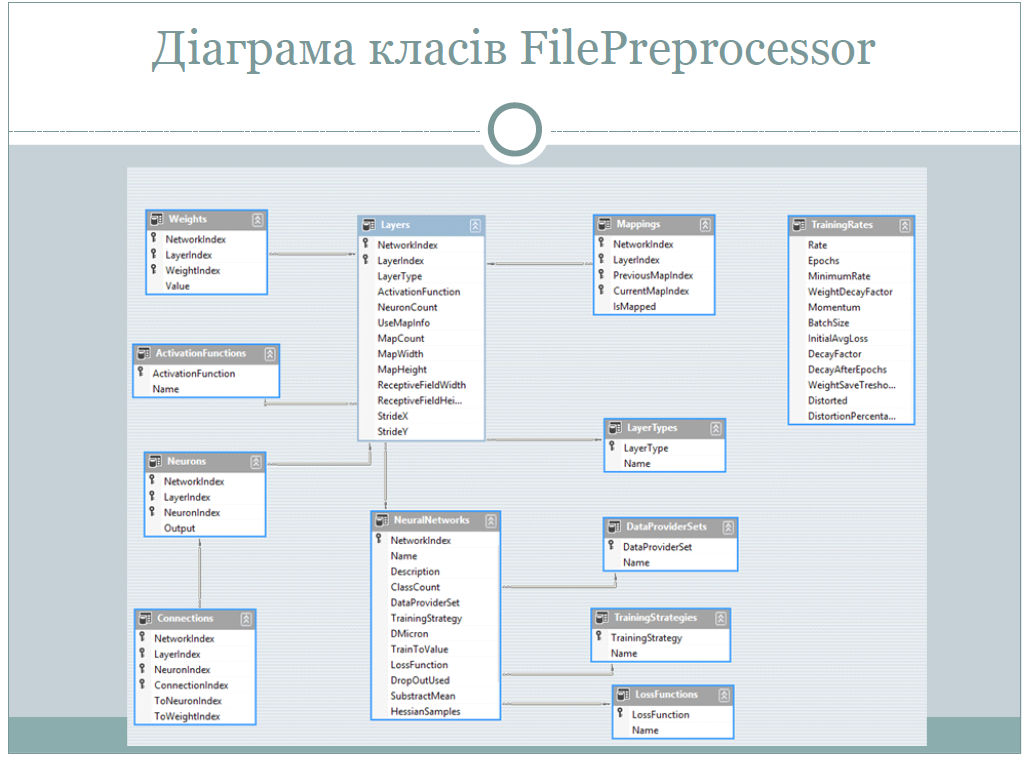
Слайд 13



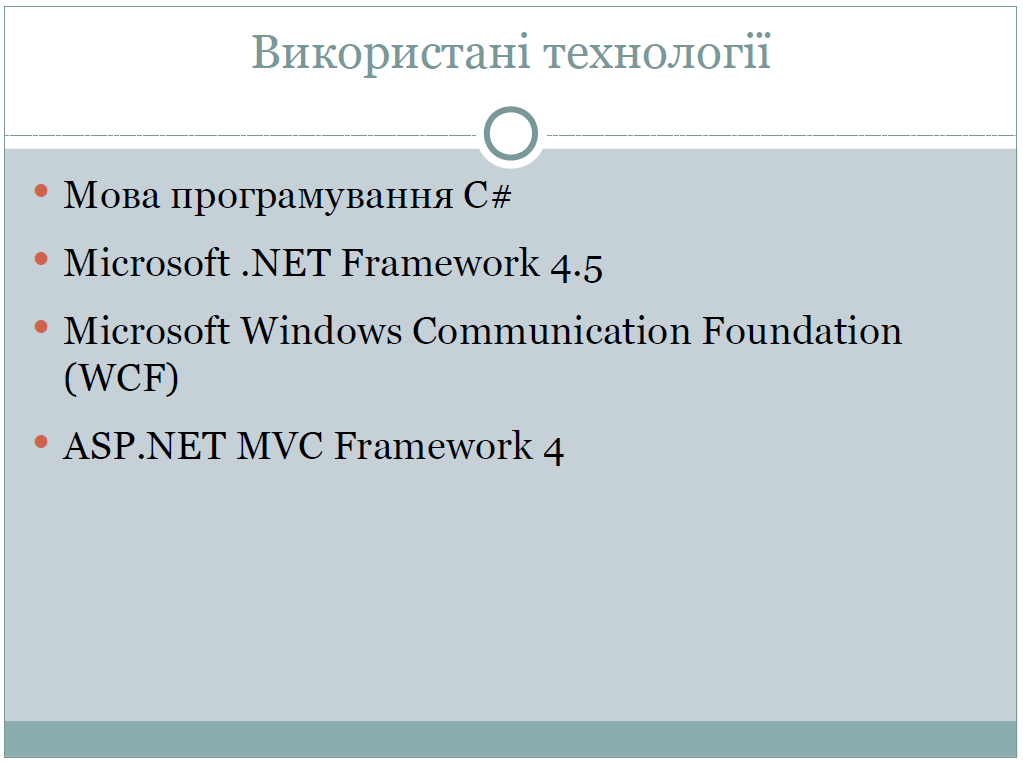
Слайд 14



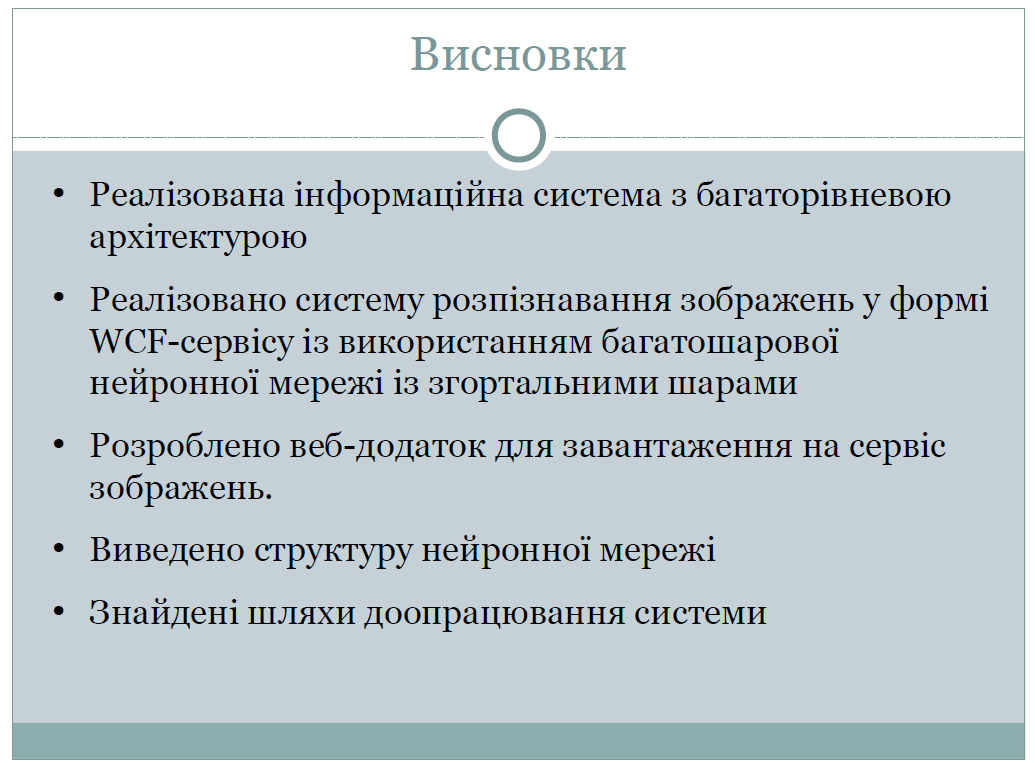
Слайд 15



Слайд 16



Слайд 17



Додаток Б

Програмний код

Програмна реаліація компоненту FilePreProcessor:

public class Program

{

public static void Main(string[] args)

{

FileProcessor fp = new FileProcessor();

fp.Init();

fp.OnCalculate();

}

}

public class FileProcessor

{

string storageDirectory;

int inputChannels = 3;

int inputWidth = 32;

int inputHeight = 32;

DataProvider dataProvider;

NeuralNetwork neuralNetwork;

ImageData currentSample;

private string physicalPath = Environment

.GetFolderPath(Environment.SpecialFolder.MyDocuments);

public string PhysicalPath

{

get { return physicalPath; }

set { physicalPath = value; }

}

private NeuralNetwork InitializeDefaultNeuralNetwork()

{

storageDirectory = PhysicalPath + @"\CNNData";

dataProvider = new DataProvider(storageDirectory);

neuralNetwork = new NeuralNetwork(dataProvider,

"CIFAR10",

10, 1D,

LossFunctions.CrossEntropy,

DataProviderSets.CIFAR10, TrainingStrategy.SGDLevenbergMarquardt,

minimumArgLimit: 0.1);

neuralNetwork.AddLayer(LayerTypes.Input,

inputChannels,

inputWidth,

inputHeight);

bool[] maps = new bool[3 \* 64] {};

neuralNetwork.AddLayer(LayerTypes.Convolutional,

ActivationFunctions.ReLU,

64, 28, 28, 5, 5,

1, 1, 0, 0,

new Mappings(maps));

neuralNetwork.AddLayer(LayerTypes.StochasticPooling,

ActivationFunctions.Ident,

64, 14, 14,

3, 3, 2, 2);

neuralNetwork.AddLayer(LayerTypes.Convolutional,

ActivationFunctions.ReLU,

64, 10, 10, 5, 5,

1, 1, 0, 0, new Mappings(64, 64, 66, 1));

neuralNetwork.AddLayer(LayerTypes.StochasticPooling,

ActivationFunctions.Ident,

64, 5, 5,

3, 3, 2, 2);

neuralNetwork.AddLayer(LayerTypes.Local,

ActivationFunctions.Logistic,

384, 1, 1, 5, 5,

1, 1, 0, 0, 50);

neuralNetwork.AddLayer(LayerTypes.FullyConnected,

ActivationFunctions.SoftMax, 10);

neuralNetwork.InitializeWeights();

network.LoadWeights(StorageDirectory +

@"\CIFAR10.weights-bin");

return neuralNetwork;

}

public void Init()

{

InitializeDefaultNeuralNetwork();

Load(PhysicalPath

+ @"\CNNData\CIFAR10.weights-bin");

//dataProvider.LoadCIFAR10DataSet();

}

public string Recognise(Bitmap bitmap)

{

ImageData imageData = new ImageData();

imageData.ProcessFromBitmap(bitmap);

var mapper = new ImageInuputMapper(imageData,

bitmap.Width, bitmap.Height,

32, 32, 3);

return Calculate(imageData.Resize(

bitmap.Width,

bitmap.Height,

32, 32, 3,

ImageData.Interpolation.Bilinear)).Detailed;

}

public void OnCalculate()

{

currentSample = dataProvider.TrainingSamples[11];

StringBuilder sb = new StringBuilder();

sb.AppendLine("Calculating...");

double[] data =

currentSample.SubstractMean(dataProvider);

for (int i = 0; i < dataProvider.SampleSize \*

dataProvider.SampleChannels; i++)

{

neuralNetwork.Layers[0].Neurons[i].Output

= data[i];

}

Stopwatch timer = new Stopwatch();

timer.Restart();

neuralNetwork.Calculate();

int bestIndex = neuralNetwork.Recognized();

double patternLoss =

neuralNetwork.GetSampleLoss(

neuralNetwork.CurrentSample.Label);

timer.Stop();

switch (dataProvider.CurrentDataSet)

{

case DataProviderSets.MNIST:

for (int i = 0;

i < neuralNetwork.LastLayer.NeuronCount;

i++)

{

sb.AppendLine(i.ToString() + " \t\t= "

+ neuralNetwork.LastLayer.Neurons[i]

.Output

.ToString("0.###################;

-0.###################",

CultureInfo.CurrentUICulture));

}

sb.AppendLine("\r\nSample loss: "

+ patternLoss.ToString("N17",

CultureInfo.CurrentUICulture));

sb.AppendLine("Highest probability: "

+ bestIndex.ToString());

break;

case DataProviderSets.CIFAR10:

CIFAR10Classes label;

for (int i = 0;

i < neuralNetwork.ClassCount; i++)

{

label = (CIFAR10Classes)i;

if (label.ToString().Length <= 5)

sb.AppendFormat(

CultureInfo.CurrentUICulture,

"{0} \t\t= {1}\r\n",

label,

neuralNetwork.LastLayer.Neurons[i].Output

.ToString(" 0.###################;

-0.###################",

CultureInfo.CurrentUICulture));

else

sb.AppendFormat(

CultureInfo.CurrentUICulture,

"{0} \t= {1}\r\n",

label,

neuralNetwork.LastLayer.Neurons[i]

.Output

.ToString(" 0.###################;

-0.###################",

CultureInfo.CurrentUICulture));

}

sb.AppendLine("\r\nSample loss: "

+ patternLoss

.ToString("N17",

CultureInfo.CurrentUICulture));

sb.AppendLine("Highest probability: "

+ ((CIFAR10Classes)bestIndex)

.ToString());

break;

}

if (bestIndex == currentSample.Label)

sb.AppendLine("Correctly recognized");

else

sb.AppendLine("Not correctly recognized");

sb.AppendFormat("Time: {0} ms.",

timer.ElapsedMilliseconds);

Console.WriteLine(sb.ToString());

//ShowCalculateOutput();

Console.WriteLine("Ready");

}

private RecogniseResult Calculate(ImageData imageData)

{

RecogniseResult result = new RecogniseResult();

StringBuilder sb = new StringBuilder();

sb.AppendLine("Calculating...");

for (int i = 0;

i < inputWidth \* inputHeight \* inputChannels; i++)

{

neuralNetwork.Layers[0].Neurons[i].Output =

(((double)imageData.Image[i] / 255D)

\* neuralNetwork.Spread)

+ neuralNetwork.Min;

}

Stopwatch timer = new Stopwatch();

timer.Restart();

neuralNetwork.Calculate();

int bestIndex = neuralNetwork.Recognized();

double patternLoss =

neuralNetwork.GetSampleLoss(imageData.Label);

timer.Stop();

switch (dataProvider.CurrentDataSet)

{

case DataProviderSets.MNIST:

for (int i = 0;

i < neuralNetwork.LastLayer.NeuronCount;

i++)

{

sb.AppendLine(i.ToString() + " \t\t= "

+ neuralNetwork.LastLayer.Neurons[i]

.Output.ToString(" 0.###################;

-0.###################",

CultureInfo.CurrentUICulture));

}

sb.AppendLine("\r\nSample loss: "

+ patternLoss.ToString("N17",

CultureInfo.CurrentUICulture));

result.Tag =

((CIFAR10Classes)bestIndex).ToString();

sb.AppendLine("Highest probability: "

+ result.Tag);

break;

case DataProviderSets.CIFAR10:

CIFAR10Classes label;

for (int i = 0;

i < neuralNetwork.ClassCount; i++)

{

label = (CIFAR10Classes)i;

if (label.ToString().Length <= 5)

sb.AppendFormat(

CultureInfo.CurrentUICulture,

"{0} \t\t= {1}\r\n",

label,

neuralNetwork.LastLayer.Neurons[i]

.Output

.ToString(

"0.###################;

-0.###################",

CultureInfo.CurrentUICulture));

else

sb.AppendFormat(

CultureInfo.CurrentUICulture,

"{0} \t= {1}\r\n",

label,

neuralNetwork.LastLayer.Neurons[i]

.Output

.ToString("0.###################;

-0.###################",

CultureInfo.CurrentUICulture));

}

sb.AppendLine("\r\nSample loss: "

+ patternLoss.ToString("N17",

CultureInfo.CurrentUICulture));

result.Tag =

((CIFAR10Classes)bestIndex).ToString();

sb.AppendLine("Highest probability: "

+ result.Tag);

break;

}

if (bestIndex == imageData.Label)

sb.AppendLine("Correctly recognized");

else

sb.AppendLine("Not correctly recognized");

sb.AppendFormat("Time: {0} ms.",

timer.ElapsedMilliseconds);

result.Detailed = sb.ToString();

return result;

//ShowCalculateOutput();

}

private void Load(string fileName)

{

if (fileName.Contains(".definition-xml"))

{

NeuralNetwork network =

NeuralNetwork.LoadDefinition(fileName,

dataProvider);

if (network != null)

{

dataProvider.LoadDataset(

network.DataProviderSet);

}

}

if (fileName.Contains(".zip"))

{

NeuralNetwork network =

NeuralNetwork.LoadFullyZipped(fileName,

dataProvider);

if (network != null)

{

dataProvider.LoadDataset(

network.DataProviderSet);

}

}

if (fileName.Contains(".weights-bin"))

{

if (neuralNetwork.LoadWeights(fileName))

{

}

}

if (fileName.Contains(".weights-xml-bin")

&& fileName.Contains(neuralNetwork.Name))

{

if (neuralNetwork.LoadWeightsXmlBin(fileName))

{

}

}

}

}

public class RecogniseResult

{

public string Tag { get; set; }

public string Detailed { get; set; }

}

public class ImageInuputMapper

{

public ImageData ImageData { get; set; }

public int OriginalWidth { get; set; }

public int OriginalHeight { get; set; }

public int OutputWidth { get; set; }

public int OutputHeight { get; set; }

public int ChannelsCount { get; set; }

double offesetRatio { get; set; }

double scaleRatio { get; set; }

public ImageInuputMapper(ImageData imageData,

int originalWidth,

int originalHeight,

int outputWidth,

int outputHeight,

int channelsCount,

double offesetRatio = 0.5,

double scaleRatio = 0.5)

{

OriginalWidth = originalWidth;

OriginalHeight = originalHeight;

OutputWidth = outputWidth;

OutputHeight = outputHeight;

ImageData = imageData;

this.offesetRatio = offesetRatio;

this.scaleRatio = scaleRatio;

ChannelsCount = channelsCount;

}

public string MapAndCall(Func<ImageData,

RecogniseResult> calculateMethod)

{

int currentHeight = OriginalHeight;

int currentWidth = OriginalWidth;

List<string> result = new List<string>();

if (scaleRatio != 1)

{

while (currentHeight >= OutputHeight

&& currentWidth >= OutputWidth)

{

int widthOffset = (int)(currentWidth

\* offesetRatio);

int heightOffset = (int)(currentHeight

\* offesetRatio);

int colCount = (int)((OriginalWidth

- currentWidth) / widthOffset) + 1;

int rowCount = (int)((OriginalHeight

- currentHeight) / heightOffset) + 1;

string[,] currentResults =

new string[rowCount, colCount];

Parallel.For(0, rowCount, y =>

{

int yPos = y \* heightOffset;

Parallel.For(0, colCount, x =>

{

int xPos = x \* widthOffset;

ImageData croppedImage =

ImageData.Crop(xPos,

yPos,

OriginalWidth,

OriginalHeight,

currentWidth,

currentHeight,

ChannelsCount);

ImageData scaledImage =

croppedImage.Resize(currentWidth,

currentHeight,

OutputWidth,

OutputHeight,

ChannelsCount,

ImageData.Interpolation

.NearestNeighbor);

RecogniseResult currentResult =

calculateMethod(scaledImage);

currentResults[y, x] =

String.Format(

"{0} ({1},{2}) {3}X{4}",

currentResult.Tag,

xPos, yPos,

currentWidth,

currentHeight);

});

});

result.AddRange(currentResults

.Cast<string>());

currentHeight = (int)(currentHeight

\* scaleRatio);

currentWidth = (int)(currentWidth

\* scaleRatio);

}

}

return string.Join(",", result);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.ServiceModel;

using System.Web;

namespace MantisShrimp.FilePreprocessor.WcfService

{

[ServiceContract]

public interface ITransferService

{

[OperationContract]

RemoteFileInfo DownloadFile(DownloadRequest request);

[OperationContract]

void UploadFile(RemoteFileInfo request);

}

[MessageContract]

public class DownloadRequest

{

[MessageBodyMember]

public string FileName;

}

[MessageContract]

public class RemoteFileInfo : IDisposable

{

[MessageHeader(MustUnderstand = true)]

public string FileName;

[MessageHeader(MustUnderstand = true)]

public long Length;

[MessageBodyMember(Order = 1)]

public System.IO.Stream FileByteStream;

[MessageHeader(MustUnderstand = true)]

public string Metadata;

public void Dispose()

{

if (FileByteStream != null)

{

FileByteStream.Close();

FileByteStream = null;

}

}

}

[MessageContract]

public class RemoteFileMetadataInfo

{

public RemoteFileMetadataInfo(RemoteFileInfo rfi)

{

FileName = rfi.FileName;

Length = rfi.Length;

Metadata = rfi.Metadata;

}

[MessageHeader(MustUnderstand = true)]

public string FileName;

[MessageHeader(MustUnderstand = true)]

public long Length;

[MessageHeader(MustUnderstand = true)]

public string Metadata;

}

}

Програмна реаліація компоненту контракту WCF-сервісу:

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.ServiceModel.Web;

using System.Text;

using System.Web.Hosting;

namespace MantisShrimp.FilePreprocessor.WcfService

{

[ServiceBehavior(InstanceContextMode =

InstanceContextMode.Single,

ConcurrencyMode = ConcurrencyMode.Single)]

public class FilePreprocessorService

: IFilePreprocessorService

{

// Get the physical path to the app\_data

directory used to store the image files.

private string imageFilePath =

HostingEnvironment.MapPath("~/App\_Data/");

// Create a temp file to store the

new images during POST operations.

private string tempFile = Path.GetTempFileName();

private FileProcessor fp;

public FilePreprocessorService()

{

fp = new FileProcessor();

fp.PhysicalPath = imageFilePath;

fp.Init();

}

public RemoteFileInfo DownloadFile(

DownloadRequest request)

{

RemoteFileInfo result = new RemoteFileInfo();

try

{

string filePath =

System.IO.Path.Combine(imageFilePath,

request.FileName);

System.IO.FileInfo fileInfo =

new System.IO.FileInfo(filePath);

// check if exists

if (!fileInfo.Exists)

throw

new System.IO.FileNotFoundException(

"File not found",

request.FileName);

// open stream

System.IO.FileStream stream =

new System.IO.FileStream(filePath,

System.IO.FileMode.Open, System.IO.FileAccess.Read);

// return result

result.FileName = request.FileName;

result.Length = fileInfo.Length;

result.FileByteStream = stream;

}

catch (Exception ex)

{

}

return result;

}

public RemoteFileMetadataInfo UploadFile(

RemoteFileInfo request)

{

string metadata = string.Empty;

FileStream targetStream = null;

Stream sourceStream = request.FileByteStream;

string uploadFolder = imageFilePath;

string filePath = tempFile;

// Path.Combine(uploadFolder, request.FileName);

Bitmap image = new Bitmap(sourceStream);

request.Metadata = fp.Recognise(image);

return new RemoteFileMetadataInfo(request);

}

}

}

Програмна реаліація компоненту контракту контроллеру веб-додатку:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Web;

using System.Web.Hosting;

using System.Web.Mvc;

namespace MantisShrimp.Web.Controllers

{

public class RecogniseController : Controller

{

public ActionResult Index()

{

ViewBag.Message = "Your contact page.";

string imageFilePath =

HostingEnvironment.MapPath("~/Images/");

string imageFileName = "automobile10.png";

string imageFileFullPath = imageFilePath

+ imageFileName;

System.IO.FileInfo fileInfo =

new System.IO.FileInfo(imageFileFullPath);

MantisShrimp.Web

.FilePreprocessorService

.IFilePreprocessorService clientUpload =

new MantisShrimp.Web

.FilePreprocessorService

.FilePreprocessorServiceClient();

MantisShrimp.Web

.FilePreprocessorService

.RemoteFileInfo

uploadRequestInfo =

new MantisShrimp.Web

.FilePreprocessorService

.RemoteFileInfo();

using (System.IO.FileStream stream =

new System.IO.FileStream(imageFileFullPath,

System.IO.FileMode.Open, System.IO.FileAccess.Read))

{

uploadRequestInfo.FileName = imageFileName;

uploadRequestInfo.Length = fileInfo.Length;

uploadRequestInfo.FileByteStream = stream;

var requestMetadataInfo =

clientUpload.UploadFile(uploadRequestInfo);

//clientUpload.UploadFile(stream);

ViewBag.Metadata = requestMetadataInfo.Metadata;

}

ViewBag.ImgUrl = "/Images/" + imageFileName;

return View();

try

{

FileTransferServiceReference.ITransferService

clientDownload =

new TransferServiceClient();

FileTransferServiceReference

.DownloadRequest requestData =

new DownloadRequest();

FileTransferServiceReference

.RemoteFileInfo fileInfo =

new RemoteFileInfo();

requestData.FileName = "codebase.zip";

fileInfo =

clientDownload.DownloadFile(requestData);

Response.BufferOutput = false;

// to prevent buffering

byte[] buffer = new byte[6500];

int bytesRead = 0;

HttpContext.Current.Response.Clear();

HttpContext.Current.Response.ClearHeaders();

HttpContext.Current.Response.ContentType =

"application/octet-stream";

HttpContext.Current.Response.AddHeader(

"Content-Disposition",

"attachment; filename="

+ requestData.FileName);

bytesRead =

fileInfo.FileByteStream.Read(

buffer, 0, buffer.Length);

while (bytesRead > 0)

{

// Verify that the client is connected.

if (Response.IsClientConnected)

{

Response.OutputStream.Write(

buffer, 0, bytesRead);

// Flush the data to the HTML output.

Response.Flush();

buffer = new byte[6500];

bytesRead =

fileInfo.FileByteStream.Read(

buffer, 0, buffer.Length);

}

else

{

bytesRead = -1;

}

}

}

catch (Exception ex)

{

// Trap the error, if any.

System.Web.HttpContext.Current.Response.Write(

"Error : " + ex.Message);

}

finally

{

Response.Flush();

Response.Close();

Response.End();

System.Web.HttpContext

.Current.Response.Close();

}

}

public ActionResult FileUpload(

HttpPostedFileBase[] files)

{

ViewBag.Metadata = "";

foreach (var file in files)

{

string imageFileName = file.FileName;

MantisShrimp.Web

.FilePreprocessorService

.IFilePreprocessorService clientUpload =

new MantisShrimp.Web

.FilePreprocessorService

.FilePreprocessorServiceClient();

MantisShrimp.Web

.FilePreprocessorService

.RemoteFileInfo uploadRequestInfo =

new MantisShrimp.Web

.FilePreprocessorService

.RemoteFileInfo();

uploadRequestInfo.FileName = imageFileName;

uploadRequestInfo.FileByteStream =

file.InputStream;

var requestMetadataInfo =

clientUpload.UploadFile(

uploadRequestInfo);

//clientUpload.UploadFile(stream);

ViewBag.Metadata += "\r\n \r\n \r\n"

+ imageFileName+": "

+ requestMetadataInfo.Metadata;

}

// after successfully uploading redirect the user

return View("Index");

}

}

}

Додаток В

Апробація результатів роботи