РЕФЕРАТ

Пояснювальна записка: 82 с., 34 рис., 1 табл., 12 джерел.

Об'єкт розробки: методи класифікації цифрових зображень за допомогою нейронних мереж.

Мета дипломної роботи: створення нейронної мережі здатної швидко і якісно класифікувати цифрові зображення.

Наукова новизна: удосконалено архітектуру згорткової нейронної мережі, яка за показниками точності та повноти не поступається відомим аналогам, але потребують менше часу на її навчання, швидше класифікують цифрові зображення і потребують менше ресурсів для розгортання і використання.

У вступі наведено стан проблеми та обґрунтована її актуальність.

У першому розділі наведені аналіз значення нейронних мереж у світі, вивчено їх різновидність, а також цілеспрямованість кожного її типу, виконано огляд існуючих рішень та аналогів.

У другому розділі розглянуті основні інструменти та технології для реалізації мети роботи, а також виконаний аналіз архітектурних складових для створення згорткової нейронної мережі, огляд шарів і оптимізаторів для майбутньої реалізації.

У третьому розділі виконано моделювання та навчання трьох моделей нейронних мереж, проведено їх навчання на навчальній виборці цифрових зображень, виконане тестування за найбільш важливими метриками та оцінка робочої здатності мереж.

Практичне значення роботи полягає у створенні власного модулю для класифікації цифрових зображень із використанням нейронних мереж.

Розроблено новий модуль згорткової нейронної мережі для класифікації цифрових зображень, що має відносно малі затрати часу на навчання і в результаті дає точність і повноту класифікації не менше 80%.

Виконано оцінку якості роботи нейронної мережі на даних тестової вибірки рентгенологічних знімків людей здорових і хворих пневмонією.

ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, ЦИФРОВІ ЗОБРАЖЕННЯ, МОВА ПРОГРАМУВАННЯ, PYTHON, КЛАСИФІКАЦІЯ, БІБЛІОТЕКИ TENSORFLOW, KERAS.

ABSTRACT

Explanatory note: 82 p., 34 fig., 1 tab., 12 references.

Object of research: methods for classifying digital images using neural networks.

The objective of research: creating a neural network capable of quickly and qualitatively classifying digital images.

Scientific novelty: improves the architecture of a convolutional neural network, which in terms of accuracy and completeness is not inferior to known analogues, but require less time to learn it, faster classify digital images and require less resources to deploy and use.

In introduction presents the state of the problem and its relevance.

The first section contains an analysis of the value of neural networks in the world, examines their diversity, as well as the purposefulness of each type, examines existing solutions and analogues.

The second section contains the basic tools and technologies for the purpose of the work, as well as analyzes the architectural components to create a convolutional neural network, an overview of layers and optimizers for future implementation.

The third section contains the modeling and training of three models of neural networks, their training was performed on a training sample of digital images, testing was performed on the most important metrics and an assessment of the network performance.

The practical value of the work is to create your own module for classifying digital images using neural networks.

A new convolutional neural network module for digital image classification has been developed, which has relatively low training time, resulting in a classification accuracy of at least 80%.

The quality of work of the neural network on the data of the test samples of radiological images of healthy people and pneumonia patients was performed.

CONVOLUTIONAL NEURAL NETWORK, DIGITAL IMAGES, PROGRAMMING LANGUAGE, PYTHON, CLASSIFICATION, TENSORFLOW LIBRARIE, KERAS FRAMEWORK.

Зміст

ВСТУП 3
РОЗДІЛ 1 5
Аналіз існуючих методів машинного навчання і сучасних готових рішень для даної задачі 5
1.1. Вивчення різновидів нейронних мереж, їх структуру та призначення ... 5
1.1.1. Нейронна мережа прямого поширення - штучний нейрон 5
1.1.2. Мережа радіальних базисних функцій (Radial Basis Function Neural Network) 6
1.1.3. Перцептрон 7
1.1.4. Багатошаровий перцептор 9
1.1.5. Згорткові нейронні мережі 9
1.1.6. Рекурентна нейронна мережа - Довга короткочасна пам'ять 13
1.1.7. Модульна нейронна мережа 14
1.2. Огляд існуючих рішень 14
1.2.1. Neuroph 14
1.2.2. Simbrain 15
1.2.3. MATLAB 16
1.2.4. Encog 18
1.2.5. Існуючі реалізації згортальних нейронних мереж 19
Висновок 19
РОЗДІЛ 2 21
ПРОЕКТНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ НЕЙРОННОЇ МЕРЕЖІ 21
2.1. Огляд вибраних інструментів для реалізації програмного модуля класифікації цифрових зображень 21
2.1.1. Засоби мови програмування Python 21
2.1.2. Фреймворк для створення нейронних мереж Keras 25
2.1.3. Бібліотека matplotlib 26
2.1.4. Бібліотека для роботи з масивами NumPy 27
2.1.5. Огляд фреймворку scikit-learn 27
2.1.6. Бібліотека для роботи з зображенням OpenCV 28

2.1.7. Програмна бібліотека для налаштування машинного навчання TensorFlow 29
2.2. Аналіз шарів для згорткової нейронної мережі 31
2.2.1. Convolutional layer 31
2.2.2. Pooling layer 31
2.2.3. ReLU layer 33
2.2.4. Dropout 33
2.2.5. Batch normalization layer 34
2.2.6. Dense/fully connected layer 36
2.3. Оптимізатори 36
Висновок 37
РОЗДІЛ 3 38
Реалізація та тестування згорткової нейронної мережі 38
3.1. Формальне визначення згорткової нейронної мережі 38
3.2. Підходи до проектування архітектури мережі 40
3.3. Створення програмного модулю 42
3.3.1. Реалізація двох додаткових нейронних мереж для порівняння результатів під час тестування. 44
3.3.2. Процес навчання нейронних мереж 45
3.3.3. Результати навчання 46
3.4. Проведення тестування створеного модуля і порівняння результатів 53
3.4.1. Критерії тестування розпізнавання 54
3.4.2. Тестування 55
3.4.3. Результати 59
Висновок 64
ВИСНОВКИ 65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 66
ДОДАТОК А Відомості матеріалів дипломного проекту 68
ДОДАТОК Б Код програми 69
ДОДАТОК В ВІДГУК 78
ДОДАТОК Г РЕЦЕНЗІЯ 79

ВСТУП

В даний час такі речі як класифікація і автоматичне (машинне) розпізнавання, групування образів, їх опис - важливі завдання у великій кількості інженерних і наукових областей, таких як біологія, фізіологія, медицина, маркетинг, комп'ютерний зір, штучний інтелект. Введемо поняття образу. Образ - протилежність хаосу; це певна сутність, якої може бути дано ім'я.

Існують два класи задач розпізнавання / класифікації:

- розпізнавання з учителем, в якому вхідний образ вважається членом певного класу;
- розпізнавання без вчителя (наприклад, класифікація), в якому образ вважається членом невідомого класу.

Потрібно зауважити, що завдання розпізнавання вважається завданням класифікації, в якій класи або задаються дизайнером системи (в розпізнаванні з учителем), або будуються на підставі схожості образів (в розпізнаванні без вчителя).

Однією з найбільш важких завдань розпізнавання образів є задача розпізнавання (класифікації) зображень. Це завдання виникає в таких областях як розпізнавання рукописного тексту, дорожніх знаків, номерів автомобілів, стерео і мультизору. Відмінною особливістю даного завдання є величезна розмірність вхідного простору - що веде до ускладнення розпізнавальних і обчислювальним труднощів. Багато підходів до розпізнавання зображень пропускають вхідні дані через фільтр, що проектує вхідний вектор на простір істотно меншої розмірності, після чого розбивають проміжні вектори на класи за допомогою стандартних розпізнавальців.

Одним з таких класифікаторів є нейронні мережі, одиничний елемент яких, нейрон, емулює роботу біологічного нейрона. Класичні нейронні мережі є рішенням для завдання розпізнавання з учителем. Незважаючи на складність

отримання знань з нейросетевой системи, вони успішно застосовуються для численних завдань класифікації, управління, прогнозування.

Сучасним підходом до розпізнавання зображень є згорткові нейронні мережі. Вони мають велику кількість шарів, в порівнянні з класичним багатошаровим персептроном. За рахунок спільних ваг, які використовуються відразу декількома нейронами в кожному шарі, вдається знизити загальну кількість учнів параметрів мережі і прискорити навчання. Також, на відміну від багатошарового персептрона, згорткові мережі сприйнятливі до топології вхідного зображення.

Дана робота присвячена реалізації системи, що дозволяють будувати нейромережеві розпізнавачі для різних завдань розпізнавання зображень, але зокрема вона буде будуватися і тестуватися на даних рентгенологічних знімків, і на виході ця нейронна мережа повинна буде класифікувати хворих і здорових людей за їх рентгенологічним знімком. Нейронні мережі, які використовуються в роботі, базуються на шарах згортки і субдискретізація. Проведено огляд алгоритмів, що прискорюють навчання нейронних мереж, і алгоритмів, що підвищують якість розпізнавання.

РОЗДІЛ 1

Аналіз існуючих методів машинного навчання і сучасних готових рішень для даної задачі

1.1. Вивчення різновидів нейронних мереж, їх структуру та призначення

Існує багато видів штучних нейронних мереж, кожна зі своїми унікальними особливостями. Розглянемо найбільш важливі і поширені типи нейронних мереж.

1.1.1. Нейронна мережа прямого поширення - штучний нейрон

Це один з найпростіших видів штучних нейронних мереж. У цьому типі нейронної мережі дані проходять через різні вхідні вузли до тих пір, поки вони не досягнуть вихідного вузла[1].

Із цього слідує, що дані рухаються лише в одному напрямку від першого рівня вгору, поки не доходять до вихідного вузла. Це також відоме як передня розповсюджена хвиля, яка зазвичай досягається за допомогою класифікаційної функції активації.

На відміну від більш складних типів нейронних мереж, у ній немає зворотного розповсюдження і дані рухаються лише в одному напрямку. Нейронна мережа може мати один шар або вона може мати приховані шари, що дозволяють обчислювати суму продуктів входів та їх ваги, які потім подаються на вихід.

Нейронні мережі прямого поширення застосовуються в технологіях для розпізнавання обличчя та комп’ютерного зору, а також дана мережа обладнана для обробки даних, що містять багато шуму. Ця нейронна мережа досить проста в обслуговуванні.

Нижче ми можемо бачити приклад нейронної мережі прямого поширення - штучний нейрон (Feedforward Neural Network – Artificial Neuron)

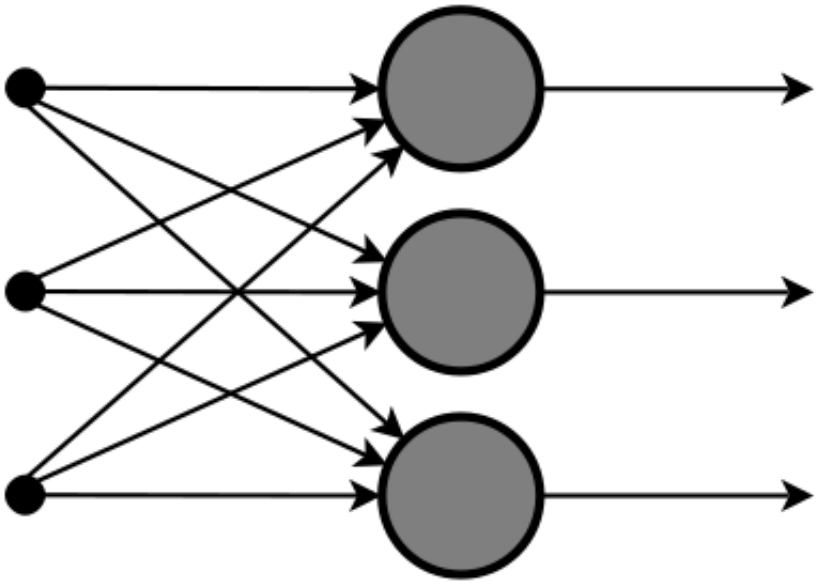


Рис 1.1. Нейронна мережа прямого поширення

1.1.2. Мережа радіальних базисних функцій (Radial Basis Function Neural Network)

Мережа радіальних базисних функцій (RBF) - це поширений тип штучної нейронної мережі для задач наближення функції. Функціональні мережі радіальної бази відрізняються від інших нейронних мереж завдяки їх універсальному наближенню та більшій швидкості навчання. Мережа RBF - це тип нейронної мережі подачі вперед, що складається з трьох шарів, а саме вхідного шару, прихованого рівня та вихідного шару, кожен з цих шарів має різні завдання.

Навчання моделі RBF припиняється після того, як обчислена помилка досягне бажаних значень (наприклад 0,01) або вже завершена кількість ітерацій тренувань (наприклад 500). Вибирається мережа RBF з певною кількістю вузлів (наприклад 10) у своєму прихованому шарі. Функція Гаусса використовується як функція передачі даних в обчислювальних одиницях. Залежно від конкретного випадку, як правило, спостерігається, що мережі RBF потрібно менше часу для завершення навчання.

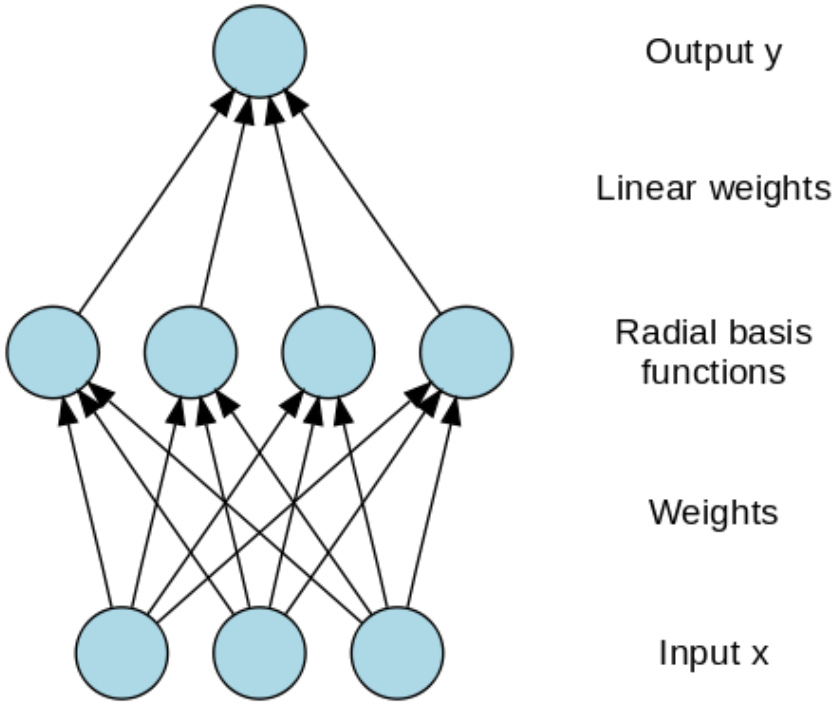


Рис 1.2. Нейронна мережа радіальних базисних функцій

Нейронна мережа радіальних базисних функцій широко функціонує в системах відновлення енергії. В останні десятиліття енергосистеми стають більшими та складнішими, що збільшує ризик затемнення. Ця нейронна мережа використовується в системах відновлення енергії з метою відновлення живлення в найкоротші терміни.

1.1.3. Перцептрон

Розенблатт сконструював перший у 1958 р. нейронний комп'ютер, перцептрон, намагаючись імітувати навчання людини. Дендрити нейрона моделюються за вагами, що множать на вхідні значення. Крім того, додається значення зміщення для моделювання необхідного потенціалу активації нейрона. Потім множинні значення та зміщення підсумовуються в тілі клітини і пропускаються через функцію активації для отримання виходу, що представляє швидкість випалу в нейроні. Описана архітектура показана на малюнку нижче.

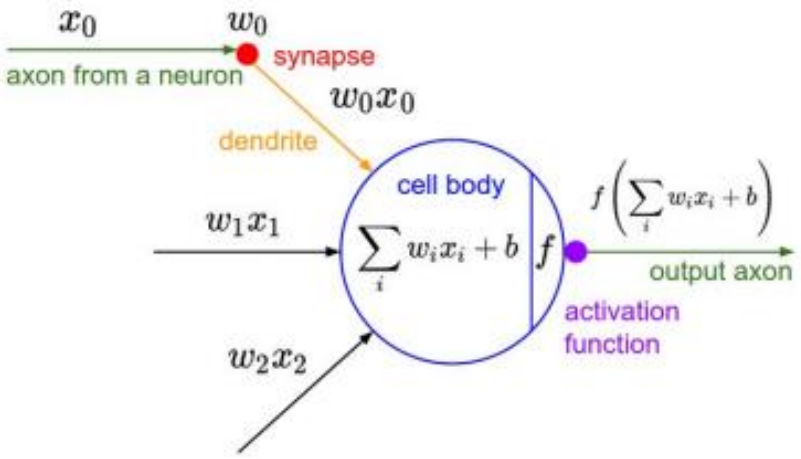


Рис. 1.3. Модель персептрона з вагами w_i, входами x_i та зсувом b.

Перцептрон - це лінійний класифікатор, визначений вагами wi, зміщенням b і функцією активації f. Можна об'єднати зміщення з вагами за допомогою однорідних координат, тобто покласти зміщення в нижній частині вагового вектора і додати постійну 1 до вхідного вектора x. Можуть використовуватися різні функції активації, але оригінальний перцептрон використовує функцію крокової важкої сторони, що призводить до двійкового виводу. Тому перцептрон є лінійним класифікатором, і його ваги визначають гіперплан як лінійну межу рішення між класами. Таким чином, його представницька сила обмежена; неможливо моделювати, наприклад, функцію XOR з використанням перцептрона, оскільки жодна лінія не може бути проведена для розділення класів, як показано на рисунку нижче.

$$f\left(\sum_i w_i x_i + b\right)$$

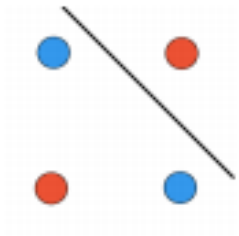


Рис. 1.4. Функція XOR створює нелінійно відокремлений набір. Два класи червоний і синій не можна розділити однією прямою лінією.

1.1.4. Багатошаровий перцептор

Багатошаровий перцептрон має три або більше шарів. Він використовується для класифікації даних, які неможливо розділити лінійно. Це тип штучної нейронної мережі, яка повністю пов'язана. Це відбувається тому, що кожен окремий вузол у шарі з'єднаний з кожним вузлом у наступному шарі.

Багатошаровий перцептрон використовує нелінійну функцію активації (головним чином гіперболічну дотичну або логістичну функцію). Ось як виглядає багатошаровий перцептрон.

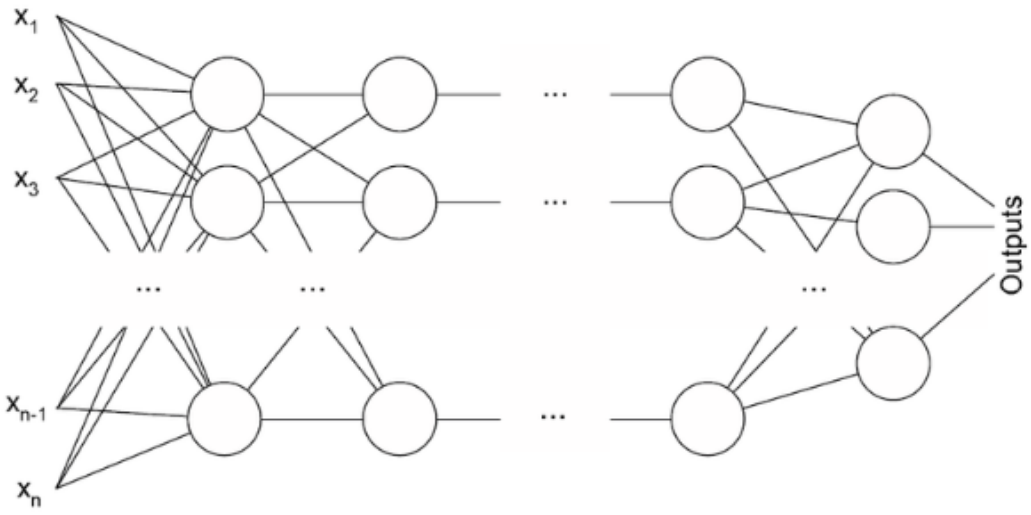


Рис 1.5. Багатошаровий перцептрон

Цей тип нейронної мережі широко застосовується в технологіях розпізнавання мовлення та технологіях машинного перекладу.

1.1.5. Згорткові нейронні мережі

Згорткові нейронні мережі (CNN) - це підгрупа алгоритмів глибокого навчання, які зосереджуються на завданнях комп'ютерного зору та обробки зображень. Системні мережі CNN надихнуті тим, як мозок обробляє візуальну інформацію. Hubel і Wiesel були першими, хто запропонував глибокі моделі, схожі на структуру зорової котячої коти. Ці моделі ототожнювали прості клітини з локальними сприйнятливими полями, схожими на фільтри або ядра, і складні комірки, схожі на об'єднання шарів. Перший CNN був введений в неокогнітроні Фукусіми [5]. Пізніше CNN були покращені LeNet-5 Yann

LeCun, який застосував стохастичні градієнти на основі градієнта для розпізнавання документів і був дуже успішним для розпізнавання рукописних завдань. Основним недоліком у дослідженні та розробці CNN в 1990-х і 2000-х роках була необхідна обчислювальна потужність для їх широкого застосування до зображень з високою роздільною здатністю. Однак це змінилося з 2010 року. Існує три причини, завдяки чому глибокі мережі стали успішними:

- більша обчислювальна здатність завдяки закону Мура, особливо це стосується сьогоднішніх графічних процесорів;
- більше навчальних даних;
- нові та кращі алгоритми.

Зі збільшенням обчислювальної потужності дослідники описали нові способи більш ефективного тренування згорткових нейронних мереж, що дозволяло створювати більш глибокі мережі.

Останнім часом продуктивність значно збільшилась завдяки безлічі баз даних зображень, наближаючись або навіть перемагаючи продуктивність людини, наприклад щодо розпізнавання цифр (<0,25 відсотка) [6] та багатьох інших завдань з розпізнавання шаблонів, найбільш важливих проблем візуальної класифікації та інших.

Загалом, CNN дуже добре класифікують такі об'єкти, як конкретні породи собак та котів на основі дрібнозернистих деталей, тоді як люди мають проблеми з цим. Недоліком глибокого навчання є те, що для розпізнавання об'єктів у реалістичних умовах потрібні дуже великі набори даних для навчання. В даний час найкращі CNN борються з невеликими або плоскими об'єктами, або спотвореними цифровими фільтрами. Недавнє дослідження виявило відмінності того, як глибокі нейронні мережі та людина розпізнають об'єкти. У цьому дослідженні зображення кодуються таким чином, який людина не в змозі розпізнати, але глибокі нейронні мережі виявляли правильний клас об'єкта майже зі 100% точністю [10].

Для тренування дуже складних згорткових нейромереж на великих наборах даних, останнім часом, з’явилася тенденція до збільшення кількості шарів та розмірів шарів при використанні відсіву [34] для вирішення проблеми надмірного оснащення. Krizhevsky та Szegedy, творці двох найбільш відомих мереж, що використовуються в дослідженнях, а саме AlexNet [6] та GoogLeNet [7], підкреслюють, наскільки важливо використовувати високий коефіцієнт випадання під час тренувань.

Ще одна тенденція - зробити мережі дуже глибокими, тобто мережі мають багато шарів. Zeiler експериментував у з глибиною CNN і дійшов висновку, що продуктивність мережі сильно залежить від кількості шарів. Виміряна продуктивність значно знижується, навіть якщо видаляється один згортковий шар. Він зазначає, що глибина мережі важливіша, ніж будь-яка інша архітектурна складова мережі.

Отже, вибір базової архітектури є ключем до ефективності мережі. Це також можна побачити на GoogLeNet, який був опублікований у 2014 році та має глибину 22 шари, не рахуючи шарів об’єднання. У 2012 році AlexNet розпочав ажіотаж навколо прогресу в глибокому навчанні, зокрема, з мережею, що складається з 8 шарів. Хоча вона досягла найсучасніших результатів, коли вона була опублікована, відтепер багато інших мереж перевершили результативність.

Обмежуючим фактором розміру CNN є об'єм пам'яті, наявної в поточних одиницях обробки графічних процесорів (GPU), і переносний час на навчання. До того, як розпочали широко використовувати GPU для навчання глибоких мереж, загальний час навчання процесів був більшим у 9 разів. Бібліотека cuDNN від Nvidia, яка оптимізована для швидкої обробки зображень та ефективних операцій згортання в графічних процесорах, додатково прискорює час навчання з майже в 9 разів. Таким чином, сучасні графічні процесори з найновішими встановленими бібліотеками CUDA та cuDNN прискорюю час обчислення в 17 разів порівняно з сьогоднішніми процесорами. Залежно від кількості параметрів мережі та розміру набору даних, навчання згорткової

нейронної мережі на графічному процесорі з випадково ініційованими вагами може іноді займати кілька днів або навіть і тижнів.

Згорткова нейронна мережа (CNN) використовує варіацію багатошарових перцептронів. CNN містить один або декілька згорткових шарів. Ці шари можуть бути повністю пов'язані між собою або об'єднані.

Перш ніж передати результат наступному шару, згортковий шар використовує згорнуту операцію на вході. Завдяки цій згортковій роботі мережа може бути набагато глибшою, але зі значно меншими параметрами.

Завдяки цій здатності конвертовані нейронні мережі показують дуже ефективні результати в розпізнаванні зображень і відео, природній обробці мови та системах рекомендування.

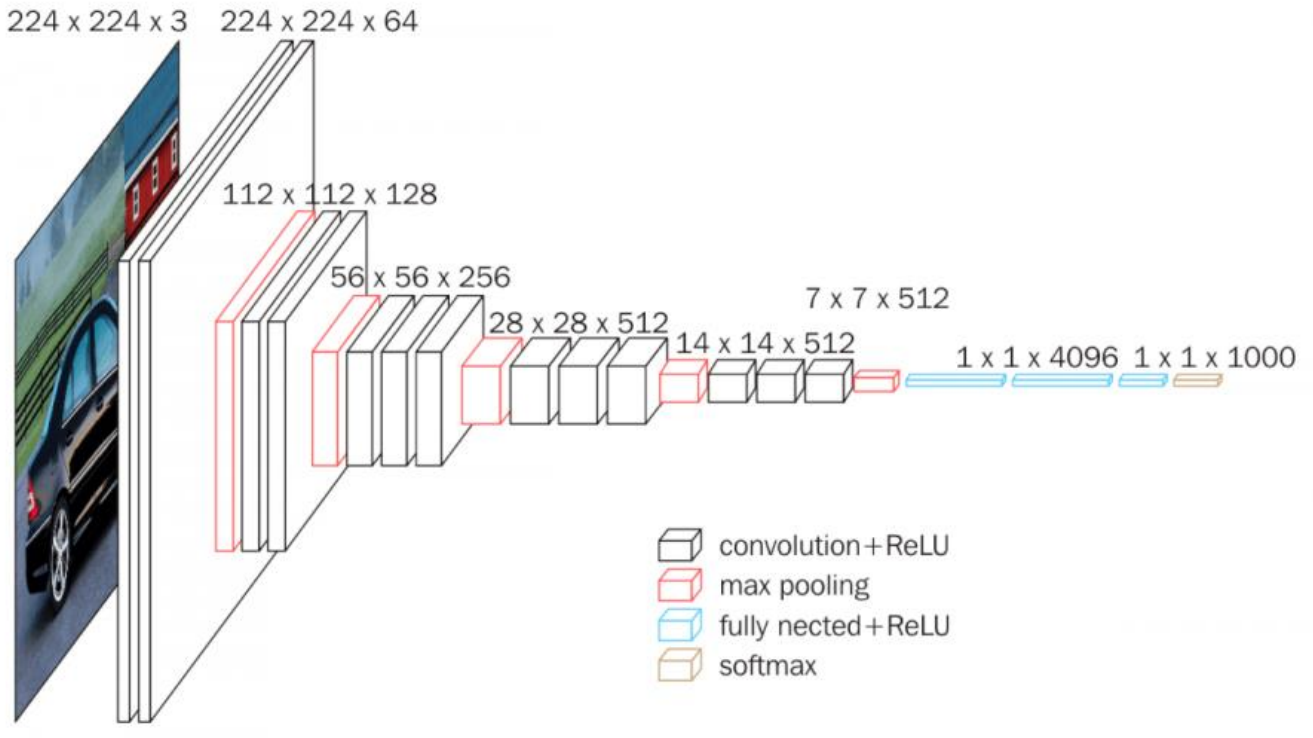


Рис 1.6. Архітектура згорткової нейронної мережі

Згорткові нейронні мережі показують чудові результати в семантичному розборі та виявленні парафрази. Вони також застосовуються при обробці сигналів та класифікації зображень, де відмінно себе проявляють.

1.1.6. Рекурентна нейронна мережа - Довга короткочасна пам'ять

Рекурентна нейронна мережа - це тип штучної нейронної мережі, в якій вихід певного шару зберігається і подається назад на вхід. Це допомагає передбачити результат шару.

Перший шар формується так само, як і в мережі подачі. Тобто з добутком суми ваг та ознак. Однак у наступних шарах починається повторюваний процес нейронної мережі.

Від кожного часового кроку до наступного кожен вузол запам'ятовуватиме інформацію, яку він мав у попередньому часовому кроці. Іншими словами, кожен вузол виконує функцію комірки пам'яті під час обчислення та виконання операцій. Нейронна мережа починається з переднього розповсюдження, як зазвичай, але запам'ятовує інформацію, яку, можливо, потрібно буде використовувати згодом.

Якщо прогноз неправильний, система самостійно навчається та працює над правильним прогнозуванням під час розмноження. Цей тип нейронної мережі дуже ефективний в технології перетворення тексту в мовлення. Ось як виглядає періодична нейронна мережа.

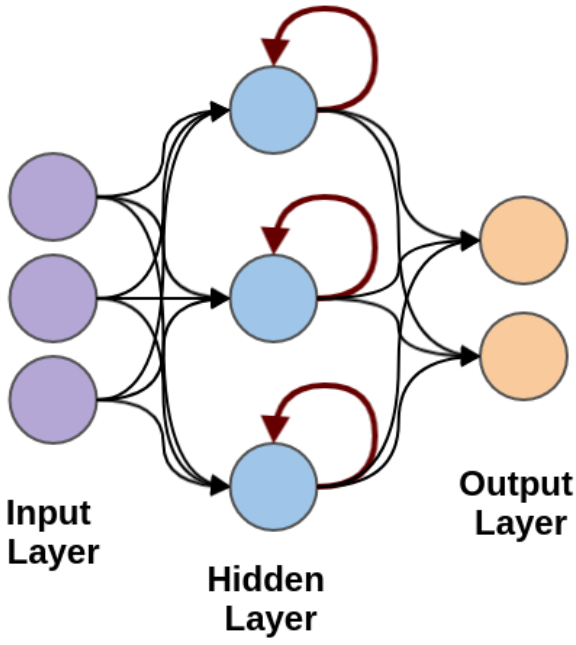


Рис 1.7. Рекурентна нейронна мережа

1.1.7. Модульна нейронна мережа

Модульна нейронна мережа має ряд різних мереж, які функціонують незалежно та виконують підзадачі. Різні мережі насправді не взаємодіють між собою та не передають сигнал під час обчислення. Вони працюють незалежно щодо досягнення результату.

В результаті великий і складний обчислювальний процес можна зробити значно швидше, розбивши його на незалежні компоненти. Швидкість обчислення зростає, оскільки мережі не взаємодіють і навіть не з'єднуються між собою. Ось наочне зображення модульної нейронної мережі.

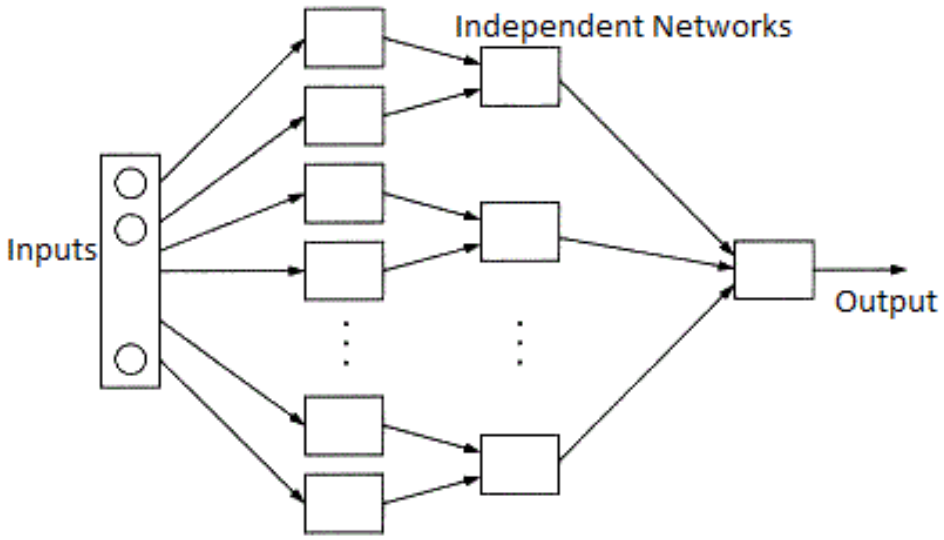


Рис 1.8. Модульна нейронна мережа

1.2. Огляд існуючих рішень

На даний момент існує велика кількість алгоритмів розпізнавання, систем для побудови нейромережевих (і не тільки) розпізнавателів.

1.2.1. Neuroph

Neuroph - система для нейросетевого розпізнавання, написана на мові Java [13, 15]. Складається з Java API, що включає в себе основні класи, класи-утиліти і реалізацію конкретних типів нейронних мереж. Також включає в себе середовище Neuroph Studio, реалізовану на платформі NetBeans.

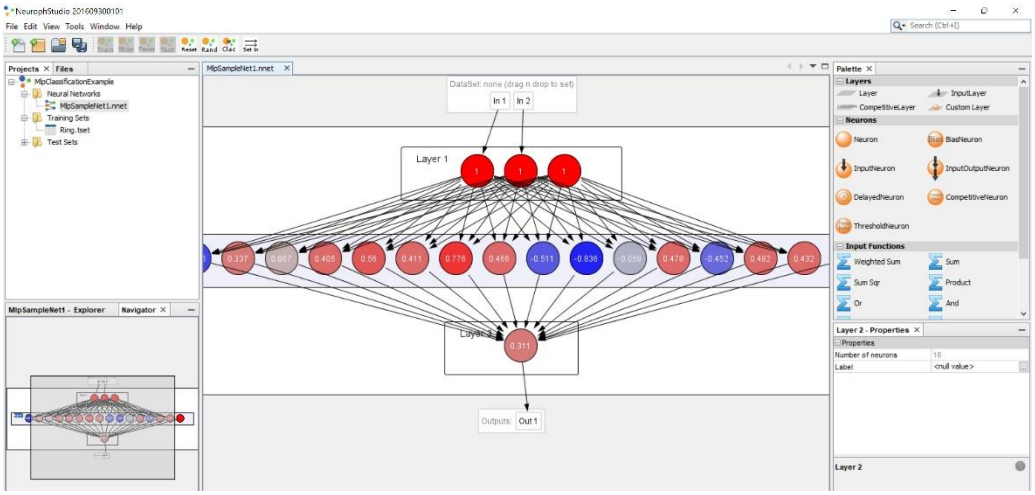


Рис 1.9. Інтерфейс програми Neuroph

У середовищі реалізовані наступні нейромережеві архітектури:

- Адалайн
- Персептрон
- Багатошаровий персептрон з алгоритмом зворотного поширення помилки, моментом
- мережа Хопфілда
- Двостороння асоціативна пам'ять
- мережа Кохонена
- мережа Хебба
- RBF нейронна мережа

У систему включені приклади використання мереж для прогнозування цін на фінансових ринках, приклади класифікації тварин і інші. У мережу включена подержка розпізнавання зображень за допомогою багатошарового перцептрона з алгоритмом зворотного поширення помилки.

У мережі немає реалізації згортальних нейронних мереж.

1.2.2. Simbrain

Simbrain - кроссплатформенная система, написана на мові Java [13, 14]. Система орієнтована на візуальність і простоту. Включена реалізація двовимірного світу з різними об'єктами, для тестування нейронних мереж, які використовуються для управління.

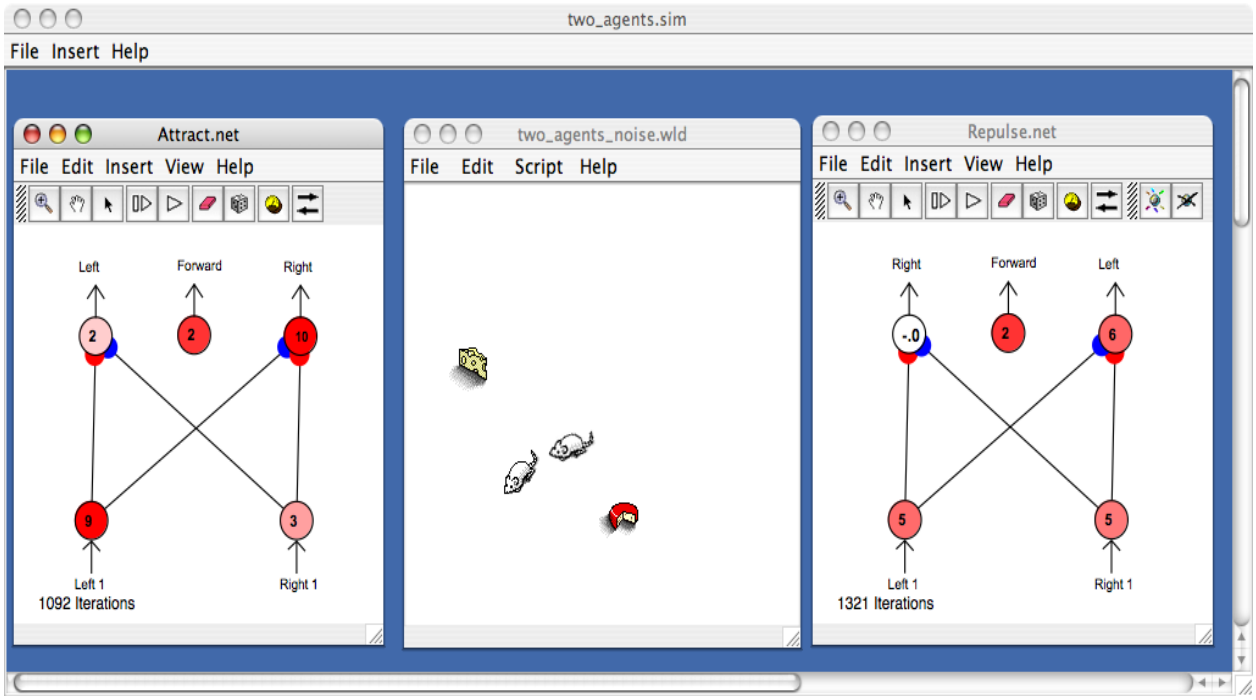


Рис 1.10. Інтерфейс програми Simbrain

В системі реалізовано велику кількість різних варіантів нейронів і типів мереж, присутній навчання без учителя.

Створення та редагування нейронних мереж в середовищі Simbrain відбувається візуально. У разі візуального побудови, користувач може бачити процес зміни ваг в мережі, передачі сигналу. Можливо відстежувати і алгоритм зворотного поширення помилки. Але користувач також може скористатися Java API для конструювання власних завдань і додавання необхідних технологій в систему.

В системі немає реалізації згортальних нейронних мереж, призначених для розпізнавання зображень. Введення вибірок в мережу здійснюється вручну, що не завжди зручно.

1.2.3. MATLAB

Deep Learning Toolbox (раніше Neural Network Toolbox) - це пакет розширення MATLAB, що містить засоби для проектування, моделювання, розробки та візуалізації нейронних мереж.

Пакет забезпечує всебічну підтримку типових нейромережевих парадигм і має відкриту модульну архітектуру. Пакет містить функції

командного рядка і графічний інтерфейс користувача для швидкого покрокового створення нейромереж.

Крім цього Deep Learning Toolbox забезпечує підтримку Simulink, що дозволяє моделювати нейромережі і створювати блоки на основі розроблених нейромережевих структур.

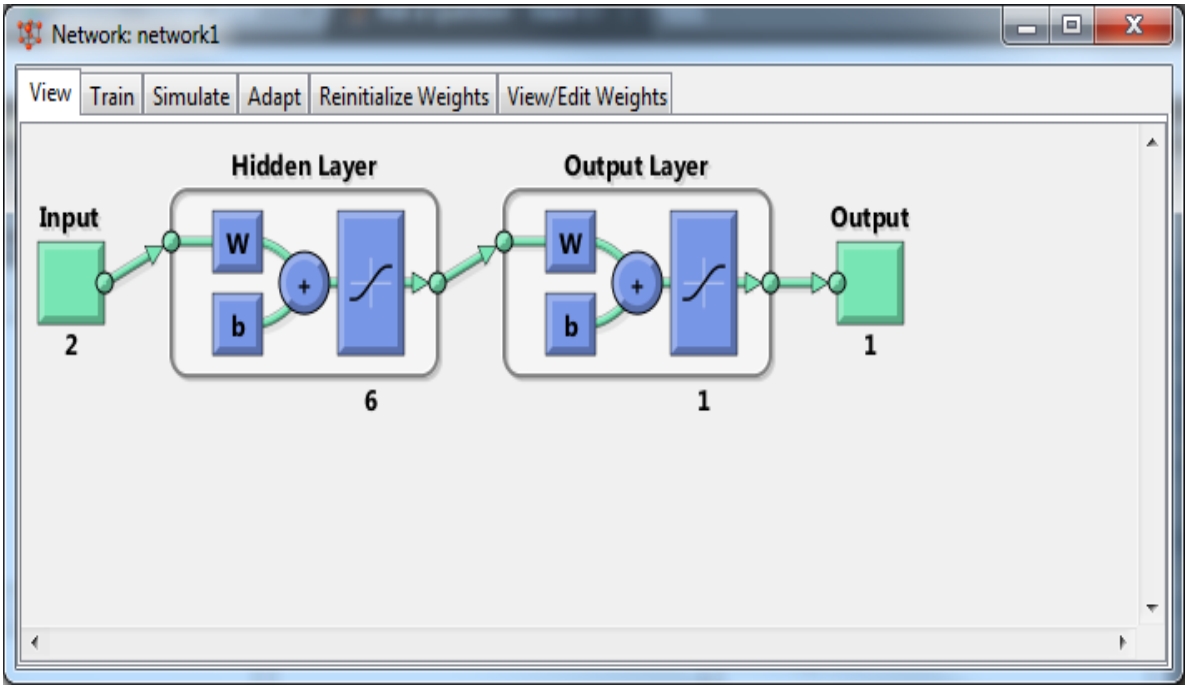


Рис 1.11. Інтерфейс модуля Deep Learning Toolbox в MATLAB

Ключові характеристики модуля:

- Графічний інтерфейс користувача для покрокового створення, навчання і імітаційного моделювання нейронних мереж
- Підтримка найбільш поширених керованих і некерованих мережевих структур
- Повний перелік навчальних і тестуючих функцій
- Динамічні алгоритми навчання мереж, що включають тимчасову затримку, нелінійну авторегресії (NARX), ланцюгові і настроюються динамічні структури
- Блоки Simulink для створення нейронних мереж і розвинених блоків для систем контролю
- Автоматична генерація блоків Simulink з об'єктів нейронної мережі

- Модульне подання мережі, що дозволяє створювати необмежену кількість вхідних шарів і об'єднаних мереж, а також графічне представлення архітектури мережі
- Функції попередньої і пост обробки і блоки Simulink для поліпшення процесу навчання і оцінки продуктивності мережі
- Візуалізація топології і процесу навчання нейронної мережі

1.2.4. Encog

Система Encog є найбільш повною з розглянутих. Вона включає в себе велику кількість ефективно написаних Java класів. На відміну від інших систем, в ній реалізовані не тільки нейромережеві алгоритми розпізнавання, але і кластеризація, генетичні алгоритми, приховані марковскі моделі, байєсовські мережі та ін.

Серед підтримуваних архітектур нейронних мереж такі, як адалайн, машина Больцмана, нейронні мережі зворотного поширення, рекурентні мережі Ельмана, рекурентні мережі Джордана, самоорганізуються карти Кохонена і т.д. Доступні наступні функції активації нейронів: біполярна, змагальна, функція Еліотта, функція Гаусса, гіперболічний тангенс, лінійна, синусоїдальна, сигмоїда.

У порівнянні з іншими система, Encog написана дуже ефективно, підтримується паралельне функціонування мереж на машинах з декількома процесорами. Але Encog так само не підтримує розпізнавання зображень - немає реалізації згортальних нейронних мереж.

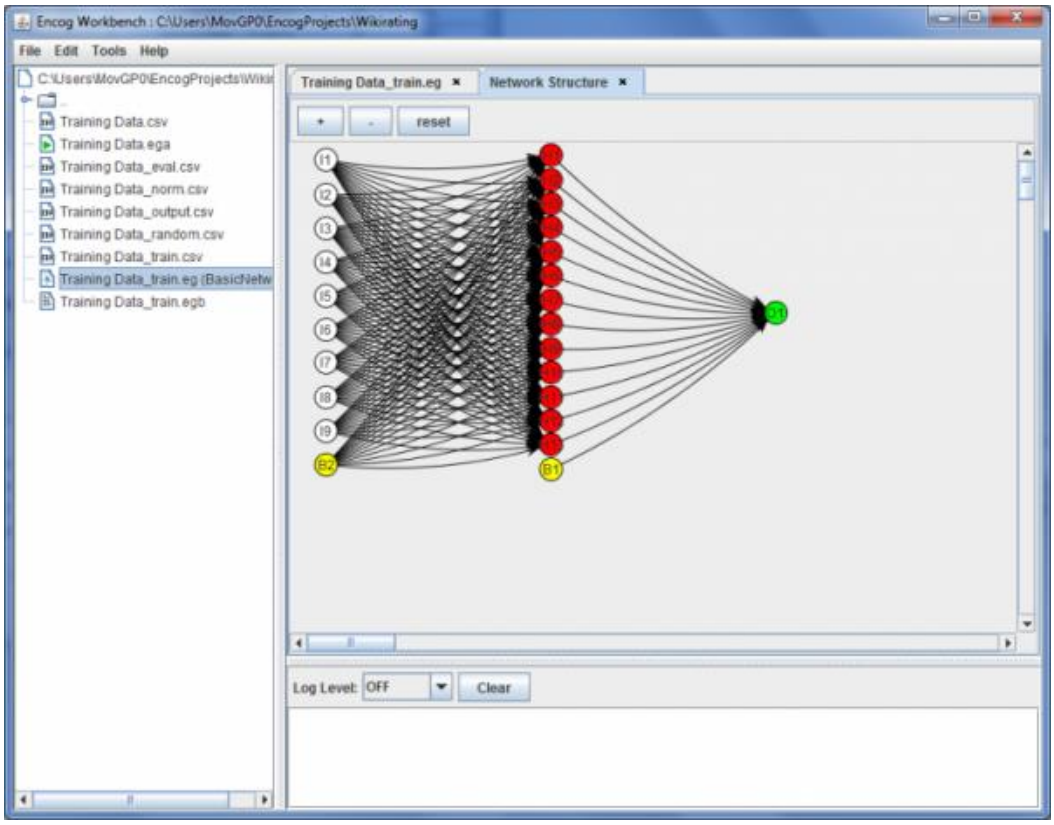


Рис 1.12. Інтерфейс системи Encog

1.2.5. Існуючі реалізації згортальних нейронних мереж

В інтернеті є кілька вільних реалізацій згортальних мереж. Але всі вони написані у вигляді додатку для одного завдання. Реалізації не дозволяють додавати нові архітектури мереж, змінювати характеристики навчання, оптимізувати структуру мережі.

Висновок

Розглянувши велику кількість систем для розпізнавання образів, був зроблений висновок, що всі вони не підходять для реалізації поставлених завдань.

Системи широкого профілю, що працюють з різними завданнями і включають в себе велику кількість доступних архітектур нейронних мереж, не можуть безпосередньо застосовуватися до задачі розпізнавання зображень. Одним із найголовніших факторів мого висновку стало те, що вони не включають в себе реалізацію згортальних мереж - шарів згортки і субдіскретізація, а саме такого типу нейронні мережі, як з’ясувалося, найбільше підходять для створення модулю класифікації зображень.

Що є до відомих реалізацій згортальних мереж, то вони не можуть бути використані як системи, бо не володіють достатньою гнучкістю. У більшості випадків, вони написані для одного-двох обраних завдань.

Прийняте рішення про постановку завдання щодо детальнішого дослідження підходів для розпізнавання та класифікації зображень методом нейромереж. Після вивчення, реалізувати нейромережевий модуль для розпізнавання і класифікації рентгенологічних зображень грудної клітини людей на предмет наявності в пацієнта такого захворювання як пневмонія. При цьому намагатися побудувати таку нейронну мережу, яка с великою долею імовірності зможе доволі швидко, раціональні і найголовніше правдиво класифікувати цифрові зображення.

РОЗДІЛ 2

ПРОЕКТНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ НЕЙРОННОЇ МЕРЕЖІ

2.1. Огляд вибраних інструментів для реалізації програмного модуля класифікації цифрових зображень

У цьому розділі розглянуто особливості обраних для реалізації програмного модуля інструментів і фреймворків.

Для навчання моделей використовується фреймворки scikit-learn і Keras для мови Python версії 3.7. Вибір обумовлений наявністю великої кількості реалізованих алгоритмів класифікації, багатим набором методів для налаштування нейронної мережі і зручним API, який дозволяє додавати нові модулі і вбудовувати їх в стандартний процес навчання.

2.1.1. Засоби мови програмування Python

Проект виконаний використовуючи мову програмування Python з використанням популярного фреймворку для налаштування нейронних мереж keras. Вибір зумовлений перевагами як самої мови так і її фреймворку. Ці інструменти набирають популярність у світі Deep Learning, що в свою чергу провокує появу великої кількості готових рішень.

Python - високорівнева мова програмування, орієнтована на підвищення продуктивності розробника і читання коду. Синтаксис ядра Python мінімалістичний. Стандартна бібліотека включає значний обсяг корисних функцій. Python підтримує декілька парадигм програмування, в тому числі структурне, функціональне, об'єктно-орієнтоване, імперативне і аспектно-орієнтоване. До основних властивостей мови належить динамічна типізація, автоматичне звільнення пам’яті, зручні стандартні типи даних, обробка виключень. Код в Python зазвичай збирається у функції та класи, які потім можна об'єднати в модулі, які в свою чергу вже при бажанні об'єднуються в пакети.

Однією з переваг цієї мови програмування є читабельність програмного коду. Ця ознака для більшості спеціалістів є основною при виборі інструменту розробки. Вона полягає в легкості читання, ясності і більш високій якості, що відрізняють Python від інших інструментів в світі мов сценаріїв. Програмний код на мові Python читається легше. Це обумовлено тим, що синтаксис побудований на відступах. В програмуванні вважається хорошим тоном писати код програми, використовуючи форматування відступами, це дозволяє краще орієнтуватися в програмі, швидше і легше її оптимізувати. Python зручний тим, що синтаксис побудований на цьому не писаному правилі програмістів а також не включає в себе зайвих синтаксичних символів.

Це означає, що багаторазове його використання і обслуговування виконується набагато простіше, ніж використання програмного коду на інших мовах сценаріїв. Автор Python Гвідо ван Россум, стверджував, що програмний код читається довше і частіше ніж пишеться, тому не дивно що принцип DRY (Don't Repeat Yourself) є одним із парадигм цієї мови.

З вище описаного випливає ще один вагомий плюс Python - висока швидкість розробки. У порівнянні з компілюючими або строго типізований мовами, такими як C, C++ і Java, Python у багато разів підвищує продуктивність розробника. За загальним обсягом програмний код на мові Python часто становить лише третину або навіть п'яту частину еквівалентного програмного коду на мові C ++ або Java. Це означає менший обсяг введення з клавіатури, менша кількість часу на налагодження і менший обсяг трудовитрат на підтримку та оптимізацію продукту. Окрім того, програми, створені на мові Python, запускаються без компіляції, минаючи тривалі етапи компіляції і зв'язування, необхідні в деяких інших мовах програмування, що ще більше збільшує продуктивність праці програміста.

Велика частина програм на мові Python виконується без змін на всіх основних платформах. Перенесення програмного коду з операційної системи Linux в Windows зазвичай полягає в простому копіюванні файлів програм з однієї машини на іншу, достатньо встановити в систему один із

інтерпретаторів Python. Більш того, Python надає масу можливостей по створенню переносних графічних інтерфейсів, програм доступу до баз даних, веб-додатків і багатьох інших типів програм. Це робить його зручним інструментом для розробки кросплатформних програм.

У складі Python поставляється велика кількість зібраних і переносних функціональних можливостей, відомих як стандартна бібліотека. Ця бібліотека надає масу можливостей, потрібних в створенні прикладних програмах, вона дає програмісту смогу виконувати пошук починаючи від звичайного тексту у шаблоні і закінчуючи мережевими функціями. Python дозволяє розширювати свій функціонал як за рахунок ваших самостійно написаних бібліотек, так і за рахунок тих, які були створені сторонніми розробниками. З числа сторонніх розробок можна назвати інструменти створення веб-сайтів, програмування математичних обчислень, розробку ігрових програм і багато іншого. Наприклад, розширення NumPy, що часто використовується розробниками, позиціонується як вільний і більш потужний еквівалент системи програмування математичних обчислень Mathlab.

Інтеграція компонентів. Сценарії Python легко можуть взаємодіяти з іншими частинами програми завдяки різним механізмам інтеграції. Ця інтеграція дозволяє використовувати Python для налаштування і розширення функціональних можливостей програмних продуктів. На сьогоднішній день програмний код на мові Python має можливість викликати функції з бібліотек на мові C / C ++, сам викликатися із побічних програм, що написані на мові C / C ++, інтегруватися з програмними компонентами на мові Java, взаємодіяти з такими платформами, як COM і .NET , і проводити обмін даними через послідовний порт або через мережу за допомогою таких протоколів, як SOAP, XML-RPC.

Марк Лутц в своїй книзі "Изучаем Python. Четвертое издание" пише: "Після 17 років роботи з мовою Python і 12 років викладання єдиний недолік, який мені вдалося виявити, - це швидкість виконання програм, яка не завжди

може бути такою ж високою, як у програм, написаних на компілює мовах програмування, таких як C або C ++ ".

Класичний Python, як і багато інших інтерпретовані мови, які не застосовують, наприклад, JIT-компілятори, мають загальний недолік - порівняно не високу швидкість виконання програм. Збереження байт-коду (розширення .pyc і, до версії 3.5, .pyo) дозволяє інтерпретатору не витрачати зайвий час на перекомпіляцію коду модулів при кожному старті програми.

Також, використовуючи можливість інтеграції компонентів, складні частини програми, які потребують високої швидкості виконання можуть бути виконані на інших мовах програмування. Це недолік який компенсується іншими перевагами Python, зокрема:

- проста, універсальна і інтуїтивно зрозуміла мову програмування;
- малі витрати часу на написання коду;
- простий синтаксис;
- модульність (є можливість використовувати модуль коду в інших програмах);
- велика кількість різноманітних бібліотек і додатків, які спрощують роботу з даними;
- доступне програмне забезпечення;
- кросплатформеність (програма працює незалежно від операційної системи, на якій була запущена).

Все вище сказане робить Python ідеальним інструментом вирішення широкого спектру проблем. Висока швидкість розробки є вагомим плюсом при виборі мови програмування у бізнес-рішеннях, адже не потребує великої кількості ресурсів. Цьому також сприяє різноманіття готових програмних рішень з відкритим кодом. Розробники з усього світу щоденно доповнюють свої репозиторії з унікальними алгоритмами мови Python.

Показовим прикладом актуальності використання розглянутої мови програмування є такі відомі компанії, які часто використовують Python, як Apple, NASA, Google і інші.

Для того, щоб працювати з цим середовищем програмування необхідно розглянути фреймворки і бібліотеки, які можуть знадобитися в подальшому.

2.1.2. Фреймворк для створення нейронних мереж Keras

Keras - API нейронних мереж високого рівня, написаний на Python і здатний працювати над TensorFlow, CNTK або Theano. Він був розроблений з акцентом на швидке експериментування. Уміння переходити від ідеї до результату з найменшою можливою затримкою є ключовим для хорошого дослідження.

Спочатку Keras розроблявся для дослідників з метою дати їм можливість швидко проводити експерименти.

Keras володіє наступними ключовими характеристиками:

- дозволяє виконувати один і той же код на CPU або GPU;
- має дружній API, що спрощує розробку прототипів моделей глибокого навчання;
- включає в себе вбудовану підтримку згортальних мереж (для розпізнавання образів), рекурентних мереж (для обробки послідовностей) і всіляких їх комбінацій;
- включає в себе підтримку довільних мережевих архітектур: моделей з множинними входами або виходами, спільне використання шарів, спільне використання моделей і т. д.

Це означає, що Keras підходить для створення практично будь-яких моделей глибокого навчання.

Фреймворк Keras поширюється на умовах вільної ліцензії і може безкоштовно використовуватися в комерційних проектах. Він сумісний з будь-якими версіями Python, від 2.7 до 3.8. Налічується понад 200 000 користувачів Keras, починаючи з академічних дослідників і інженерів в стартапи і великих компаніях і закінчуючи аспірантами і любителями. Keras використовується в

Google, Netflix, Uber, CERN, Yelp, Square і сотнях стартапів, які вирішують широке коло завдань.

Також, Keras є бібліотекою рівня моделі, яка надає високорівневі будівельні блоки для конструювання моделей глибокого навчання. Вона не реалізує низькорівневі операції, такі як маніпуляції з тензорами і диференціювання, - для цього використовується спеціалізована і оптимізована бібліотека підтримки тензорів. При цьому Keras не покладається на якусь одну бібліотеку підтримки тензорів, а використовує модульний підхід, тобто до фреймворку Keras можна підключити декілька різних низькорівневих бібліотек.

В даний час підтримуються три такі бібліотеки: TensorFlow, Theano і Microsoft Cognitive Toolkit (CNTK). Будь-код, який використовує Keras, можна запускати з будь-якої з цих бібліотек без необхідності змінювати щось в коді: можна швидко перемикатися між ними в процесі розробки, що часто виявляється корисно, наприклад, якщо одна з бібліотек показує більш високу продуктивність при вирішенні даної конкретної завдання.

2.1.3. Бібліотека matplotlib

Бібліотека на мові програмування Python для візуалізації даних двовимірної (2D) графікою (3D графіка також підтримується). Вихідні зображення можуть бути використані в якості ілюстрацій в публікаціях [4]. Matplotlib являється гнучким, а також легко конфігурованим пакетом, що разом із іншими бібліотеками, такими як NumPy, SciPy і IPython надає можливості, подібні MATLAB.

Пакет може підтримувати наступні види графіків та діаграм: діаграми розкиду (scatter plot), графіки (line plot), гістограми (histogram), стовпчасті діаграми (bar chart), кругові діаграми (pie chart), контурні графіки (contour plot), стовбур-лист діаграми (stem plot), поля градієнтів (quiver) та спектральні діаграми (spectrogram)

2.1.4. Бібліотека для роботи з масивами NumPy

NumPy - це бібліотека для мови програмування Python, що забезпечує підтримку великих, багатовимірних масивів і матриць. з великою колекцією математичних функцій високого рівня для роботи над цими масивами. Родоначальник NumPy, Numeric, спочатку був створений Джимом Хугуніним за участі кількох інших розробників. У 2005 році Травіс Оліфант створив NumPy, включивши в Numarray функції Numarray з числовими змінами з великими модифікаціями. NumPy - це програмне забезпечення з відкритим кодом та має багато учасників.

NumPy націлений на реалізацію еталонної програми Python, яка є оптимізатором інтерфейсу байт-коду. Математичні алгоритми, написані для цієї версії Python, часто працюють набагато повільніше, ніж складені еквіваленти. NumPy частково вирішує проблему повільності, надаючи багатовимірні масиви та функції та операторів, які ефективно працюють над масивами, вимагаючи переписати деякий код, переважно внутрішні петлі за допомогою NumPy.

Основною функціональністю NumPy є її "ndarray" - структура даних для n-мірного масиву. Ці масиви мають строгий вигляд на пам'ять. На відміну від вбудованої структури даних Python списку, у цього масиву всі елементи повинні бути одного типу.

Такі масиви можуть також переглядати буфери пам'яті, що виділяються розширеннями C / C ++, Cython та Fortran в інтерпретатор CPython без необхідності копіювати дані навколо, надаючи ступінь сумісності з існуючими бібліотеками чисел. NumPy має вбудовану підтримку для ndarrays, відображених на пам'ять.

2.1.5. Огляд фреймворку scikit-learn

Даний фреймворк (біблотеки) містить в собі реалізації алгоритмів, заснованих на навчанні з учителем і без. Він використовується при роботі на мові Python.

Scikit-learn є бібліотекою, що спирається на бібліотеку SciPy (Scientific Python), призначену для виконання наукових і інженерних розрахунків. Тому Scientific Python повинна бути встановлена перед використанням scikit-learn.

В даній бібліотеці особлива увага приділяється питанням зручності користування, документації і оптимізації швидкості роботи. До плюсів даної бібліотеки можна віднести простоту і ефективність інструментів аналізу даних, а також можливість використання відкритого вихідного коду.

Бібліотека Scikit-learn орієнтована не на маніпуляції з даними (завантаження, узагальнення), а на моделювання. Деякими функціями, виконуваними Scikit-learn, є:

- Кластеризація (групування об'єктів в безлічі);
- Класифікація (визначення класу об'єкта);
- регресія;
- Скорочення розмірності (зменшення кількості атрибутів);
- Оптимізація параметрів алгоритму (отримання максимально точного результату);
- Алгоритмічні композиції (комбінація декількох моделей);
- Алгоритми навчання з учителем (дискримінантний аналіз, нейронні мережі, дерева прийняття рішень та інші).

2.1.6. Бібліотека для роботи з зображенням OpenCV

OpenCV (Open Source Computer Vision Library) - бібліотека програмного забезпечення з відкритим кодом, яка використовується для машинного навчання. OpenCV був побудований для забезпечення загальної інфраструктури для програм комп’ютерного зору та для прискорення використання машинного сприйняття в комерційних продуктах. Будучи BSD-ліцензованим продуктом, OpenCV дозволяє використання та зміну коду на рівні як бізнесу, так і наукової роботи.

Бібліотека має понад 2500 оптимізованих алгоритмів, що включає в себе вичерпний набір як класичних, так і сучасних алгоритмів комп'ютерного зору

та машинного навчання. Ці алгоритми можна використовувати для виявлення та розпізнавання облич, ідентифікації об'єктів, класифікації дій людини у відео, відстеження рухів камери, відстеження рухомих об’єктів, вилучення 3D-моделей об'єктів, отримання 3D-хмарних точок із стереокамер, скріплення зображень разом для отримання високої роздільної здатності зображення цілої сцени, знаходити подібні зображення з бази даних зображень, видаляти червоні очі із зображень, зроблених за допомогою спалаху, слідкувати за рухами очей, розпізнавати краєвиди та встановлювати маркери для накладання її на розширену реальність тощо. У OpenCV є спільнота із понад 47 тис. користувачів та приблизна кількість завантажень понад 18 мільйонів. Бібліотека широко використовується у компаніях, наукових групах та урядових органах.

Поряд з налагодженими компаніями, такими як Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota, які використовують бібліотеку, існує безліч стартапів, таких як Applied Minds, VideoSurf і Zeitera, які широко використовують OpenCV.

Він має інтерфейси C ++, Python, Java та MATLAB та підтримує Windows, Linux, Android та Mac OS. OpenCV в основному схиляється до додатків зору в режимі реального часу та використовує переваги інструкцій MMX та SSE, коли вони доступні. Існує понад 500 алгоритмів і приблизно в 10 разів більше функцій, які складають або підтримують ці алгоритми. OpenCV написано на мові C ++ і має шаблоновий інтерфейс, який легко працює з контейнерами STL.

2.1.7. Програмна бібліотека для налаштування машинного навчання TensorFlow

Бібліотека TensorFlow була представлена вперше з відкритим кодом в листопаді 2015 року, компанія Google зуміла зацікавити потенційних користувачів, і сьогодні ця технологія стає все більш популярною.

Бібліотека програмного забезпечення для машинного навчання є новим поколінням внутрішньої технології DistBelief, яка була розроблена командою

Google Brain для безлічі завдань, включаючи пошук зображень і поліпшення алгоритмів розпізнавання мови.

TensorFlow - це нейронна мережа, яка вчиться вирішувати завдання шляхом позитивного посилення і обробляє дані на різних рівнях (вузлах), що допомагає знаходити вірний результат.

Відкривши вихідний код бібліотеки машинного навчання TensorFlow, в Google спростили процес побудови і розгортання складних нейронних мереж. TensorFlow не надає кожному розробнику можливості скористатися плодами машинного навчання, але пропонує інтерфейси API для мов Python і C / C ++, що дозволяють підключатися до програми розробника.

Машинне навчання такого роду призначено винятково для дослідницьких цілей, але завдяки програмному забезпеченню з відкритим кодом на зразок TensorFlow підприємство отримує потужні засоби для використання своїх власних даних і їх обробки в дешевій хмарної середовищі.

Бібліотеки TensorFlow помітно спрощують вбудовування в додатки самонавчального елементів і функцій штучного інтелекту, призначених для розпізнавання мови, організації комп'ютерного зору або обробки природної мови.

Звичайно, TensorFlow не єдина бібліотека глибинного навчання, але, як і пошуковий механізм Google, вона вважається кращою в своєму класі. Альтернативами є програмне забезпечення Torch, створене швейцарськими дослідниками, і розробка Каліфорнійського університету в Берклі Caffe, остання версія якої, Caffe2, була спроектована за участю Facebook.

Згідно з інформацією, опублікованою на сайті TensorFlow, бібліотеку використовує цілий ряд великих компаній, в тому числі Airbnb, Airbus, Dropbox, Snapchat і Uber (правда, можливо, і не найкращим для себе способом).

2.2. Аналіз шарів для згорткової нейронної мережі

2.2.1. Convolutional layer

Звитий шар - це основний будівельний блок CNN. Параметри шару складаються з набору вивчених фільтрів (або ядер), які мають невелике сприйнятливе поле, але поширюються на всю глибину вхідного об'єму. Під час прямого проходу кожен фільтр згортається по ширині та висоті вхідного об'єму, обчислюючи крапковий добуток між записами фільтра та входом і створюючи двовимірну карту активації цього фільтра. Як результат, мережа засвоює фільтри, які активуються, коли виявляє певний тип функції у певному просторовому положенні на вході.

Укладання карт активації для всіх фільтрів по глибинному виміру утворює повний вихідний об'єм шару згортки. Кожен запис об'ємного виходу може, таким чином, також інтерпретуватися як вихід нейрона, який дивиться на невелику область вхідних даних і ділиться параметрами з нейронами в одній карті активації[12].

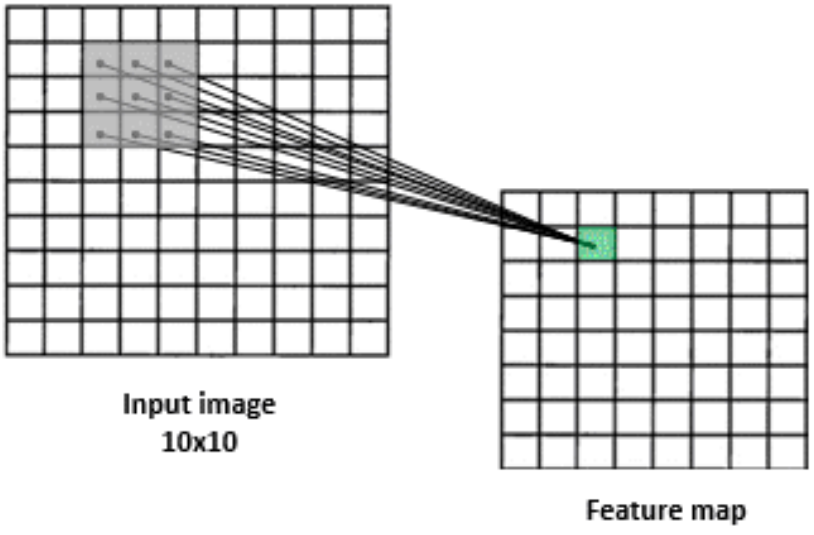


Рис. 2.1. Ілюстрація процесу згортки

2.2.2. Pooling layer

Ще одна важлива концепція CNN - це об'єднання, яке є формою нелінійної вибірки вниз. Існує кілька нелінійних функцій для здійснення пулу,

серед яких найчастіше зустрічається максимальне об'єднання. Він розділяє вхідне зображення на набір прямокутників, що не перекриваються, і для кожного такого субрегіону виводить максимум.

Інтуїтивно зрозуміле, точне місце розташування функції менш важливе, ніж її грубе розташування щодо інших ознак. Це ідея використання об’єднання в конволюційних нейронних мережах. Шар об'єднання служить для прогресивного зменшення просторових розмірів представлення, зменшення кількості параметрів, сліду пам’яті та кількості обчислень у мережі, а отже, також для контролю над насадкою. Часто періодично вставляти шар об'єднання між послідовними згортковими шарами в архітектурі CNN. Операція об'єднання забезпечує ще одну форму інваріантності перекладу.

Шар об'єднання працює незалежно на кожному фрагменті глибини входу і змінює його просторово. Найпоширенішою формою є шар об'єднання з фільтрами розміром 2 × 2, нанесеними з кроком 2 зразків знизу на кожен зріз глибини на вході по 2 по ширині та висоті, відкидаючи 75% активацій. У цьому випадку кожна максимальна операція перевищує 4 числа. Розмір глибини залишається незмінним.

На додаток до максимального об'єднання, пул може використовувати й інші функції, такі як *середнє* об'єднання або об'єднання *ℓ2-норми*. Середнє об'єднання часто використовувалося в минулому, але останнім часом воно не в пріоритеті порівняно з максимальним об'єднанням, яке на практиці краще.

Через агресивне зменшення розміру представлення, останнім часом спостерігається тенденція до використання менших фільтрів або взагалі відкидання об'єднаних шарів.

Об'єднання "Область інтересів" - це варіант максимального об'єднання, в якому розмір виводу є фіксованим, а вхідний прямокутник є параметром.

Об'єднання є важливим компонентом згорткових нейронних мереж для виявлення об'єктів на основі швидкої архітектури R-CNN.

Рис. 2.2. Концепція роботи шару пулінгу

2.2.3. ReLU layer

ReLU - це абревіатура випрямленої лінійної одиниці, яка застосовує ненасичуючу функцію активації:

$$f(x) = max(0, x)$$

Він ефективно видаляє негативні значення з карти активації, встановлюючи їх на нуль. Це збільшує нелінійні властивості функції прийняття рішень і загальної мережі, не впливаючи на сприйнятливі поля шару згортки.

Однак, для активації вибирається замість звичайних функцій функції типу гіпертангенса:

$$f(x) = tanh(x), f(x) = | tanh(x) |$$

і сигмоїдна функція:

$$f(x) = (1 + e^{-x})^{-1}$$

Така функція показує хороші результати при навчанні нейронних мереж і відповідає за відсікання непотрібних деталей в каналі.

2.2.4. Dropout

Оскільки повністю пов'язаний шар займає більшість параметрів, він схильний до надмірного розміщення. Одним із методів зменшення надмірної кількості є dropout. На кожному навчальному етапі окремі вузли або «випадають» з мережі з вірогідністю ***1-p***, або зберігаються з ймовірністю ***p***, так що залишається скорочена мережа; вхідні та вихідні краї до випадаючого

вузла також видаляються. На цій стадії дані навчаються лише скороченою мережею. Потім вилучені вузли знову вставляються в мережу з оригінальними вагами.

На етапах тренінгу ймовірність того, що прихований вузол буде скинутий, як правило, становить 0,5; для вузлів введення це має бути значно нижчим, інтуїтивно зрозумілим, оскільки інформація безпосередньо втрачається при ігноруванні вхідних вузлів.

Уникаючи тренувань усіх вузлів за всіма даними про навчання, dropout зменшує надмірне обладнання. Метод також значно покращує швидкість тренувань. Це робить комбінацію моделей практичною навіть для глибоких нейронних мереж. Ця техніка, здається, зменшує взаємодію вузлів, приводячи їх до вивчення більш надійних функцій, які краще узагальнюють нові дані.

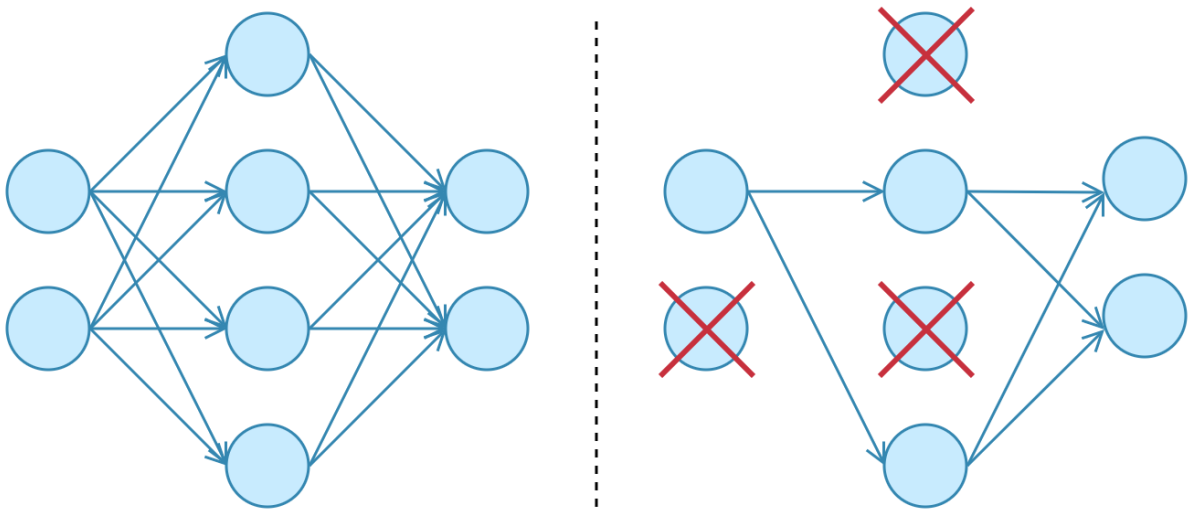


Рис. 2.3. Візуальна демонстрація роботи шару Dropout

2.2.5. Batch normalization layer

Пакетна нормалізація - це метод, який ми можемо використовувати для нормалізації входів кожного шару, щоб боротися з проблемою внутрішнього коваріатного зсуву.

Під час навчального періоду шар нормалізації виконує наступні дії:

- Розраховує середнє значення і дисперсію вхідних шарів.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{Batch mean}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \qquad \text{Batch variance}$$

Статистика партії для кроку 1

- Нормалізує входи шару з використанням раніше обчисленої статистики.

$$\overline{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Нормалізація введення шарів на етапі 2

- Масштабує і зсуває в залежності від отримання виходу шару.

$$y_i = \gamma\overline{x_i} + \beta$$

Масштабування та зміщення нормованого входу для кроку 3

Отже, якщо когда партія мала б m данних и то для j партій отримуємо наступну нормалізацію на виході:

$$E_x = \frac{1}{m}\sum_{i=1}^{j} \mu_B^{(i)} \qquad \text{Inference mean}$$

$$Var_x = \left(\frac{m}{m-1}\right)\frac{1}{m}\sum_{i=1}^{j} \sigma_B^{2(i)} \qquad \text{Inference variance}$$

$$y = \frac{\gamma}{\sqrt{Var_x + \epsilon}}x + \left(\beta + \frac{\gamma E_x}{\sqrt{Var_x + \epsilon}}\right) \qquad \text{Inference scaling/shifting}$$

Формули виводу

2.2.6. Dense/fully connected layer

Нарешті, після декількох згорткових і максимальних об'єднаних шарів, міркування високого рівня в нейронній мережі здійснюються через повністю пов'язані шари. Нейрони у повністю пов'язаному шарі мають з'єднання з усіма активаціями попереднього шару, як це спостерігається у звичайних (не згорткових) штучних нейронних мережах. Таким чином, їх активацію можна обчислити як афінну трансформацію з множенням матриць з подальшим зміщенням зміщення (векторне додавання вивченого або фіксованого терміну зміщення).

2.3. Оптимізатори

Оптимізатор організовує оптимізацію моделі, координуючи прямий і зворотний градієнти мережі для формування оновлень параметрів, які намагаються поліпшити функцію втрат. Обов'язки навчання розподіляються наступним чином: оптимізатор використовується для контролю оптимізації та генерації оновлень параметрів, а мережа - для отримання збитків і градієнтів. В нашому випадку був використаний такий оптимізатор як Adam (Adaptive Moment Estimation).

Adam - це алгоритм оптимізації, який можна використовувати замість класичної процедури стохастичного градієнта спуску для оновлення ітераційних мережевих ваг на основі даних тренувань.

Алгоритм називається Adam і це не є абревіатурою, і не пишеться як "ADAM".

Adam відрізняється від класичного стохастичного градієнтного спуску. Стохастичний градієнтний спуск підтримує єдину швидкість навчання (називається альфа) для всіх оновлень ваги, а рівень навчання не змінюється під час навчання. Коефіцієнт навчання підтримується для кожної ваги мережі (параметра) та окремо адаптується під час розробки навчання.

Цей алгоритм має наступні привабливі переваги у використання для оптимізаційних задач перед іншими оптимізаторами:

- Пряма реалізація.

- Обчислювально ефективний.
- Малі вимоги до пам'яті.
- Інваріантний до діагонального масштабування градієнтів.
- Добре підходить для великих проблем з даними та/або параметрами.
- Підходить для нестаціонарних цілей.
- Підходить для проблем з дуже шумними або розрідженими градієнтами.
- Гіперпараметри мають інтуїтивну інтерпретацію і зазвичай потребують невеликої настройки.

Висновок

Для подальшої роботи і розробки згорткової нейронної мережі використовуватиметься високорівнева мова програмування Python, оскільки вона є однією із основних мов програмування для використання у машинному навчанні. Стандартна бібліотека її хоча і не містить необхідних функцій та можливостей для роботи із нейронними мережами, але Python має відкритий код і багато нових і корисних бібліотек, які можна використовувати абсолютно вільно для своїх потреб, у нашому випадку такими додатками до основної мови програмування є фреймворк Keras, бібліотеки TensorFlow та OpenCV, та інші. Для того, щоб їх встановити та використовувати достатньо лише скористатися системою управляння пакетами в Python під назвою *pip*.

Створення безпосередньо згорткової нейронної мережі буде базуватися на 4-х або 5-х блоках шарів із схожою архітектурою, яка складатиметься із згорткових шарів, шарів пулінгу та ReLU шарів, які іноді будуть розбавлятися шарами Dropout та Normalization.

За оптимізатор буде взятий алгоритм оптимізації під назвою Adam.

РОЗДІЛ 3

Реалізація та тестування згорткової нейронної мережі

3.1. Формальне визначення згорткової нейронної мережі

Згорткові нейронні мережі (CNN) в останні роки повністю домінували в просторі машинного зору. CNN складається з вхідного шару, вихідного шару, а також безлічі прихованих шарів. Приховані шари CNN зазвичай складаються з згорткових шарів, об'єднання шарів, повністю пов'язаних шарів і шарів нормалізації (ReLU). Для складніших моделей можна використовувати додаткові шари. Приклади типового CNN можна побачити на малюнку нижче.

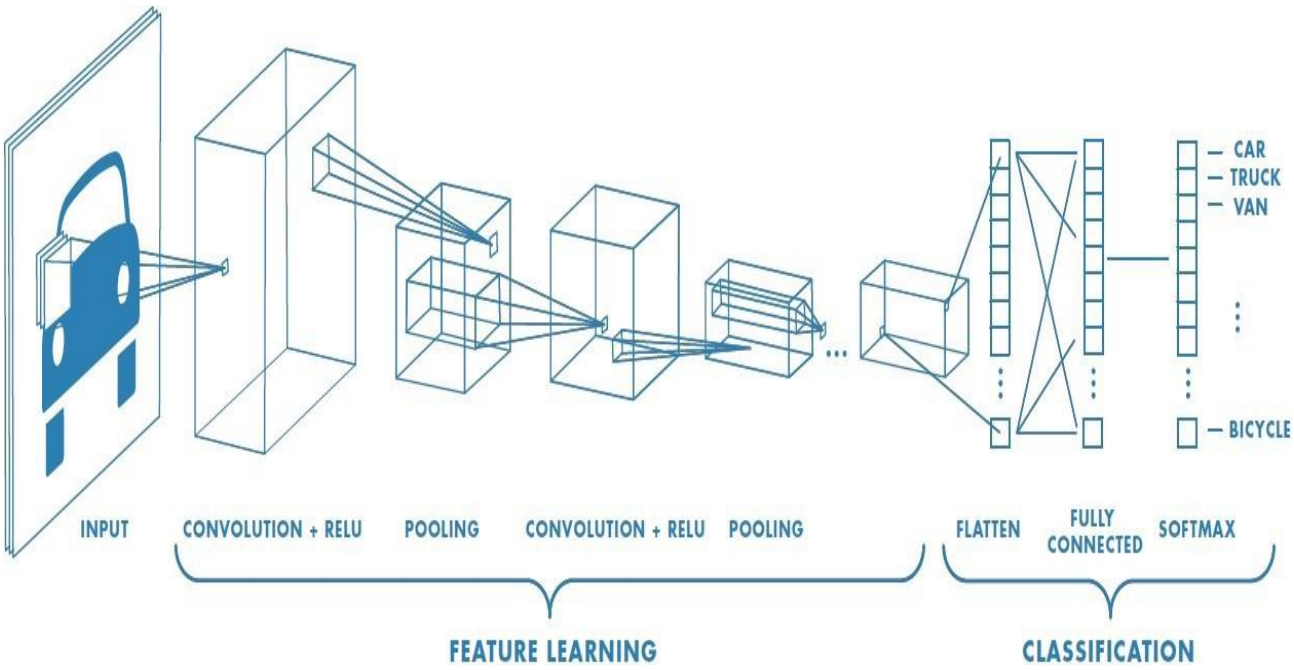


Рис 3.1. Приклад архітектури шарів у CNN

Архітектура CNN показала відмінні результати у багатьох проблемах із комп’ютерним зором та машинним навчанням[2]. CNN тренує та прогнозує на абстрактному рівні з деталями, залишеними для наступних розділів. Ця модель CNN широко використовується в сучасних програмах машинного навчання завдяки постійній ефективності рекордів. Лінійна алгебра є основою для роботи цих CNN. Множення матричного вектора лежить в основі представлення даних і ваг. Кожен з шарів містить різний набір характеристик для набору зображень. Наприклад, якщо зображення обличчя є входом до CNN, мережа вивчить деякі основні характеристики, такі як краї, яскраві плями, темні плями, фігури тощо, у своїх початкових шарах. Наступний набір шарів буде складатися з фігур і предметів, пов'язаних із зображенням, які можна розпізнати, такі як: очі, ніс і рот. Наступний шар складається з аспектів, схожих на фактичні обличчя, іншими словами, форми та об'єкти, які мережа може використовувати для визначення людського обличчя. CNN відповідає деталям, а не цілому зображенню, тому розбиває процес класифікації зображення на більш дрібні частини (характеристики). Сітка 3x3 визначена для відображення функції вилучення CNN для оцінки. Наступний процес, відомий як фільтрування, включає в себе підшивку функції патчем зображення. Один за одним кожен піксель множиться на відповідний піксель функції, і після завершення всі значення підсумовуються та діляться на загальну кількість пікселів у просторі функцій. Кінцеве значення для функції потім розміщується в патчі функції. Цей процес повторюється для інших патчів функцій з подальшим випробуванням усіх можливих повторних застосувань цього фільтра, який відомий як згортка.

Наступний шар CNN називається "максимальним об'єднанням", що передбачає зменшення стека зображення. Щоб об'єднати зображення, потрібно визначити розмір вікна (наприклад, зазвичай 2x2 / 3x3 пікселі), також слід визначити крок (наприклад, 2 пікселі). Потім вікно фільтрується через зображення кроками, при цьому максимальне значення записується для кожного вікна. Максимальне об'єднання зменшує розмірність кожної карти

зображень, зберігаючи найважливішу інформацію. Нормалізаційний шар CNN, який також називають процесом випрямленого лінійного блоку (ReLU), включає зміну всіх негативних значень у відфільтрованому зображенні на 0. Цей крок повторюється на всіх відфільтрованих зображеннях, шар ReLU збільшує не -лінійні властивості моделі.

Наступним кроком CNN є складання шарів (згортка, об'єднання, ReLU), так що вихід одного шару стає входом наступного. Шари можна повторити, що призведе до «глибокої укладання». Заключний шар в архітектурі CNN називається повністю пов'язаним шаром, також відомим як класифікатор. У цьому шарі кожне значення отримує голос за визначення класифікації зображення.

Повністю з'єднані шари часто складаються між собою, при цьому кожен проміжний шар «голосує» за фантомними «прихованими» категоріями. Насправді кожен додатковий шар дозволяє мережі вивчити ще більш складні комбінації функцій для кращого прийняття рішень.

Значення, використовувані для шару згортки, а також ваги для повністю з'єднаних шарів отримують за допомогою зворотного розповсюдження, що здійснюється глибокою нейронною мережею. Зворотне розповсюдження полягає в тому, що нейронна мережа використовує помилку в остаточній відповіді, щоб визначити, наскільки мережа коригується та змінюється.

3.2. Підходи до проектування архітектури мережі

Найчастіше при побудові архітектури CNN складають кілька шарів згортки ректифікації (ReLu), за ними слідують шари пулінгу; цей шаблон продовжує працювати, доки знімок не буде об'єднано до малого розміру. В деякий момент здійснюється перехід до повнозв'язним шарам. Останній повнозв'язний шар містить вихідну інформацію, наприклад, оцінки класу[3]. Іншими словами, найбільш поширена архітектура CNN відповідає схемі:

$$\text{INPUT} \rightarrow [[\text{CONV} \rightarrow \text{RELU}]*N \rightarrow \text{POOL?}]*M \rightarrow [\text{FC} \rightarrow \text{RELU}]*K \rightarrow \text{FC}$$

Тут «*» означає повторення, а POOL? вказує на додатковий шар пулінгу. Крім того, N⩾0, (зазвичай N≤3), M≥0, K≥0 (зазвичай K <3). Нижче представлені архітектури CNN, організовані за цим зразком:

$$INPUT \rightarrow FC$$

реалізує лінійний класифікатор. Тут N = M = K = 0.

$$INPUT \rightarrow CONV \rightarrow RELU \rightarrow FC$$

$$INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 2 \rightarrow FC \rightarrow RELU \rightarrow FC$$

Тут один згортковий шар між кожен шаром пулінгу.

$$INPUT \rightarrow [CONV \rightarrow RELU \rightarrow CONV \rightarrow RELU \rightarrow POOL] * 3 \rightarrow [FC \rightarrow RELU] * 2 \rightarrow FC$$

Тут 2 шари згортки укладаються перед кожним шаром пулінгу.

Це хороша реалізація для великих і більш глибоких мереж, оскільки кілька складених шарів згортки можуть розвивати більш складні властивості вхідних даних перед деструктивною операцією пулінгу[11].

Як правило, віддають перевагу стеку шарів з невеликими фільтрами, як одному великому рецептивної полю свёрточного слоя. Припустимо, ви, дотримуючись нелінійність між шарами, викладаєте 3 шари згортки розміру 3 × 3 один на одного. При такому розташуванні, кожен нейрон на першому згортковому шарі має видом 3 × 3 вхідного обсягу. Нейрони на другому шарі мають видом 3 × 3 першого шару і 5 × 5 вхідного обсягу. Точно також нейрони на третьому шарі мають видом 3 × 3 другого шару і 7 × 7 вхідного обсягу. Припустимо замість цих трьох згорткових шарів, ми хочемо використовувати тільки один згорткових шар з рецептивних полями розміру 7 × 7. Нейрони такого шару матимуть рецептивне поле вхідного обсягу, аналогічне просторової протяжності 7 * 7, але з деякими недоліками. По-перше, нейрони будуть обчислювати лінійну функцію вхідного обсягу, в той час як стек з 3 шарів згортки містить нелінійності, що підкреслюють властивості шарів[11]. По-друге, якщо ми припустимо, що весь вхідний обсяг має C каналів, тоді один згортковий шар розміру 7 * 7 міститиме З * (7 * 7 * C) = 49C2 параметрів, в той час як стек з 3 шарів містить лише 3 * (C * (3 * 3 * C)) = 27C2 параметрів.

Очевидно, що використання декількох згорткових шарів з невеликими фільтрами дозволяє підкреслити більш потужні особливості вхідної інформації з меншою кількістю параметрів. Недоліком є більший обсяг використовуваної пам'яті для збереження всіх проміжних результатів.

Ми в даному проекті будемо налаштовувати згорткову нейронну мережу за наступною схемою:

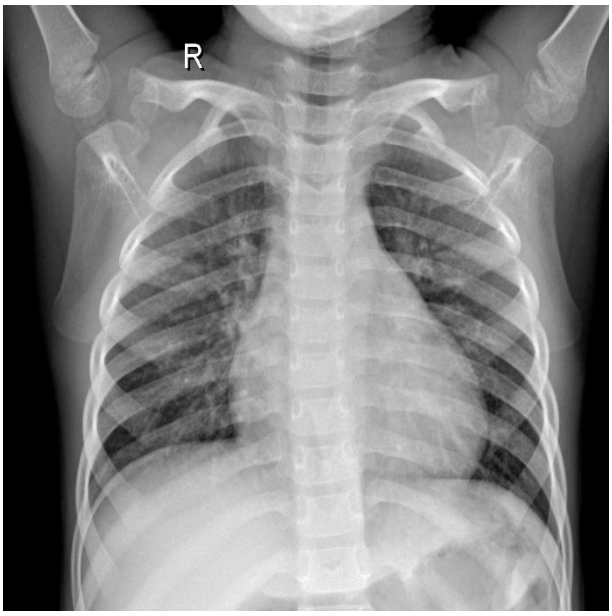
INPUT→[CONV→RELU→CONV→RELU→POOL]∗1→
[CONV→RELU→CONV→RELU→NORMALIZATION→POOL]∗2 →
[CONV→RELU→CONV→RELU→NORMALIZATION→
POOL→DROPOUT]∗2 → [FC→RELU]∗3

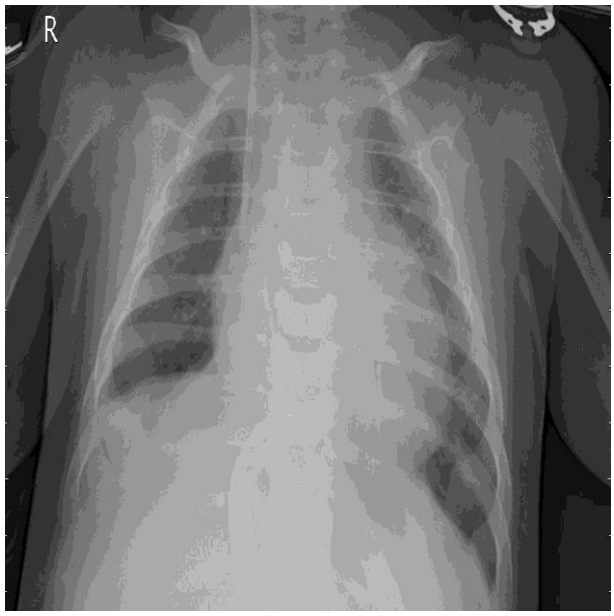
3.3. Створення програмного модулю

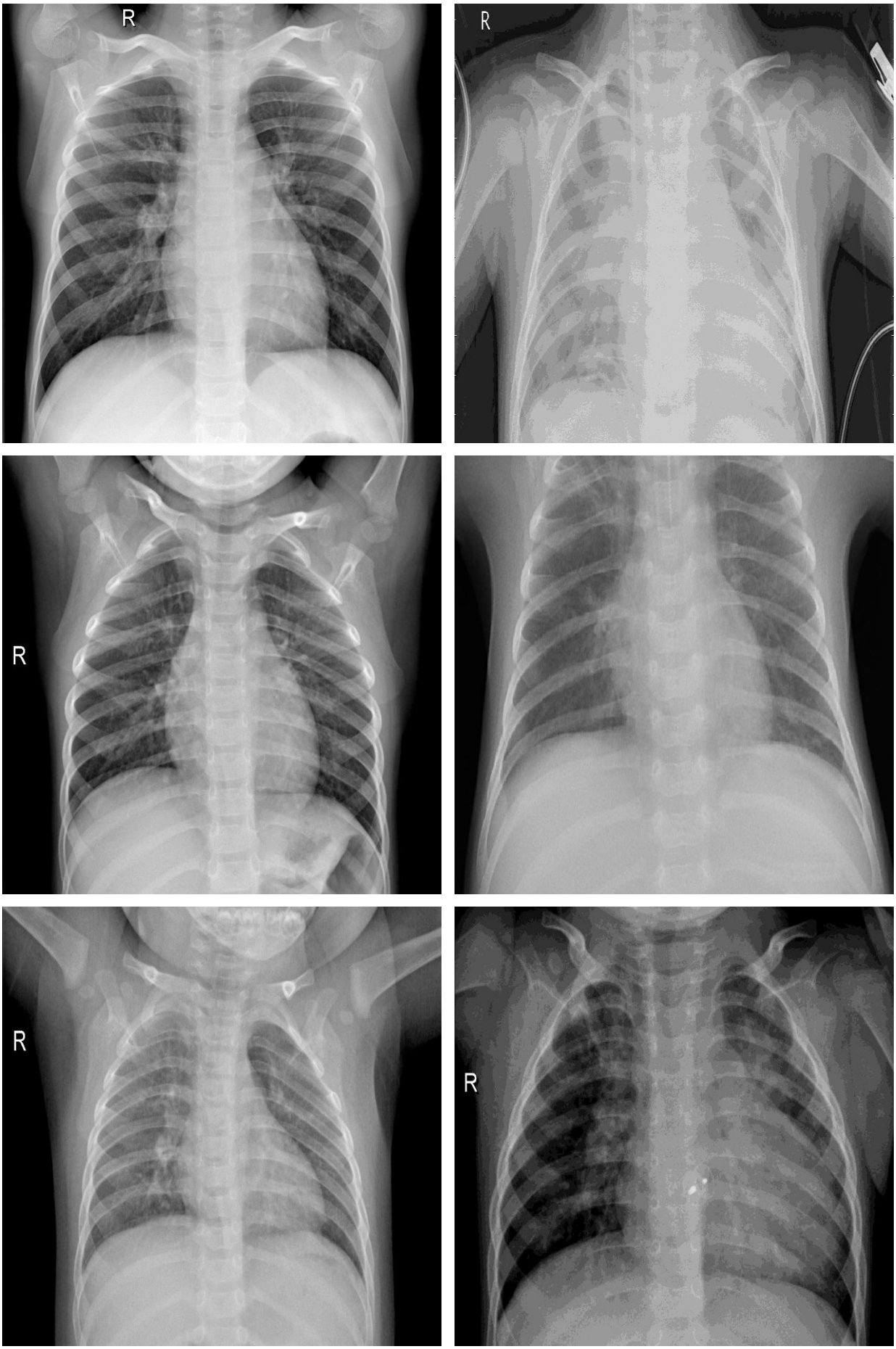
В цьому розділі будуть повністю описані етапи розробки програмної складової даного проекту для класифікації рентгенологічних знімків грудної клітини людини на предмет виявлення у неї пневмонії.

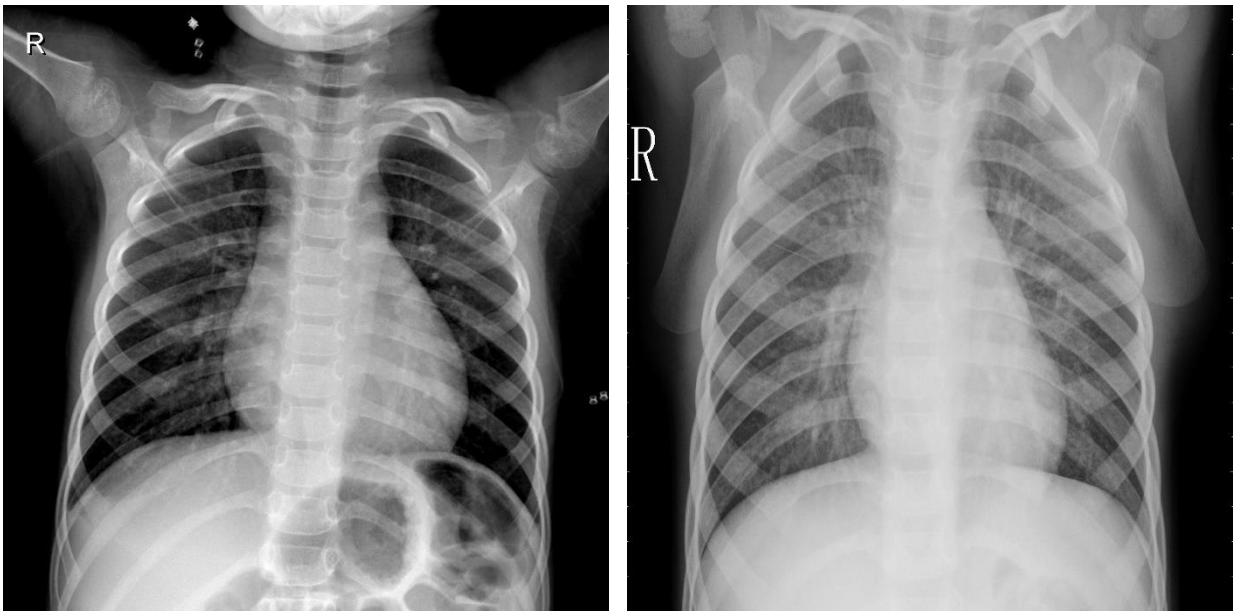
Дані для проведення навчання нейронної мережі, а також для валідації її як в процесі навчання, так і після нього, були використані дані зібрані із відкритого джерела *kaggle.com*[5] у сумарній кількості 5840 екземплярів, із них 5216 цифрових зображень будуть направлені на тренінг нейронних мереж, а 624 для їх тестування і валідації.

Приклади зображень можна побачити на фрагментах нижче:






R
R
R
R
R



Нормальний стан легень *Людина хвора на пневмонію*

Рис. 3.2. Фрагменти зображень для класифікації

Кожне зображення в процесі роботи CNN буде приведено до одного стандартного розміру і лише після цього передається на вхід нейронній мережі.

3.3.1. Реалізація двох додаткових нейронних мереж для порівняння результатів під час тестування.

Під час даного дипломного проекту були реалізовані за допомогою мови програмування Python і усіх вище перелічених бібліотек і методів ще дві згорткові нейронні мережі, які будуть базуватися на загальновідомих попередньо-навчених мережах Inception-v3 та VGG16 для того, щоб порівняти всі три модуля за різними метриками, і дійти висновку щодо створеної в рамках проекту власної мережі.

Модель ***Inception-v3*** - це архітектура згорткових мереж. Це одна з найточніших моделей для класифікації зображень у своїй галузі, яка досягає 3,46% за рівнем "найвищих 5 помилок", які пройшли навчання на наборі даних ImageNet. Спочатку створена командою Google Brain, ця модель може використовувались для різних завдань, таких як виявлення об'єктів, а також інших доменів за допомогою Transfer Learning.

VGG16 - модель згорткової нейронної мережі, представлена К. Симоняном та А. Зіссерманом з Оксфордського університету в статті «Дуже глибокі згорткові мережі для широкомасштабного розпізнавання зображень». Модель досягає точності 92,7% - топ-5, при тестуванні на ImageNet в задачі розташування об'єктів на зображенні. Даний набір даних знаходиться в більш ніж 14 мільйонах зображених, доступних до 1000 класів. Це нейронна мережа є однією із найзнаменитіших моделей, відправлених на змагання ILSVRC-2014. Вона має вдосконалену версію AlexNet, в якій були заміщені великі фільтри (розміри 11 і 5 в першому і другому срібному шарах, відповідно) на кілька фільтрів розміром 3x3, наступних один для інших. Модель VGG16 проходила навчання протягом декількох тижнів, використовуючи відеокарти NVIDIA TITAN BLACK.

Дві ці моделі виступатимуть базовою конфігурацією із додавання декількох шарів за наступною схемою:

BASE_MODEL → DROPOUT → POOL → FC → NORMALIZATION → FC

3.3.2. Процес навчання нейронних мереж

Навчання проводилося на системі із наступними характеристиками:

- CP - 2.4 ГГц (100% зайнятість процесора під час роботи коду)
- RAM – 11.9 GB (93% зайнятості пам’яті)

В процесі навчання було задіяно 5216 зображень двох класів для тренування мереж, при цьому 624 зображення були тестовими безпосередньо під час навчання. На вхід подавалися зображення розміром 150*150, а також була додана функція чекпоінтів для збереження найбільш оптимального значення ваг за метрикою “val_loss” (чим менше значення помилки для тестових даних – тим правильніше налаштовані ваги)

Навчання усіх CNN проводилось по черзі і складалося із 10 епох кожне.

3.3.3. Результати навчання

3.3.3.1 Згорткової нейронної мережі на базі моделі VGG16

Epoch 1/10

163/163 [===] - 7221s 44s/step - loss: 0.4011 - accuracy: 0.8242 - val_loss: 0.8826 - val_accuracy: 0.3750

Epoch 2/10

163/163 [===] - 8425s 52s/step - loss: 0.2971 - accuracy: 0.8681 - val_loss: 0.7777 - val_accuracy: 0.7247

Epoch 3/10

163/163 [===] - 8820s 54s/step - loss: 0.2864 - accuracy: 0.8727 - val_loss: 0.4768 - val_accuracy: 0.6199

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.

Epoch 4/10

163/163 [===] - 8864s 54s/step - loss: 0.2222 - accuracy: 0.9064 - val_loss: 1.4179 - val_accuracy: 0.6267

Epoch 5/10

163/163 [===] - 9022s 55s/step - loss: 0.2007 - accuracy: 0.9206 - val_loss: 1.3873 - val_accuracy: 0.6149

Epoch 00005: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.

Epoch 6/10

163/163 [===] - 7259s 45s/step - loss: 0.1818 - accuracy: 0.9293 - val_loss: 0.3818 - val_accuracy: 0.6639

Epoch 7/10

163/163 [===] - 7226s 44s/step - loss: 0.1796 - accuracy: 0.9296 - val_loss: 0.4437 - val_accuracy: 0.7061

Epoch 00007: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.

Epoch 8/10

163/163 [===] - 7277s 45s/step - loss: 0.1679 - accuracy: 0.9383 - val_loss: 0.1687 - val_accuracy: 0.8902

Epoch 9/10

163/163 [===] - 7185s 44s/step - loss: 0.1600 - accuracy: 0.9379 - val_loss: 0.3617 - val_accuracy: 0.8294

Epoch 00009: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.

Epoch 10/10

163/163 [===] - 7477s 46s/step - loss: 0.1612 - accuracy: 0.9377 - val_loss: 0.4113 - val_accuracy: 0.8733

Було передбачувано те, що саме ця нейронна мережа витратить найбільше часу на її навчання, адже сама модель VGG16 вирізняється не тільки великою завантаженістю пам’яті на диску, але й часом тренінгу на зображеннях, в даному випадку це зайняло в середньому 7 878 секунд на одну епоху, що дорівнює близько 2 години 11 хвилин.

На графіку нижче ми побачимо, що на тренінгових даних модель показала себе прекрасно і точність постійно росла, а втрати навпаки зменшувались

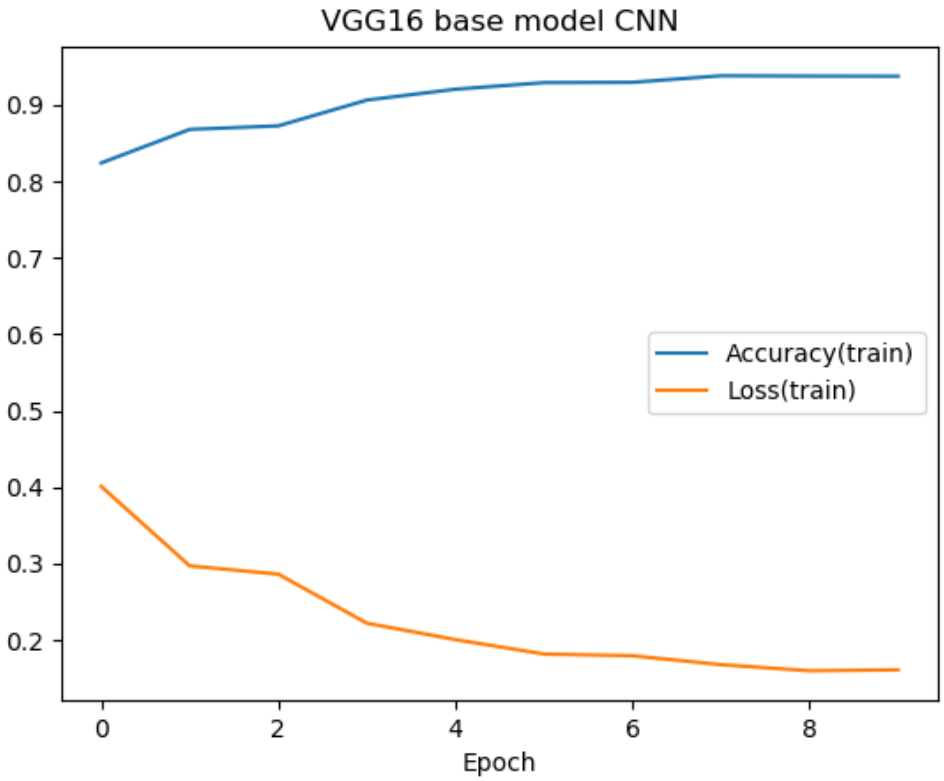


Рис. 3.3. Точність і втрати під час навчання нейронної мережі на базі моделі VGG16

Якщо ж поглянути на графік цієї ж моделі, але на тестових даних, то справи трохи гірші,

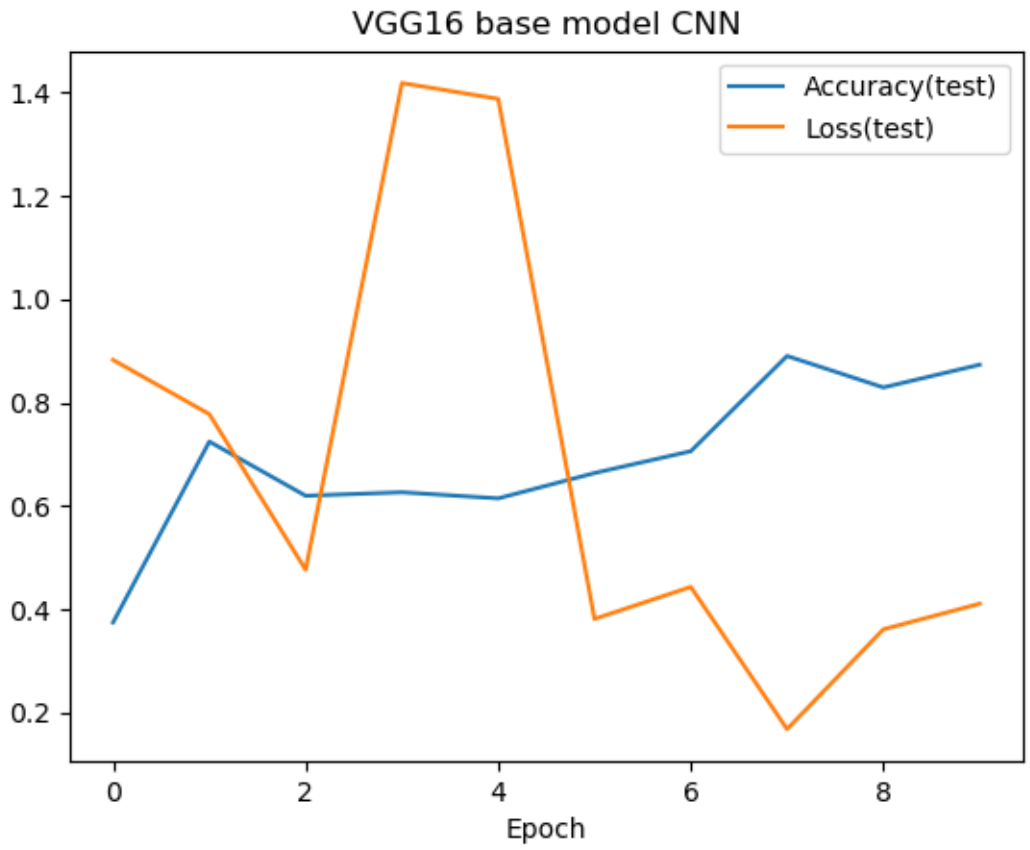


Рис 3.4. Графік точності і втрати тестових даних під час навчання нейронної мережі на базі моделі VGG16

видно, що із другої до п'ятої епохи значення втрат сильно змінилося і графік не має зрозумілого тренду, тому передбачити яким буде навчання далі важко, але з графіку зрозуміло що найкращі ваги були під час 7-ї епохи.

3.3.3.2 Згорткової нейронної мережі на базі моделі Inception-v3

Epoch 1/10

163/163 [===] - 1878s 12s/step - loss: 0.3239 - accuracy: 0.8911 - val_loss: 13677.5908 - val_accuracy: 0.3766

Epoch 2/10

163/163 [===] - 1923s 12s/step - loss: 0.2118 - accuracy: 0.9225 - val_loss: 0.4207 - val_accuracy: 0.8193

Epoch 3/10

163/163 [===] - 1968s 12s/step - loss: 0.1716 - accuracy: 0.9373 - val_loss: 0.4953 - val_accuracy: 0.7922

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.

Epoch 4/10

163/163 [===] - 1890s 12s/step - loss: 0.1277 - accuracy: 0.9553 - val_loss: 0.3978 - val_accuracy: 0.9003

Epoch 5/10

163/163 [===] - 1913s 12s/step - loss: 0.1054 - accuracy: 0.9620 - val_loss: 0.2503 - val_accuracy: 0.9003

Epoch 00005: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.

Epoch 6/10

163/163 [===] - 1909s 12s/step - loss: 0.0929 - accuracy: 0.9697 - val_loss: 0.3040 - val_accuracy: 0.9307

Epoch 7/10

163/163 [===] - 1912s 12s/step - loss: 0.0897 - accuracy: 0.9722 - val_loss: 0.2407 - val_accuracy: 0.9358

Epoch 00007: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.

Epoch 8/10

163/163 [===] - 1914s 12s/step - loss: 0.0680 - accuracy: 0.9774 - val_loss: 0.3595 - val_accuracy: 0.9257

Epoch 9/10

163/163 [===] - 1923s 12s/step - loss: 0.0648 - accuracy: 0.9768 - val_loss: 0.1724 - val_accuracy: 0.9071

Epoch 00009: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.

Epoch 10/10

163/163 [===] - 1921s 12s/step - loss: 0.0619 - accuracy: 0.9772 - val_loss: 0.3399 - val_accuracy: 0.9206

На відміну від попередньої моделі, дана в рази швидше навчалася і витрачала менше оперативної пам’яті комп’ютера.

Граф нижче дає нам зрозуміти, що кінцевий результат на даних тренінгу дуже схожий на той, що ми бачили у згортковій нейронній мережі на базі VGG16, і навіть трохи краще.

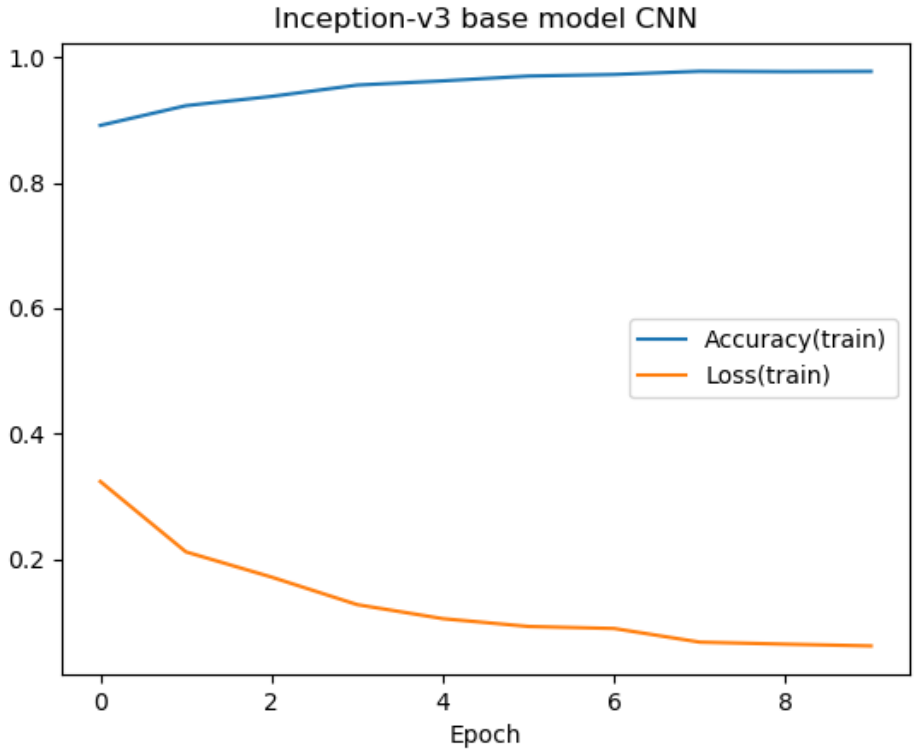


Рис 3.5. Графік точності і втрати під час навчання нейронної мережі на базі моделі Inception-v3

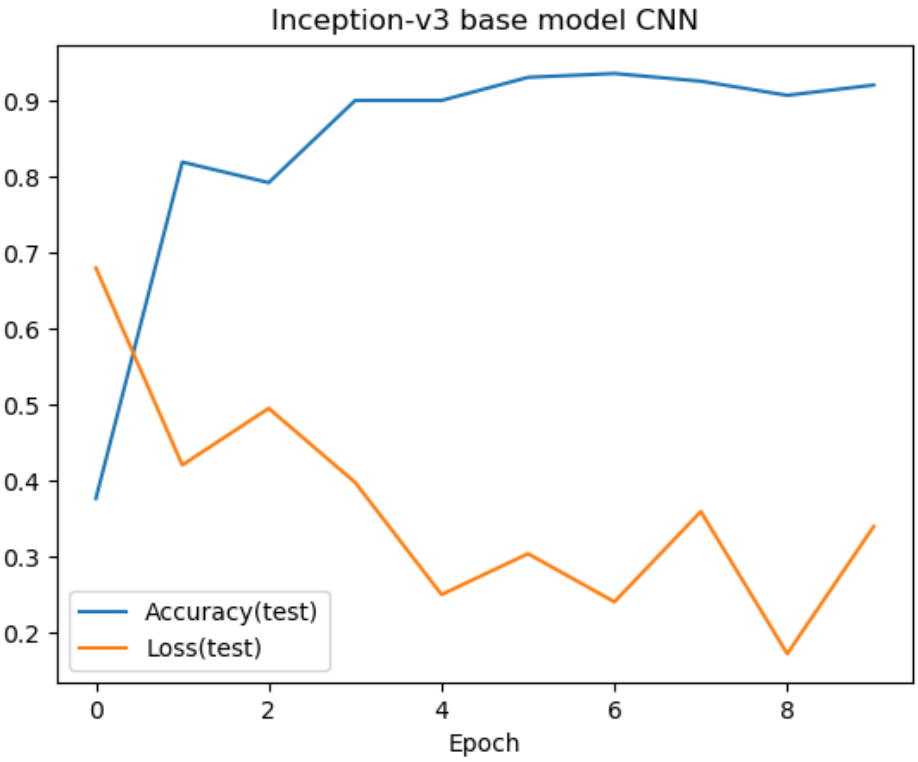


Рис 3.6. Графік точності і втрати тестових даних під час навчання нейронної мережі на базі моделі Inception-v3

Графік із тестовими даними (рис. 3.6.) дає нам зрозуміти, що в цієї моделі вже є тренд, згідно з якого точність із кількістю епох тільки покращується, а втрати стають меншими

І знову, що цікаво, то згідно з тестових даних бачимо, що найкращі ваги були не на 10-й епосі, а на 8-й.

3.3.3.2 Розробленої згорткової нейронної мережі

Epoch 1/10

163/163 [===] - 467s 3s/step - loss: 0.3655 - accuracy: 0.8395 - val_loss: 0.7054 - val_accuracy: 0.6234

Epoch 2/10

163/163 [===] - 409s 3s/step - loss: 0.2732 - accuracy: 0.8861 - val_loss: 0.6771 - val_accuracy: 0.6843

Epoch 3/10

163/163 [===] - 489s 3s/step - loss: 0.2689 - accuracy: 0.8926 - val_loss: 0.7043 - val_accuracy: 0.7215

Epoch 4/10

163/163 [===] - 538s 3s/step - loss: 0.2248 - accuracy: 0.9124 - val_loss: 0.7643 - val_accuracy: 0.7343

Epoch 5/10

163/163 [===] - 412s 3s/step - loss: 0.1988 - accuracy: 0.9239 - val_loss: 0.6138 - val_accuracy: 0.7551

Epoch 6/10

163/163 [===] - 425s 3s/step - loss: 0.1839 - accuracy: 0.9331 - val_loss: 0.4491 - val_accuracy: 0.7956

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.

Epoch 7/10

163/163 [===] - 436s 3s/step - loss: 0.1637 - accuracy: 0.9402 - val_loss: 0.3606 - val_accuracy: 0.8699

Epoch 8/10

163/163 [===] - 474s 3s/step - loss: 0.1476 - accuracy: 0.9484 - val_loss: 0.5867 - val_accuracy: 0.8868

Epoch 00008: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.

Epoch 9/10

163/163 [===] - 477s 3s/step - loss: 0.1328 - accuracy: 0.9567 - val_loss: 0.2773 - val_accuracy: 0.9139

Epoch 10/10

163/163 [===] - 472s 3s/step - loss: 0.1305 - accuracy: 0.9534 - val_loss: 0.3104 - val_accuracy: 0.9088

Epoch 00010: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.

Навчання цієї нейронної мережі зайняло найменше часу, а порівняно із моделлю на базі VGG16, то взагалі майже у 20 разів

Якщо подивитися на графік точності та втрат на тренінгових даних, то ця модель нічим не гірша за попередні, а в деяких моментах і трохи краща.

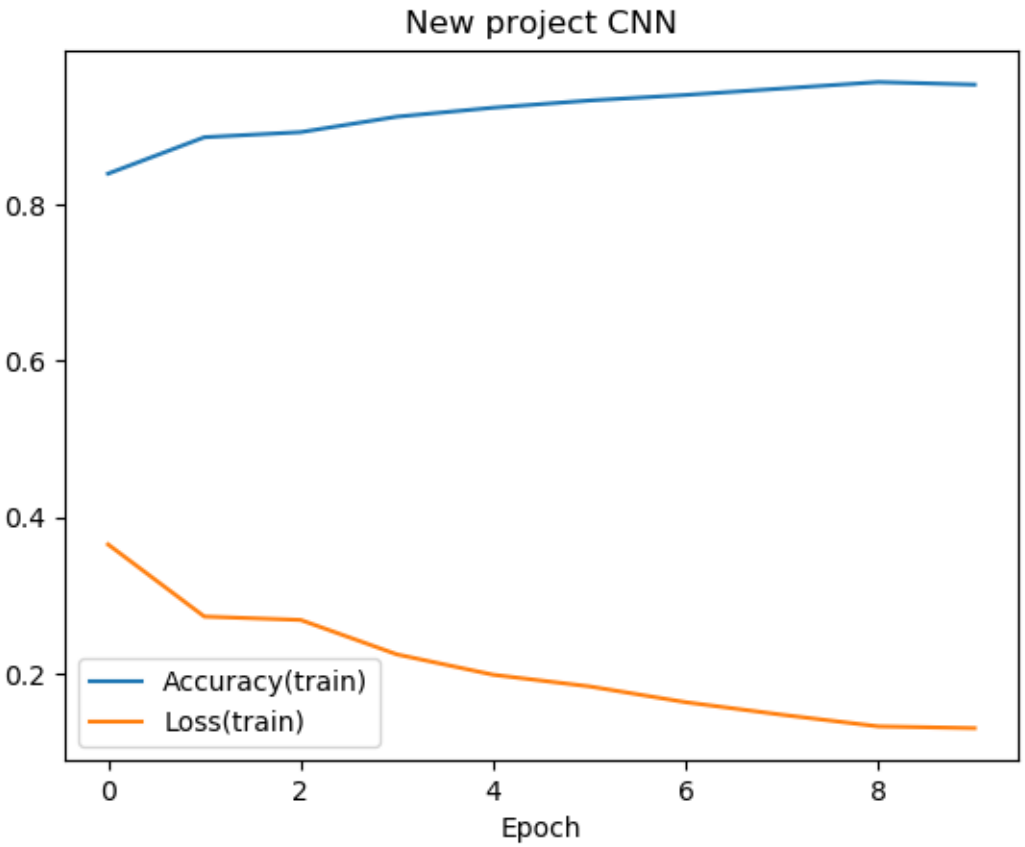


Рис 3.7. Графік точності і втрати під час навчання розробляємої згорткової нейронної мережі

А ось точність і втрати на тестових даних дуже неоднозначні, перші 3-4 епохи створюють хаос, але далі втрати поступово ідуть на спад, лише з одним винятком на 7-й епосі.

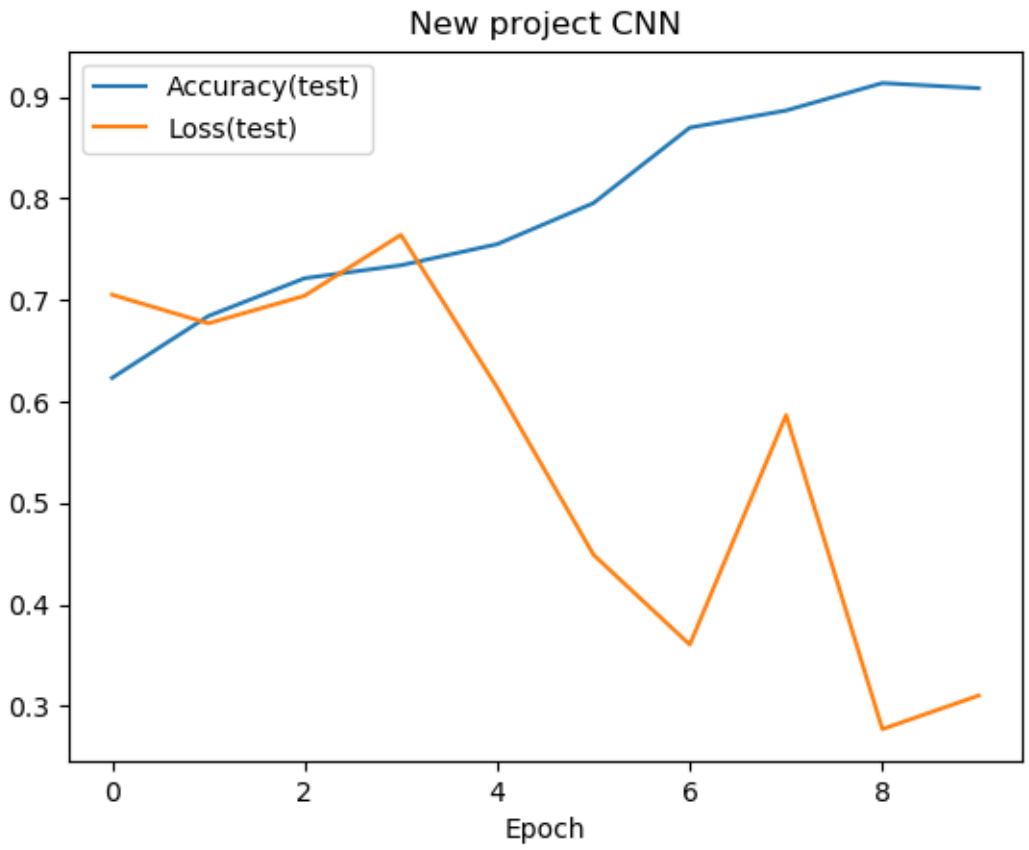


Рис 3.8. Графік точності і втрати тестових даних під час навчання розробляємої згорткової нейронної мережі

3.4. Проведення тестування створеного модуля і порівняння результатів

Для тестування створеної нейронної мережі використовуватимуться зображення із бази даних *kaggle.com* попередньо підігнані під розмір 150*150, який приймає кожна с нейронних мереж на вхід. Дані розподілені на 2 класи, мають схожий ракурс зображень, але відрізняються розмірами грудної клітини, якістю знімку, наявністю шуму. В базі наявні рентгенологічні знімки як дітей, так і дорослих людей, що додаватиме стабільності і універсальності моделям при класифікації зображень.

Під час проведення тестових робіт на персональному комп'ютері не виконувалися ніякі інші операції, щоб не впливати ніяким чином на результати тестування.

3.4.1. Критерії тестування розпізнавання

Для визначення ступеня задовільності результату навчання згорткової нейронної мережі використовується ряд метрик.

Часом розпізнавання є той час, який потрібен детектору для знаходження об'єктів на одному вхідному зображенні. Для аналізу розглянутих методів розпізнавання за даним критерієм використовується середнє арифметичне час роботи детектора, тобто при проведенні замірів часу на N зображеннях, за середній час роботи будемо розуміти величину, наведену на формулі:

$$\frac{1}{N}\sum_{i=1}^{N} ti \ ,$$

де *ti* - час обробки детектором *i*-ого зображення.

Метриками ж для визначення якості отриманого результату виступатимуть:

- точність (precision);
- повнота (recall).
- F-міра (F-score)

Їх значення обчислюються на основі параметрів true positive (TP), true negative (TN), false positive (FP) і false negative (FN). Більш докладно про них:

- True positive (TP, істинно-позитивне значення): на зображенні присутній шуканий об'єкт, результат класифікації позитивний (тобто класифікатор вирішує, що шуканий об'єкт є на зображенні).
- True negative (TN, істинно-від'ємне значення): на зображенні відсутній шуканий об'єкт, результат класифікації негативний (тобто класифікатор вирішує, що шуканого об'єкта немає на зображенні).
- False positive (FP, помилково-позитивне значення): на зображенні відсутній шуканий об'єкт, результат класифікації позитивний.

- False negative (FN, помилково-від'ємне значення): на зображенні присутній шуканий об'єкт, результат класифікації негативний.

Точність (precision) інтерпретується як частка об'єктів, які класифікатор визначив як позитивні, при цей дійсно є позитивними. Обчислюється за такою формулою:

$$precision = \frac{TP}{TP+FP}.$$

Повнота (recall) показує, яка частка позитивних об'єктів була знайдена класифікатором з усіх об'єктів позитивного класу. Обчислюється за такою формулою:

$$recall = \frac{TP}{TP+FN}.$$

F-міра є гармонійним середнім між точністю і повнотою. Вона прагне до нуля, якщо точність або повнота прагне до нуля.

Дана формула надає однакову вагу точності і повноты, тому F-міра буде падати однаково при зменшенні і точності і повноти.

F-міра є хорошим кандидатом на формальну метрику оцінки якості класифікатора. Вона зводить до одного числа дві інших основоположних метрики: точність і повноту.

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

3.4.2. Тестування

Для тестування створених і навчених згорткових нейронних мереж ми використовуватимемо по-чергове надання даних починаючи від 124 і закінчуючи 624 екземплярами цифрових зображень.

Якщо подивитися на рисунок нижче, то можна побачити що точність на найменшій розмірності даних найбільша, це можна пов’язати із тим, що

зображення на цій розмірності мають кращу якість і менше шуму, тому точність достатньо велика.

Порівнюючи ж нейронні мережі, робиться висновок, що початкова точність у власної створеної нейронної мережі та в мережі на базі Inception-v3 майже однакова, а от на більших об’ємах даних власноруч побудована мережа краще справляється із класифікацією цифрових зображень на базі метрики «точність».

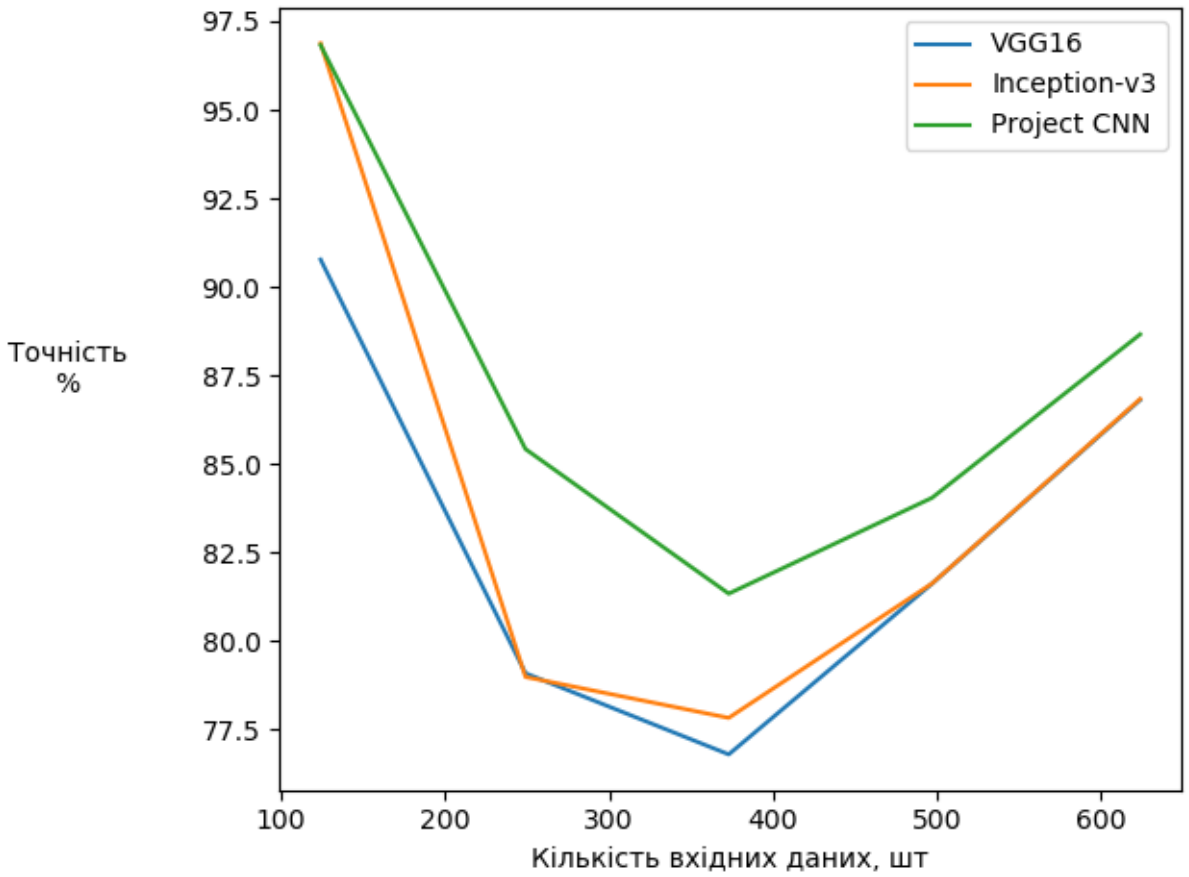


Рис 3.9. Графік точності для різної кількості даних на вхід кожної нейромережі

Повнота класифікації рентгенологічних знімків наглядно продемонстрована на графіку нижче. Відчутно видно, що за цим критерієм невелику перевагу має нейромережа на базі Inception-v3, а ось нейронна мережа на базі VGG16 знову демонструє відверту не кращі показники

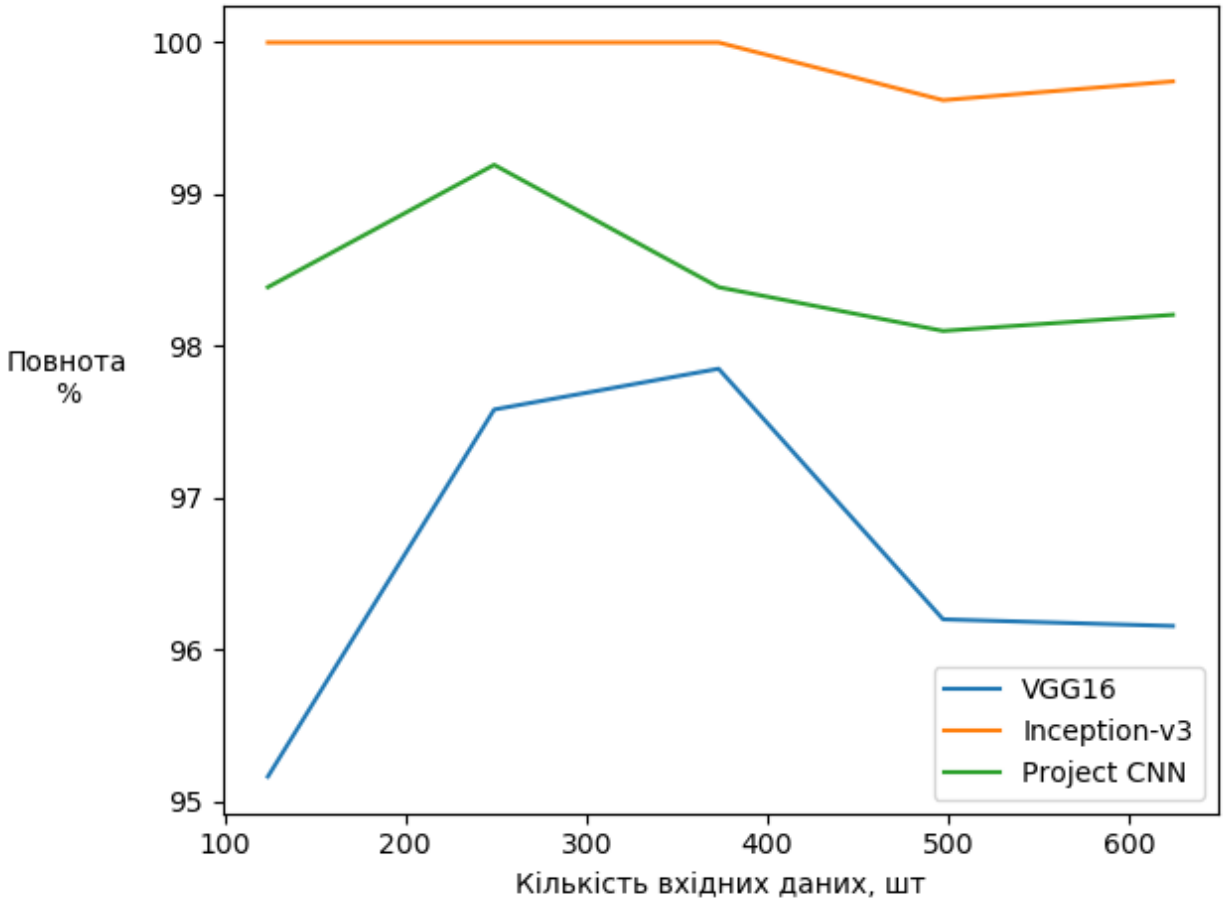


Рис. 3.10. Графік повноти для різної кількості даних на вхід кожної нейромережі

Час, за який проводиться класифікація зображення до певного класу може бути свідком того наскільки швидко можна обробити дані і віднести їх до кожного із класів (рис. 3.11.).

На графіку нижче видно, що найбільш продуктивною за даною метрикою була нейронна мережа розроблена в рамках дипломного проекту, доволі швидко також впорається і Inception-v3, однак для роботи із мережею на базі VGG16 частіше за все потребується додатковий час.

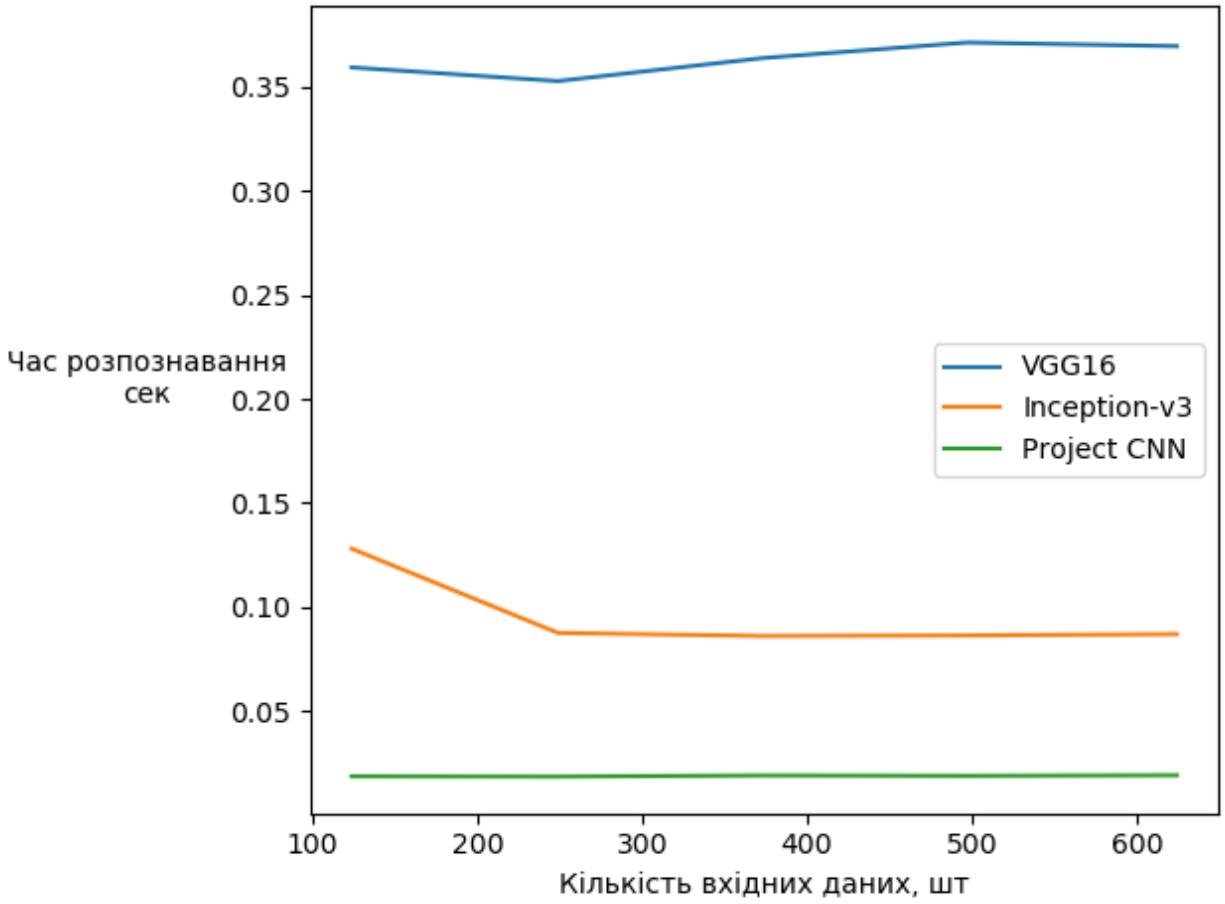


Рис 3.11. Графік часу розпознавання для різної кількості даних на вхід кожної нейромережі

Останньою і найбільш ефективною метрикою являється F-міра, і як ми можемо побачити на графі нижче, усі три нейромережі на відмінно справляються із задачею розпізнавання зображень, і кожна має значення більше 90%.

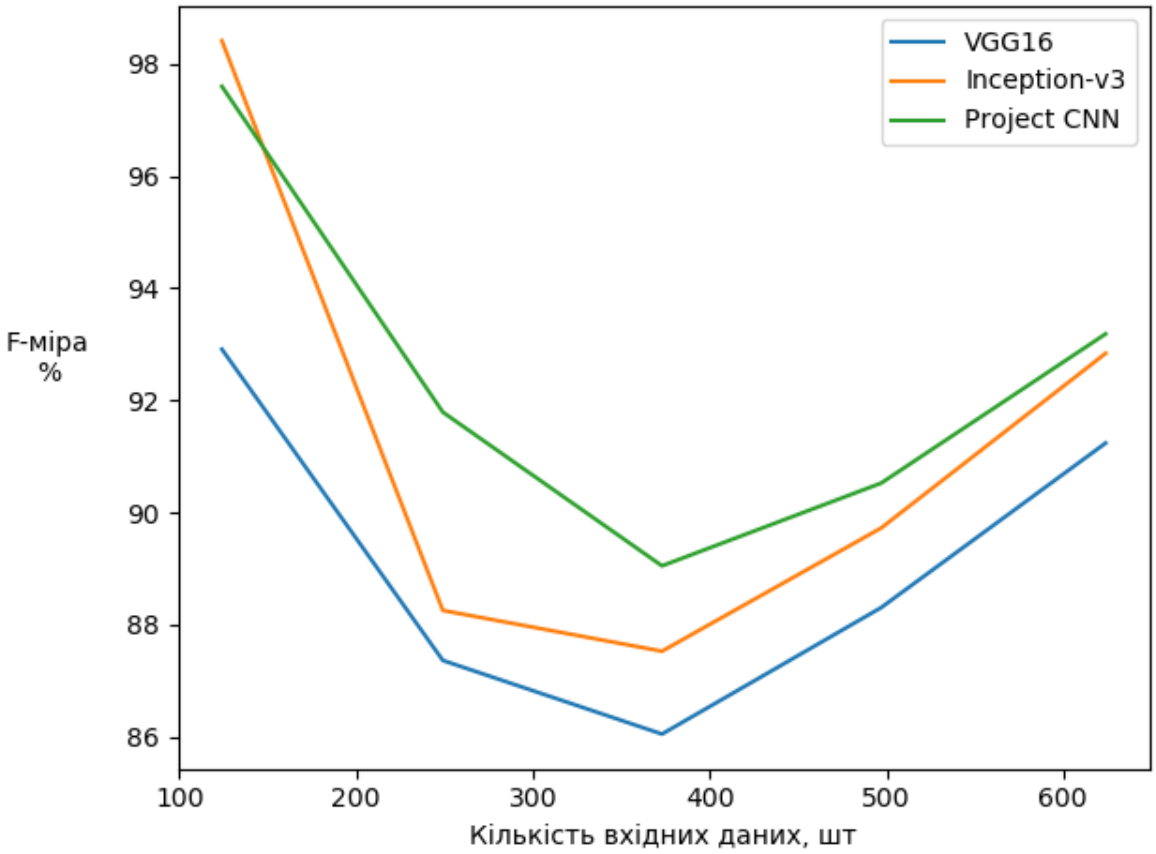


Рис 3.12. Графік F-міри для різної кількості даних на вхід кожної нейромережі

3.4.3. Результати

За результатами тестування навчених нейронних мереж можемо створити таблицю відповідних метрик (табл. 1)

	Розроблена CNN	CNN з основою Inception-v3	CNN з основою VGG16
Повнота (%)	98.2	99.7	96.2
Точність (%)	88.7	86.8	86.8
Час розпізнавання (сек)	12.1	54.3	230.5
F-міра (%)	93.2	92.8	91.2

Таблиця 1 - Результати

За результатами тестування робиться висновок, що розроблена в ході роботи згорткова нейронна мережа чудово себе проявляє у зазначених цілях, а саме класифікації цифрових зображень. Таким чином, можна казати про достатню ефективність застосування згорткових нейронних мереж для розпізнавання і класифікації цифрових зображень.

Нижче приведені графіки порівняння чотирьох представлених в таблиці метрик (повнота, точність, час та F-міра)

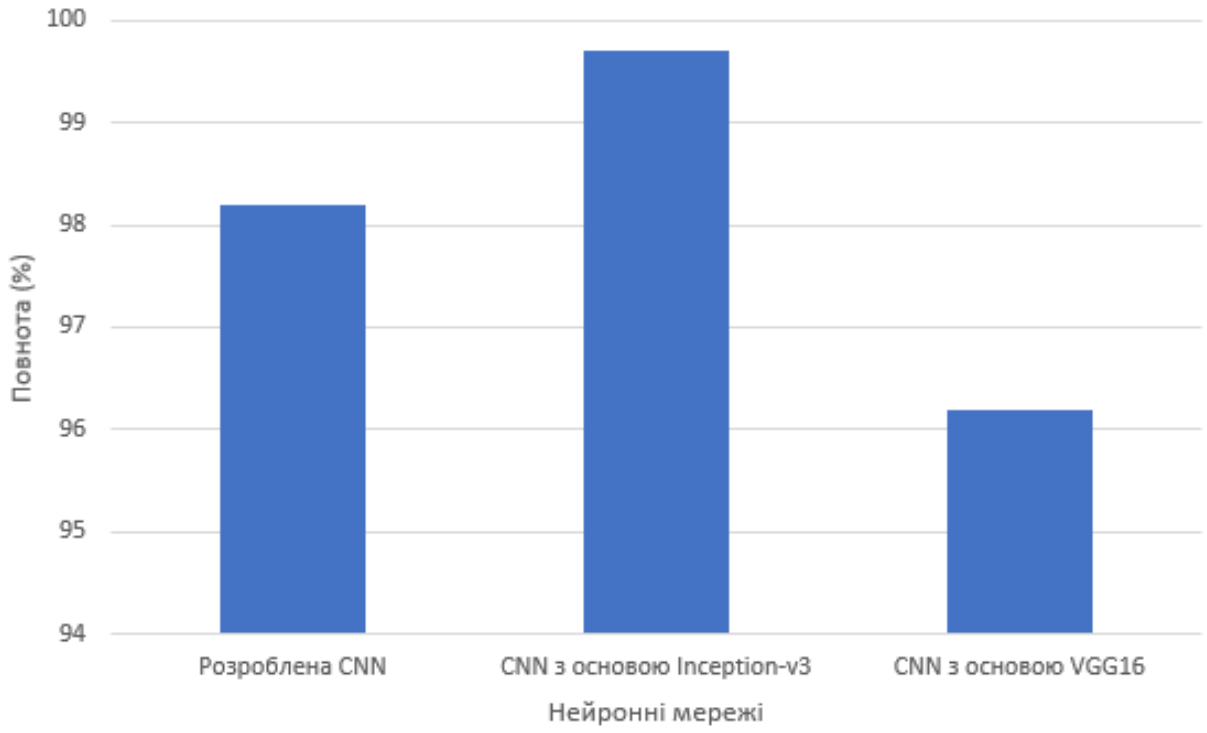


Рис. 3.13. Метрика повноти

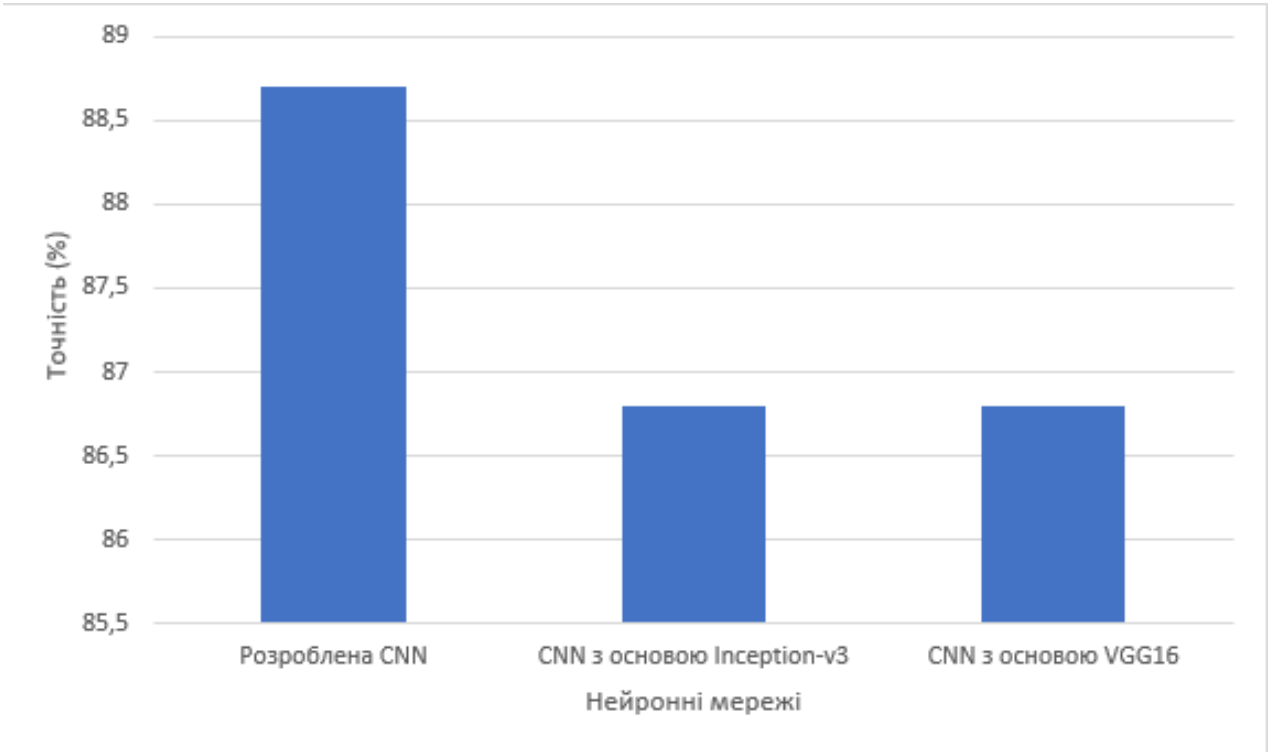


Рис. 3.14. Метрика точності

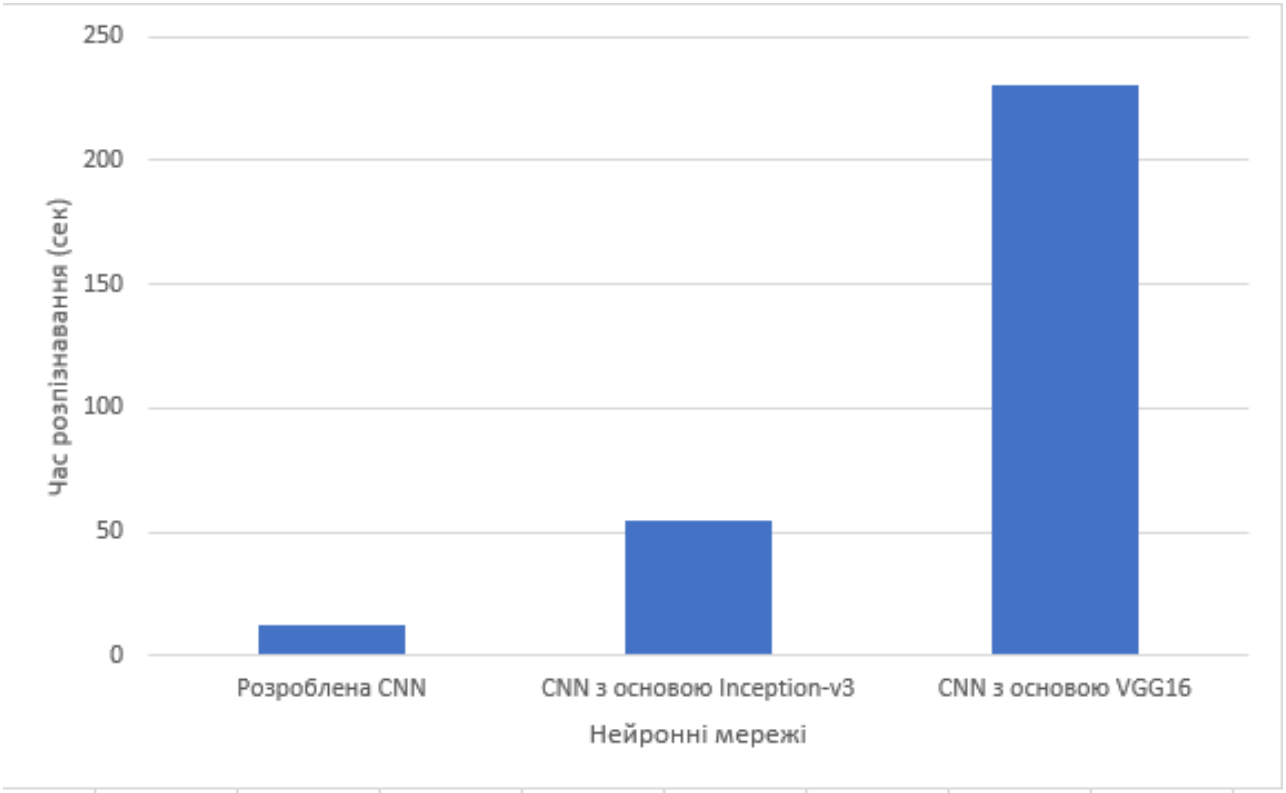


Рис. 3.15. Метрика затраченого на розпізнавання часу

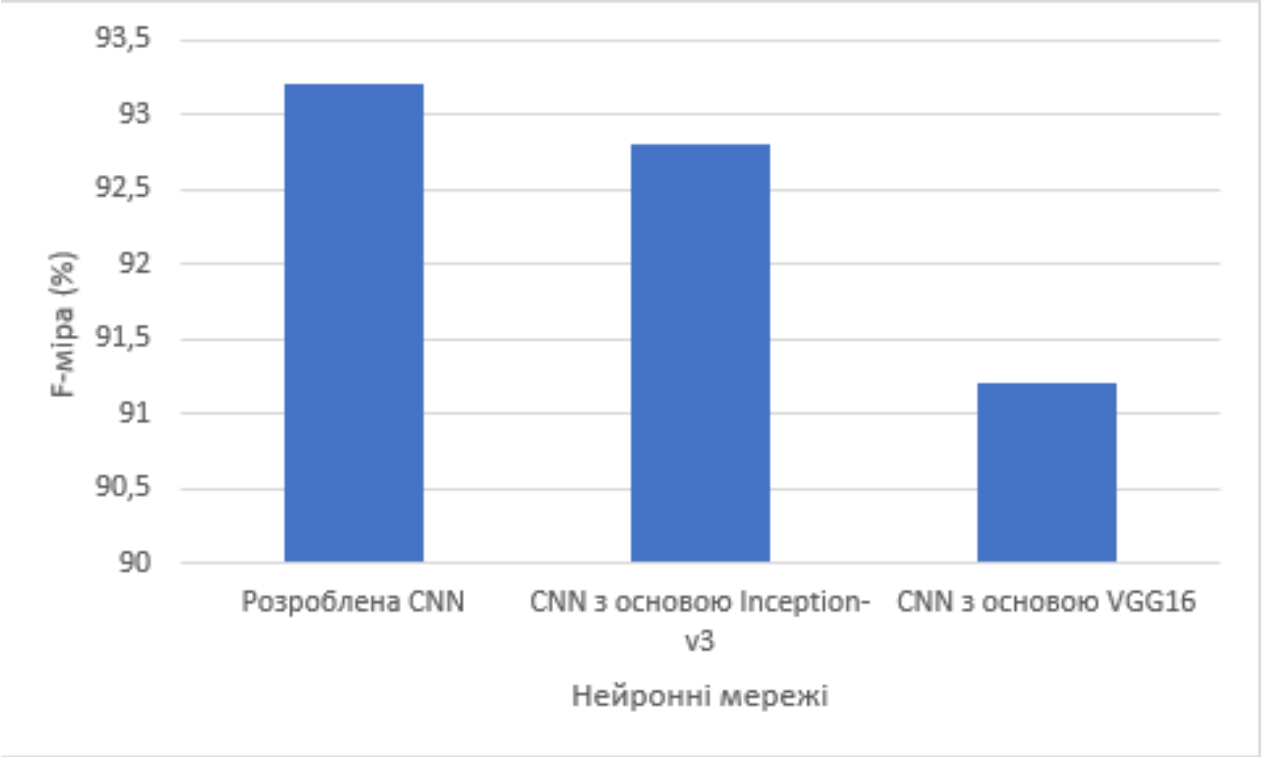


Рис. 3.16. Метрика F-міри

Окрім цих метрик, нижче будуть приведені ще графіки матриці плутанини для кожної нейронної мережі, які дають зрозуміти як саме діяла модель при класифікації тестових цифрових зображень.

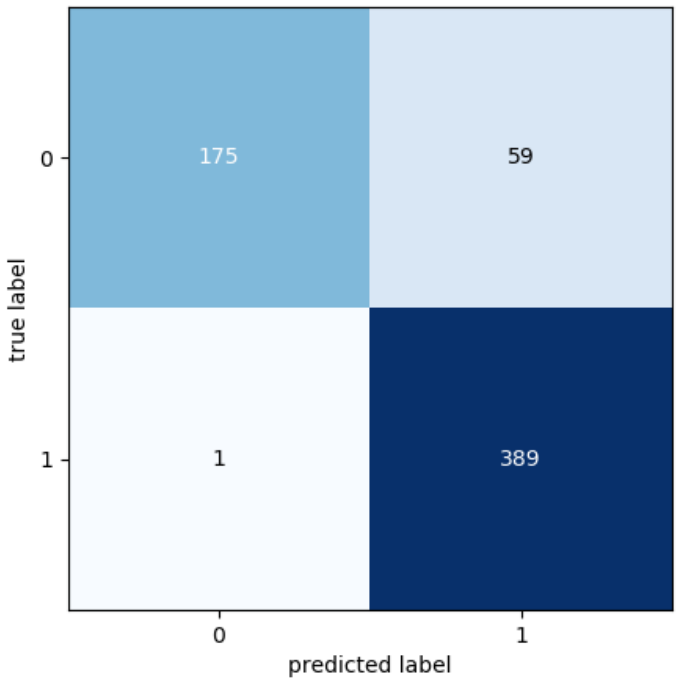


Рис. 3.17. Матриця плутанини для нейронної мережі на базі Inception-v3

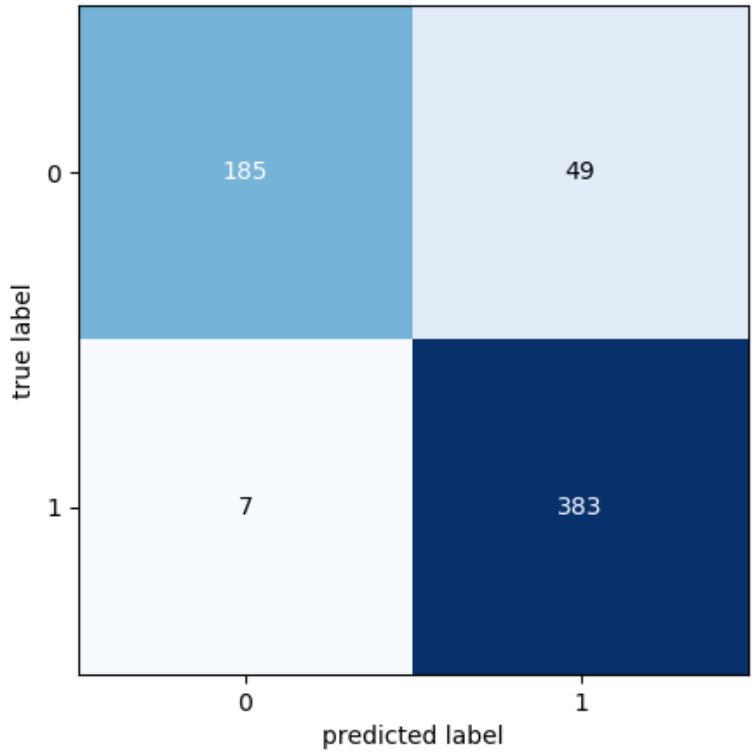


Рис. 3.18. Матриця плутанини для розробленої в проекті нейронної мережі

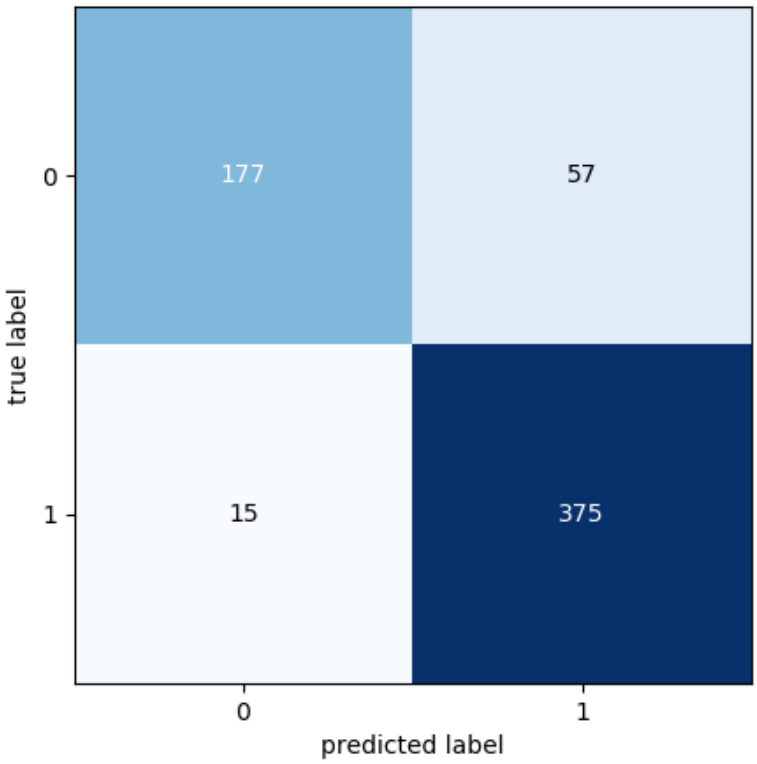


Рис. 3.19. Матриця плутанини для нейронної мережі на базі VGG16

Висновок

Виходячи із отриманих даних, можна зробити висновок, що за вказаними вище метриками розроблений модуль не поступається таким гігантам в сфері згорткових нейронних мереж, в частості для класифікації цифрових зображень, як Inception-v3 та VGG16, а в деяких особливостях навіть кращий, а саме в затратах часу і ресурсів на навчання моделі і в підсумку на класифікацію за допомогою цієї нейромережі.

ВИСНОВКИ

В результаті виконання магістерської дипломної роботи вивчені методи машинного навчання і, зокрема, згорткові нейронні мережі стосовно до розпізнавання та класифікації цифрових зображень. Здійснено огляд існуючих аналогів і рішень для розпізнавання і розкладання на класи цифрових зображень.

Також, вивчена документація і освоєно використання фреймворків Keras, scikit-learn, бібліотека OpenCV і TensorFlow. Розроблено архітектуру програмного модуля, а також здійснена його реалізація на мові програмування Python версії 3.7.5.

Проведено навчання згорткової нейронної мережі, а також тестування класифікації цифрових зображень за різними метриками, і порівняння їх із доволі відомими моделями шарів згорткової нейронної мережі.

З результатів яких стало відомо, що максимальна точність розпізнавання завдяки підібраною архітектурі мережі сягає близько 91%, що є не просто гарним результатом, а й перевищує очікування, а саме тому, що в деяких моментах вигравала у набагато більш відомих моделей нейронної мережі.

Розроблений модуль не тільки відмінно впорався із поставленим завданням за повнотою і точністю розпізнавання і класифікації, але й довів, що нейронна мережа не обов’язково повинна займати багато місця на диску, потребляти багато ресурсів і довго й нестерпно навчатися і займати дорогоцінний час

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bishop, C.M. Neural Networks for Pattern Recognition – Oxford University Press, 1995. – 498 p.
2. Хайкин, С. Нейронные сети: полный курс / С. Хайкин. – М.:Вильямс, 2006. – 1104 с.
3. D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Convolutional neural network committees for handwritten character classification. In International Conference on Document Analysis and Recognition, pages 1250–1254, 2011.
4. D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In International Joint Conference on Artificial Intelligence, pages 1237–1242, 2011.
5. Kaggle - [Електронний ресурс] система організації конкурсів з дослідження даних, а також соціальна мережа фахівців з обробки даних та машинного навчання. Спосіб доступу - https://www.kaggle.com/
6. James Atwood and Don Towsley. Search-Convolutional Neural Networks. CoRR, abs/1511.02136, 2015.
7. Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. CoRR, abs/1106.1813, 2011.
8. Charu Aggarwal and Karthik Subbian. Evolutionary Network Analysis: A Survey. ACM Comput. Surv., 47(1):10:1—-10:36, may 2014.
9. Fran¸cois Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv preprint arXiv:1610.02357, 2016.
10. Jaeyong Chung and Taehwan Shin. Simplifying deep neural networks for neuromorphic architectures. In Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE, pages 1–6.IEEE, 2016.

11. A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Computer Science Department, University of Toronto, 2009.
12. Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Advances in Neural Information Processing Systems, pages 1269–1277, 2014.

ДОДАТОК А

Відомості матеріалів дипломного проекту

		Позначення	Найменування	Кіль–кість аркушів	Примітки
1					
2			Документація		
3					
4	А4	**ІСТ.КР 19.01.ДА.ПЗ**	Пояснювальна записка	82	
5					
6			Електронний носій інформації		
7					
8			Диск DVD–R з презентацією	2	

Зм	Лист	№ докум	Підпис	Дата	**ІСТ.КР 19.01.ДА.ПЗ**			
Розроб.		Курочкін Є.Г.			**Матеріали дипломного проекту**	Літ.	Лист	Аркуш–ів
						Н	1	1
Керівник		Гнатушенко В.В.				НТУ «ДП», 12; 122м-18-1		
Н.контр.		Коротенко Г.М.						
Зав.каф.		Бусигін Б.С.						

ДОДАТОК Б

Код програми

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

from time import time
from keras.applications import InceptionV3, VGG16
from keras.models import Model, load_model
from keras.layers import (Input, Dense, Flatten,
                          Dropout, BatchNormalization,
                          GlobalAveragePooling2D,
                          SeparableConv2D,
                          Conv2D, MaxPool2D)
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
from sklearn.metrics import accuracy_score, confusion_matrix
from mlxtend.plotting import plot_confusion_matrix


SEED = 232
PATH_TO_DATA = 'data/'
IMG_WEIGHT = 150
IMG_HEIGHT = 150
EPOCHS = 10
BATCH_SIZE = 32


def get_image_data_from_directories(img_w, img_h, batch_size, path_to_data):

    # Генеруємо об'єкти для наших даних
    train_datagen = ImageDataGenerator(rescale=1. / 255,
                                       zoom_range=0.3,
                                       vertical_flip=True)
    test_val_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
# Змінна яка подаватиме в модель дані у вказаних розмірах партії та розмірах зображення
train_generate_image = train_datagen.flow_from_directory(
    directory=path_to_data + 'train',
    target_size=(img_w, img_h),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True)

test_generate_image = test_val_datagen.flow_from_directory(
    directory=path_to_data + 'test',
    target_size=(img_w, img_h),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True)

# Отримаємо тестові данні та генеруємо масив із їх відношенням до первого класу
test_data = []
test_labels = []


for dircts in ['/healthy/', '/pneumonia/']:
    for image in (os.listdir(path_to_data + 'test' + dircts)):
        image = plt.imread(path_to_data + 'test' + dircts + image)
        image = cv2.resize(image, (img_w, img_h))
        image = np.dstack([image, image, image])
        image = image.astype('float32') / 255
        if dircts == '/healthy/':
            label = 0
        elif dircts == '/pneumonia/':
            label = 1
        test_data.append(image)
        test_labels.append(label)

test_data = np.array(test_data)
test_labels = np.array(test_labels)

return train_generate_image, test_generate_image, test_data, test_labels
```

```
def create_my_cnn_model(img_w, img_h):

    # Вхідний шар
    inputs = Input(shape=(img_w, img_h, 3))


    # Створення першого згорткового шару
    m = Conv2D(filters=16, kernel_size=(3, 3),
               activation='relu', padding='same')(inputs)
    m = Conv2D(filters=16, kernel_size=(3, 3),
               activation='relu', padding='same')(m)
    m = MaxPool2D(pool_size=(2, 2))(m)


    # Створення другого згорткового шару
    m = SeparableConv2D(filters=32, kernel_size=(3, 3),
                        activation='relu', padding='same')(m)
    m = SeparableConv2D(filters=32, kernel_size=(3, 3),
                        activation='relu', padding='same')(m)
    m = BatchNormalization()(m)
    m = MaxPool2D(pool_size=(2, 2))(m)


    # Створення третього згорткового шару
    m = SeparableConv2D(filters=64, kernel_size=(3, 3),
                        activation='relu', padding='same')(m)
    m = SeparableConv2D(filters=64, kernel_size=(3, 3),
                        activation='relu', padding='same')(m)
    m = BatchNormalization()(m)
    m = MaxPool2D(pool_size=(2, 2))(m)


    # Створення четвертого згорткового шару
    m = SeparableConv2D(filters=128, kernel_size=(3, 3),
                        activation='relu', padding='same')(m)
    m = SeparableConv2D(filters=128, kernel_size=(3, 3),
                        activation='relu', padding='same')(m)
```

```
    m = BatchNormalization()(m)
    m = MaxPool2D(pool_size=(2, 2))(m)
    m = Dropout(rate=0.2)(m)


    # Створення п'ятого згорткового шару
    m = SeparableConv2D(filters=256, kernel_size=(3, 3),
                        activation='relu', padding='same')(m)
    m = SeparableConv2D(filters=256, kernel_size=(3, 3),
                        activation='relu', padding='same')(m)
    m = BatchNormalization()(m)
    m = MaxPool2D(pool_size=(2, 2))(m)
    m = Dropout(rate=0.2)(m)


    # Створення повністю пов'язанного шару
    m = Flatten()(m)
    m = Dense(units=512, activation='relu')(m)
    m = Dropout(rate=0.7)(m)
    m = Dense(units=128, activation='relu')(m)
    m = Dropout(rate=0.5)(m)
    m = Dense(units=64, activation='relu')(m)
    m = Dropout(rate=0.3)(m)


    # Вихідний шар
    output = Dense(units=1, activation='sigmoid')(m)

    return inputs, output


def create_cnn_base_inception_v3(img_w, img_h):

    # Генерація моделі на основі InceptionV3
    base_model = InceptionV3(weights=None, include_top=False,
                             input_shape=(img_w, img_h, 3))
    m = base_model.output
    m = Dropout(0.5)(m)
```

```
    m = GlobalAveragePooling2D()(m)
    m = Dense(128, activation='relu')(m)
    m = BatchNormalization()(m)

    # Вихідний шар
    output = Dense(units=1, activation='sigmoid')(m)

    base_model.load_weights("inception_v3_weights.h5")
    inputs = base_model.input

    return inputs, output


def create_cnn_base_vgg16(img_w, img_h):

    # Генерація моделі на основі VGG16
    base_model = VGG16(weights=None, include_top=False,
                       input_shape=(img_w, img_h, 3))
    x = base_model.output
    x = Dropout(0.5)(x)
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    x = BatchNormalization()(x)

    # Вихідний шар
    output = Dense(units=1, activation='sigmoid')(x)

    base_model.load_weights("vgg16_weights.h5")
    inputs = base_model.input

    return inputs, output


def show_plot_accuracy_current_model(history):
    # Відображення на графі даних про Точність моделі під час навчання
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
```

```
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['train', 'test'])
    plt.show()


def show_plot_loss_current_model(history):
    # Відображення на графі даних про Втрати моделі під час навчання
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['train', 'test'])
    plt.show()


def get_metrics_about_current_model(test_data, test_labels, model, history):

    test_data_list = [
        np.concatenate((test_data[:62], test_data[-62:])),
        np.concatenate((test_data[:125], test_data[-124:])),
        np.concatenate((test_data[:187], test_data[-186:])),
        np.concatenate((test_data[:249], test_data[-248:])),
        test_data
    ]

    test_labels_list = [
        np.concatenate((test_labels[:62], test_labels[-62:])),
        np.concatenate((test_labels[:125], test_labels[-124:])),
        np.concatenate((test_labels[:187], test_labels[-186:])),
        np.concatenate((test_labels[:249], test_labels[-248:])),
        test_labels
    ]

    model_metrics_data_dict = {
        "accuracy": [],
        "precision": [],
```

```
        "recall": [],
        "F1-score": [],
        "time": [],
        "train_accuracy": []

    }

    for i in range(len(test_data_list)):
        t1 = time()
        preds = model.predict(test_data_list[i])
        t1 = time() - t1
        model_metrics_data_dict["time"].append(t1 / len(test_labels_list[i]))
        model_metrics_data_dict["accuracy"].append(
            accuracy_score(test_labels_list[i], np.round(preds)) * 100)
        cm = confusion_matrix(test_labels_list[i], np.round(preds))

        tn, fp, fn, tp = cm.ravel()

        precision = tp / (tp + fp) * 100
        recall = tp / (tp + fn) * 100
        model_metrics_data_dict["precision"].append(precision)
        model_metrics_data_dict["recall"].append(recall)
        model_metrics_data_dict["F1-score"].append(
            2 * precision * recall / (precision + recall))

    model_metrics_data_dict["train_accuracy"] = \
        np.round((history.history['accuracy'][-1]) * 100, 2)
    return model_metrics_data_dict


if __name__ == "__main__":
    # Задаємо константу для генерації випадкових чисел
    np.random.seed(SEED)
    tf.compat.v1.set_random_seed(SEED)

    train_gen, test_gen, test_data, test_labels = \
        get_image_data_from_directories(IMG_WEIGHT, IMG_HEIGHT, BATCH_SIZE, PATH_TO_DATA)
```

```
while True:
    choose = input("Будь ласка, введіть номер категорії в залежності від бажаної нейронної мережі:\n\t"
                   "1 - Розробляєма згорткова нейронна мережа\n\t"
                   "2 - Нейронна мережа на базі Inception-V3\n\t"
                   "3 - Нейронная мережа на базі VGG16\n")
    if choose == "1":
        inputs, output = create_my_cnn_model(IMG_WEIGHT, IMG_HEIGHT)
        break
    elif choose == "2":
        inputs, output = create_cnn_base_inception_v3(IMG_WEIGHT, IMG_HEIGHT)
        break
    elif choose == "3":
        inputs, output = create_cnn_base_vgg16(IMG_WEIGHT, IMG_HEIGHT)
        break
    else:
        print("Ви обрали хибний варіант, спробуйте ще")
        continue

# Створення моделі та компіляція її
model = Model(inputs=inputs, outputs=output)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Створення колбєку
checkpoint = ModelCheckpoint(filepath='best_weights.hdf5',
                             save_best_only=True, save_weights_only=True)
lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.3,
                              patience=2, verbose=2, mode='max')
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1,
                           patience=1, mode='min')

# Навчання моделі
history = model.fit_generator(
    train_gen, steps_per_epoch=train_gen.samples // BATCH_SIZE,
    epochs=EPOCHS, validation_data=test_gen,
    validation_steps=test_gen.samples // BATCH_SIZE,
    callbacks=[checkpoint, lr_reduce])

show_plot_accuracy_current_model(history)
```

```
show_plot_loss_current_model(history)

# Тестуємо дану модель на тестових даних і створюємо матрицю плутанини
preds = model.predict(test_data)

acc = accuracy_score(test_labels, np.round(preds)) * 100
cm = confusion_matrix(test_labels, np.round(preds))

fig, ax = plot_confusion_matrix(conf_mat=cm, figsize=(5, 5))
plt.show()

# Отримуємо метрики щодо моделі
model_metrics = get_metrics_about_current_model(test_data, test_labels, model, history)
```

ДОДАТОК В

ВІДГУК

на дипломну роботу магістра

" Класифікація цифрових зображень з використанням нейронних мереж"

студента групи 122м-18-1 Курочкіна Євгенія Геннадійовича

1. Метою дипломної роботи є дослідження різновидів нейронних мереж та реалізація найбільш відповідної архітектури для розпізнавання цифрових зображень

2. Завдання та зміст дипломної роботи відповідає основній меті – оцінці знань і ступеня підготовленості студента за спеціальністю 122 "Комп'ютерні науки".

3. Тема роботи представляється актуальною, оскільки створення автоматизованого інструментарію інформаційної технології для класифікації цифрових зображень являє собою високу актуальність у безлічі сфер діяльності у сучасному світі, таких як медицина, наука, біологія та інші.

4. Для досягнення мети дипломної роботи Курочкіним Є.Г. запропоновано метод згорткових нейронних мереж, що враховує створення 5-х блоків, складених із згорткових шарів та шарів пулінгу, які розташовуються один за одним, а завершується блоком повністю пов'язаних шарів.

5. Оформлення пояснювальної записки виконано, в основному, відповідно до діючих стандартів і нормативних вимог.

6. У якості зауваження слід відзначити недостатнє порівняння отриманих результатів дослідження з існуючими аналогами.

Незважаючи на зазначений недолік, дипломна робота заслуговує оцінки "_____".

Керівник,
д.т.н., професор кафедри ІСТ В.В. Гнатушенко

ДОДАТОК Г

РЕЦЕНЗІЯ

на дипломну роботу магістра

" Класифікація цифрових зображень з використанням нейронних мереж "

студента групи 122м-18-1 Курочкіна Євгенія Геннадійовича

1. Тема дипломного проекту, присвячена класифікації цифрових зображень з використанням нейронних мереж, є актуальною на даний момент і спрямована на застосування у різних сферах діяльності людини.

2. Рецензована робота спрямована на створення згорткової нейронної мережі, а саме створення модулю для класифікації цифрових зображень, навченого на виборці із більш ніж 5000 екземплярів декількох класів.

3. Практична значимість результатів дипломної роботи полягає в автоматизації етапу діагностування в людини такого захворювання як пневмонія легень, для чого необхідно мати лише тільки актуальний цифровий рентгенологічний знімок. В тому числі, дана архітектура дозволяє с невеликою поправкою налаштування використовувати її і для інших завдань класифікації цифрових зображень.

Дипломна робота цілком відповідає вимогам, що пред'являються до кваліфікаційних робіт рівня магістра.

Недолік: недостатня повнота опису результатів практичних досліджень в зоні їх застосування.

Незважаючи на зазначений недолік, дипломна робота в цілому може бути відзначена оцінкою "_____".

Рецензент,

Довідка

про результати перевірки тексту кваліфікаційної роботи магістра
на тему: Класифікація цифрових зображень з використанням нейронних мереж
12 "Інформаційні технології".
(назва за наказом ректора)

студента групи 122м-18-1
(шифр групи)

Курочкіна Євгенія Геннадійовича
(прізвище, ім'я, по батькові)

Зазначена робота перевірялася комп'ютерною програмою перевірки плагіату «ETXT».

За результатами перевірки порушень не знайдено.

Керівник кваліфікаційної роботи: _____.
(П.І.Б.)

Нормоконтролер _Коротенко Г.М.__
(П.І.Б.)

Завідувач кафедри ІСТ __Бусигін Б.С.____
(П.І.Б.)

23.12.2019 р.