НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ННК «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Системного проектування
(повна назва кафедри)

«На правах рукопису»
УДК 004.853

«До захисту допущено»
Завідувач кафедри
_____ _____
(підпис) (ініціали, прізвище)
“___”_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності: 8.05010102 “Інформаційні технології проектування”
(код і назва)

на тему: Дослідження і розробка обчислювальної мережі в концепції Інтернет речей з використанням штучних нейронних мереж

Виконав: студент __6__ курсу, групи ___ДА-51м_____
(шифр групи)

Попеляєв Денис Павлович
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник Харченко Костянтин Васильович, к.т.н., доц.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2017 року

Національний технічний університет України
«Київський політехнічний інститут»

Інститут (факультет) ННК «ІПСА»
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – другий (магістерський)

Спеціальність 8.05010102 “Інформаційні технології проектування”
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ _____
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Попеляєв Денис Павлович
(прізвище, ім’я, по батькові)

1. Тема дисертації Дослідження і розробка обчислювальної мережі в концепції Інтернет речей з використанням штучних нейронних мереж
науковий керівник дисертації Харченко Костянтин Васильович, к.т.н., доц.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом дисертації

3. Об’єкт дослідження Поєднання технологій Machine Learning та Internet of things.

4. Предмет дослідження Оптимізація процесу навчання штучної нейронної мережі для задоволення особливостей роботи в технології інтернету речей через зменшення необхідної кількості навчальних даних.

5. Перелік завдань, які потрібно розробити

Дослідження існуючих матеріалів в галузі machine learning та IoT.

Опис підходів до створення та обробки начальної множини для штучних нейронних мереж

Розробка тестових платформ для оцінки вибраного підходу

Тестування на різних архітектурах штучних нейронних мереж, оцінка результатів

Підбивання висновків щодо можливостей використання запропонованого підходу в технології Internet of Things

6. Орієнтовний перелік ілюстративного матеріалу: презентація на тему: «Дослідження методів створення та обробки навчальних даних для штучних нейронних мереж для використання в технології інтернету речей»

7. Орієнтовний перелік публікацій

Popeliaev D.P. Methods of training data augmentation for artificial neural networks / Popeliaev D.: International Scientific Journal “Intarnauka”, May 2017.

Popeliaev D.P. Approaches to fire recognition on image. Convolutional neural networks / Popeliaev D.: міжнародна науково-технічна конференція «САІТ-2017», 22 травня 2017, Київ, Україна: матеріали. – К. :НТУУ «КПІ ім. І. Сікорського», 2017. – С. 315-316

8. Консультанти розділів дисертації[1]*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	30.09.2016	
2	Збір інформації	17.01.2017	
3	Розробка алгоритму і програми	05.03.2017	
4	Розробка плану оцінювання	16.04.2017	
5	Розробка програмних моделей	14.05.2017	
6	Тестування підходу на програмних моделях	28.05.2017	
7	Отримання допуску до захисту та подача роботи в ДЕК	12.06.2017	

Студент _____ ___Д.П._Попеляєв____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____
___К.В._Харченко____
(підпис) (ініціали, прізвище)

[1] Консультантом не може бути зазначено наукового керівника магістерської дисертації.

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Дослідження і розробка обчислювальної мережі в концепції Інтернет речей з використанням штучних нейронних мереж

студентом: Попеляєвим Денисом Павловичем

Робота виконана на 120 сторінках, містить 31 ілюстрацію, 31 таблицю. При підготовці використовувалася література з 32 різних джерел.

Актуальність

Технологія машинного навчання, як і інтернету речей вже є досить популярними та все більш використовуються в різноманітних областях. Велика кількість задач може успішно вирішуватися за допомогою нейронних мереж, а технологія IoT має значний потенціал та постійно розвивається.

Актуальність роботи полягає в тому, що оптимізація навчання нейронної мережі в технології IoT значно зменшить необхідні затрачувані ресурси, які в контроллерах IoT зазвичай досить обмежені.

Мета

Метою цієї дипломної роботи є дослідження підходів до зменшення необхідної кількості навчальних даних для нейронних мереж різних архітектур для полегшення їх використання в технології Інтернету речей.

Завдання

Для досягнення поставленої мети необхідне рішення наступних завдань:

- Дослідження існуючих матеріалів в галузі machine learning та IoT.
- Опис підходів до створення та обробки начальної множини для штучних нейронних мереж
- Розробка тестових платформ для оцінки вибраного підходу
- Тестування на різних архітектурах штучних нейронних мереж, оцінка результатів
- Підбивання висновків щодо можливостей використання запропонованого підходу в технології Internet of Things

Об’єкт досліджень

У відповідності до поставленої мети об'єктом досліджень обрано поєднання технологій Machine Learning та Internet of things.

Предмет досліджень

Оптимізація процесу навчання штучної нейронної мережі для задоволення особливостей роботи в технології інтернету речей через зменшення необхідної кількості навчальних даних.

Наукова новизна

Наукова новизна роботи полягає в використанні алгоритму штучного збільшення навчальних даних для нейронних мереж при роботі на пристроях Internet of Things.

Практична цінність

В якості результату розроблено програмне забезпечення для штучного збільшення навчальних даних для нейронних мереж. Для тестування було виконано навчання нейронних мереж з різною архітектурою на отриманій штучній вибірці. Результат даної роботи можна використовувати в подальшому для виконання штучного збільшення навчальної вибірки для рішення інших задач, особливо в концепції IoT.

Ключові слова

Машинне навчання, Інтернет речей, штучні нейронні мережі, навчальна множина, аугментація навчальних даних.

РЕФЕРАТ

на магистерскую диссертацию

выполненную на тему: Исследование и разработка вычислительной сети в концепции Интернет вещей с использованием искусственных нейронных сетей

студентом: Попеляевым Денисом Павловичем

Работа выполнена на 120 страницах, содержит 31 иллюстрацию, 31 таблицу. При подготовке использовалась литература из 32 различных источников.

Актуальность

Технология машинного обучения, как и интернета вещей уже достаточно популярны и все более используются в различных областях. Большое количество задач может успешно решаться с помощью нейронных сетей, а технология IoT имеет значительный потенциал и постоянно развивается.
Актуальность работы заключается в том, что оптимизация обучения нейронной сети в технологии IoT значительно уменьшит необходимые затрачиваемые ресурсы, которые в контроллерах IoT обычно весьма ограничены.

Цель

Целью настоящей дипломной работы является исследование подходов к уменьшению необходимого количества учебных данных для нейронных сетей различных архитектур для облегчения их использования в технологии Интернета вещей.

Задача

Для достижения поставленной цели необходимо решение следующих задач:

- Исследование существующих материалов в области machine learning и IoT.
- Описание подходов к созданию и обработки учебного множества для искусственных нейронных сетей
- Разработка тестовых платформ для оценки выбранного подхода

- Тестирование на различных архитектурах искусственных нейронных сетей, оценка результатов
- Подведение выводов относительно возможностей использования предложенного подхода в технологии Internet of Things

Объект исследования

В соответствии с поставленной целью объектом исследований выбрано сочетание технологий Machine Learning и Internet of things.

Предмет исследования

Оптимизация процесса обучения искусственной нейронной сети для удовлетворения особенностей работы в технологии интернета вещей из-за уменьшения необходимого количества учебных данных.

Научная новизна

Научная новизна работы заключается в использовании алгоритма искусственного увеличения учебных данных для нейронных сетей при работе на устройствах Internet of Things.

Практическая ценность

В качестве результата разработано программное обеспечение для искусственного увеличения учебных данных для нейронных сетей. Для тестирования было выполнено обучение нейронных сетей с различной архитектурой на полученной искусственной выборке. Результат данной работы можно использовать в дальнейшем для выполнения искусственного увеличения обучающей выборки для решения других задач, особенно в концепции IoT.

Ключевые слова

Машинное обучение, Интернет вещей, искусственные нейронные сети, обучающее множество, аугментация учебных данных.

ABSTRACT

on the master's thesis

on topic: Research and development of a computer network in the Internet of things concept using artificial neural networks.

student: Denis Popeliaev

Work carried out on 120 pages, containing 31 figures, 31 tables. The paper was written with references to 32 different sources.

Topicality

The technology of machine learning, like the Internet of things, is already quite popular and is increasingly being used in various fields. A large number of tasks can be successfully solved with the help of neural networks, and IoT technology has considerable potential and is constantly developing.

The relevance of the work lies in the fact that optimizing the training of a neural network in IoT technology will significantly reduce the required resources that are usually very limited in IoT controllers.

Purpose

The purpose of this thesis is to explore approaches to reducing the necessary amount of training data for neural networks of different architectures to facilitate their use in Internet technology of things.

Solution

To achieve research goal, it is necessary to solve the following tasks:

- Examination of existing materials in the field of machine learning and IoT.
- Description of approaches for creating and processing training set for artificial neural networks
- Development of test platforms for evaluating the selected approach
- Testing on various architectures of artificial neural networks, evaluation of results
- Conclusion on the possibilities of using the proposed approach in Internet of Things

The object of research

In accordance with the research goal, the object of research is a combination of Machine Learning and Internet of things.

The subject of research

Optimization of the learning process of an artificial neural network to meet the specifics of working in the technology of the Internet of things due to the reduction of the necessary amount of training data.

Scientific novelty

The scientific novelty of the work is to use the algorithm of artificial augmentation of training dataset for neural networks when working on Internet of Things devices.

The practical value of research

As a result, software was developed to artificially increase learning data for neural networks. For testing, neural networks with different architectures were trained on the obtained artificial sample. The result of this work can be used in the future to perform an artificial augmentation of the training samples for solving other problems, especially in the IoT concept.

Keywords

Machine learning, the Internet of things, artificial neural networks, training dataset, training data augmentation.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

IoT	Internet of things, інтернет речей, концепція мережі з розумних речей з вбудованими технологіями для комцнікації один з одним і зовнішнім світом
CNN	convolutional neural network, одна з найпопулярніших архітектур нейронних мереж для розпізнавання зображень
MLP	Multilayer perceptron, багатошаровий персептрон
dataset	набір даних для навчання нейронної мережі
MNIST	Mixed National Institute of Standards and Technology, об’ємна база даних зразків рукописних цифр
Python	високорівнева мова програмування загального призначення
PyCharm	інтегроване середовще розробки для мови програмування Python
ОС	операційна система
Arduino	апаратна обчислювальна платформа для аматорського конструювання, основними компонентами якої є плата мікроконтролера
Linux	загальна назва UNIX-подібних операційних систем на основі однойменного ядра
Skimage	бібліотека для мови програмування Python для просунотої роботи з зображеннями
Matplotlib	бібліотека для мови програмування Python для просунотої роботи з графічними об’єктами

ВСТУП

В останні роки великий інтерес викликає проблематика нейроних мереж. Цей напрямок відноситься до наукової області, обумовленої в англомовній літературі терміном Machine Learning. Людей завжди цікавило їхнє власне мислення. Це самопитання, думання мозку самого про себе є, можливо, відмітною рисою людини. Нейробіологи і нейроанатоми досягли в цій області значного прогресу. Ретельно вивчаючи структуру і функції нервової системи людини, вони багато чого зрозуміли в структурі мозку, але мало довідалися про його функціонування. У процесі нагромадження ними знань з'ясувалося, що мозок має приголомшуючу складність. Сотні мільярдів нейронів, кожний з яких з'єднаний із сотнями або тисячами інших, утворюють систему, що далеко перевершує наші самі сміливі мрії про суперкомп'ютери.

На сьогоднішній день існують дві взаємно збагачуючі одна одну мети нейронного моделювання: перша – зрозуміти функціонування нервової системи людини на рівні фізіології і психології і друга – створити обчислювальні системи (штучні нейроні мережі), що виконують функції, подібні до функцій мозку. Нейроні мережі можна розглядати як сучасні обчислювальні системи, що перетворять інформацію з образа процесів, що відбуваються в мозку людини.

Оброблювана інформація має числовний характер, що дозволяє використовувати нейрону мережу, наприклад, як модель об'єкта з зовсім невідомими характеристиками. Інші типові додатки нейроних мереж охоплюють задачі розпізнавання, класифікації, аналізу образів. Понад 80 % усіх додатків нейроних мереж відноситься до так називаних багатошарових мереж без зворотних зв'язків. У них сигнал пересилається в напрямку від вхідного шару через сховані шари (якщо вони існують) до вихідного шару. Інтелектуальні системи на основі штучних нейроних мереж дозволяють з успіхом розв’язувати проблеми розпізнавання образів, виконання прогнозів, оптимізації, асоціативній пам'яті і керування.

Традиційні підходи до розв’язання цих проблем не завжди дають необхідну гнучкість і багато додатків виграють від використання нейромереж. Штучні нейромережи є електронними моделями нейроної структури мозку, який, головним чином, вчиться на досвіді. Природна аналогія доводить, що множина проблем, що не піддаються розв’язанню традиційними комп'ютерами, можуть бути ефективно вирішені за допомогою нейромереж. Тривалий період еволюції додав мозкові людини багато якостей, відсутніх у сучасних комп'ютерах з архітектурою фон Неймана.

До них відносяться:

- розподілене представлення інформації і паралельні обчислення
- здатність до навчання й узагальнення
- адаптивність
- толерантність до помилок
- низьке енергоспоживання

Системи, побудовані на принципах біологічних нейронів, мають перераховані характеристики, які можна вважати істотним досягненням в індустрії обробки даних. Біологічний мозок розглядається як множина елементарних елементів — нейронів, з'єднаних один з одним численними зв'язками. Нейрони бувають трьох типів: рецептори (приймаючі сигнали з зовнішнього середовища і передавальні іншим нейронам), внутрішні нейрони (приймаючі сигнали від інших нейронів, що перетворюють їх і передають іншим нейронам) і реагуючі нейрони (приймаючі сигнали від нейронів і сигнали, що виробляють, у зовнішнє середовище).

Як і обчислювальні штучні нейронні мережі, є й інші технології, що продовжують невпинно набирати популярність. Особливого поширення набула технологія Інтернету речей, основною концепцією якої є можливість підключати різноманітні об’єкти (речі) до мережі, обробляти інформацію, що надходить з навколишнього середовища, обмінюватися нею і виконувати різні дії залежно від отриманої інформації. Інтернет речей вважають наступним етапом технічної революції. Вона стосується зміни побуту, виробництва,

мобільних пристроїв й індустріальної галузі. Важливими функціями цієї концепції є полегшення повсякденного життя, підвищення ефективності та якості роботи, енергозаощадження тощо. Тенденції до зацікавлення Інтернетом речей спостерігаються і в Україні.

Сучасна концепція Інтернету речей передбачає комунікацію об’єктів, які використовують технології для взаємодії між собою та з навколишнім середовищем. Ця концепція дає змогу пристроям виконувати певні дії без втручання людини. Отже, усі пристрої в будинках, в автомобілях та інших системах інфраструктури повинні виконувати обробку інформації, її аналіз та здійснювати обмін між собою і залежно від результатів приймати рішення та виконувати певні дії. Експерти стверджують, що Інтернет речей є однією з найперспективніших технологій останніх років, що вже сьогодні фактично створює сотні нових продуктів і приводить до появи нових компаній на ринку, які диктують свої умови IT-гігантам. Споживач не зауважує, що він та його друзі чи колеги вже не перший рік кожного дня користуються такими пристроями. Більше того, у багатьох українських домівках вже встановлені системи “розумного будинку”, в які інтегровані десятки сенсорів. Переваги Інтернету речей, які вже доступні і які ще в процесі розробки, демонструють безліч реальних прикладів, тим паче, що сфер використання цієї технології чимало.

Саме на перетині технологій інтернету речей та штучних нейронних мереж і розгортується дослідження у цій роботі. Так як в мережах Інтернету речей задіяні контролери, що зазвичай мають досить обмежені ресурси, слід коригувати та оптимізовувати розрахунки, здійснювані нейронними мережами, в сторону зменшення використовуваних ресурсів. Особлива увага в дослідженні приділяється саме аспектам оптимізації роботи з нейронною мережею на усіх етапах. Обмеження в доступній флеш-пам’яті призводить до того, що навчання необхідно проводити на мінімально доступній вибірці, намагаючись отримати задовільні результати.

1 СТВОРЕННЯ МНОЖИНИ НАВЧАЛЬНИХ ДАНИХ ДЛЯ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

За останні роки машинне навчання не тільки набрало популярності, а й сформувалося як окрема дисципліна з досліджень розпізнавання образів та теорії обчислювального навчання в галузі штучного інтелекту. Репрезентативні вхідні дані високої якості є ключем до якісних моделей машинного навчання, а їх дефіцит може перешкоджати розвитку моделі. Формування множини навчальних даних має принципово важливе значення для успішного вирішення завдань машинного навчання. Часто завдання машинного навчання зводяться саме до правильного формування навчальної множини. Помилки у формуванні навчальної множини зазвичай виявляються критичними і здатні звести нанівець ефективність самих алгоритмів навчання. Серед фахівців по машинному навчанню загальновизнаним вважається, що наявність якісних навчальних даних часто набагато важливіше якості алгоритму навчання. У зв'язку з активним розвитком глибоких нейронних мереж в останнє десятиліття питання формування множини навчальних даних приймає особливо важливе значення, оскільки в багатьох задачах глибокі нейронні мережі демонструють якість, що істотно перевершує інші алгоритми, однак, щоб отримати подібний виграш в якості, необхідно використовувати навчальну множину дуже великого розміру (до декількох мільйонів зображень, при цьому навчання вимагає великого обсягу обчислювальних ресурсів і може займати кілька тижнів на багатопроцесорному кластері), а також спеціальні методи розширення та імітації розширення навчальної множини, які будуть розглянуті далі. У той же час, в сучасній літературі по машинному навчанню, питанню формування навчальної множини приділяється недостатня увага, недостатньо розвинена теоретична база, що пояснює явища, що виникають в процесі формування множини навчальних даних.

Найпершим та одним з найважливіших етапів навчання нейронної мережі є збір даних, які сформують навчальну множину. Навчальні множини повинні бути досить великими, щоб містити всю необхідну інформацію для виявлення

важливих особливостей і зв'язків. Але і навчальні приклади повинні містити широке різноманіття даних. Якщо мережа навчається лише для одного прикладу, ваги старанно встановлені для цього прикладу, радикально змінюються у навчанні для наступного прикладу. Попередні приклади при навчанні наступних просто забуваються. В результаті система повинна навчатись всьому разом, знаходячи найкращі вагові коефіцієнти для загальної множини прикладів. Наприклад, у навчанні системи розпізнавання піксельних образів для десяти цифр, які представлені двадцятьма прикладами кожної цифри, всі приклади цифри "сім" не доцільно представляти послідовно [13]. Краще надати мережі спочатку один тип представлення всіх цифр, потім другий тип і так далі. Головною компонентою для успішної роботи мережі є представлення і кодування вхідних і вихідних даних. Штучні мережі працюють лише з числовими вхідними даними, отже, необроблені дані, що надходять із зовнішнього середовища повинні перетворюватись. Додатково необхідне масштабування, тобто нормалізація даних відповідно до діапазону всіх значень. Попередня обробка зовнішніх даних, отриманих за допомогою сенсорів, у машинний формат спільна для стандартних комп'ютерів і є легко доступною. Якщо після контрольованого навчання нейромережа ефективно опрацьовує дані навчальної множини, важливим стає її ефективність при роботі з даними, які не використовувались для навчання. У випадку отримання незадовільних результатів для тестової множини, навчання продовжується. Тестування використовується для забезпечення запам'ятовування не лише даних заданої навчальної множини, але і створення загальних образів, що можуть міститись в даних.

При підготовці даних для навчання нейронної мережі необхідно звертати увагу на декілька суттєвих моментів. Кількість спостережень в наборі даних. Слід враховувати той фактор, що чим більше розмірність даних, тим більше часу буде потрібно для навчання мережі. Робота з викидами. Слід визначити наявність викидів і оцінити необхідність їх присутності в вибірці. Навчальна вибірка повинна бути представницької (репрезентативною). Навчальна вибірка

не повинна містити протиріч, так як нейронна мережа однозначно зіставляє вихідні значення вхідним.

Нейронна мережа працює тільки з числовими вхідними даними, тому важливим етапом при підготовці даних є перетворення і кодування даних. При використанні на вхід нейронної мережі слід подавати значення з того діапазону, на якому вона навчалася. Наприклад, якщо при навчанні нейронної мережі на один з її входів подавалися значення від 0 до 10, то при її застосуванні на вхід слід подавати значення з цього ж діапазону або прилеглі. Існує поняття нормалізації даних. Метою нормалізації значень є перетворення даних до виду, який найбільш підходить для обробки, тобто дані, що надходять на вхід, повинні мати числовий тип, а їх значення повинні бути розподілені в певному діапазоні. Нормалізатор може призводити дискретні дані до набору унікальних індексів або перетворювати значення, що лежать в довільному діапазоні, в конкретний діапазон, наприклад, [0..1]. Нормалізація виконується шляхом ділення кожної компоненти вхідного вектора на довжину вектора, що перетворює вхідний вектор в одиничний [18].

1.1 Передобробка даних

Після знайомства з базовими принципами нейромережевої обробки, можна приступати до застосувань отриманих знань для вирішення конкретних завдань. Перше, з чим стикається користувач будь-якого нейропакета - це необхідність підготовки даних для нейромережі. Майже в усіх матеріалах по роботі з нейронними мережами не торкаються цього, взагалі кажучи, непростого питання, мовчазно припускаючи, що дані для навчання вже є і представлені у вигляді, доступному для нейромережі. На практиці ж саме предобробка даних може стати найбільш трудомістким елементом нейромережевого аналізу. Причому, знання основних принципів і прийомів передобробки даних не менше, а може бути навіть більш важливе, ніж знання власне нейромережевих алгоритмів. Останні як правило, вже "зашиті" в різних

нейроемуляторах, доступних на ринку. Сам же процес вирішення прикладних завдань, в тому числі і підготовка даних, цілком лягає на плечі користувача. Спробуємо заповнити цю прогалину в описі технології нейромережевого аналізу.

Для початку випишемо з невеликими коментарями весь технологічний ланцюжок, тобто необхідні етапи нейромережевого аналізу:

- Кодування входів-виходів: нейромережі можуть працювати тільки з числами.
- Нормування даних: результати нейроаналізу не повинні залежати від вибору одиниць виміру.
- Передобробка даних: видалення очевидних регулярностей з даних полегшує нейромережі виявлення нетривіальних закономірностей.
- Навчання декількох нейромереж з різною архітектурою: результат навчання залежить як від розмірів мережі, так і від її початкової конфігурації.
- Відбір оптимальних мереж: тих, які дадуть найменшу помилку передбачення на невідомих поки даних.
- Оцінка значущості пророкувань: оцінка помилки прогнозів не менш важлива, ніж саме передбачене значення.

Зазвичай розглядають, в основному, останні етапи, пов'язані з навчанням власне нейромереж, але цьому розділі зосередимося на перших етапах нейромережевого аналізу - передобробці даних. Хоча перобробка не пов'язана безпосередньо з нейромережами, вона є одним з ключових елементів цієї інформаційної технології. Успіх навчання нейромережі може вирішальним чином залежати від того, в якому вигляді представлена інформація для її навчання.

Буде розглянута передобробка даних для навчання з учителем і, головним чином, буде виділено і проілюстровано на конкретних прикладах основний принцип такої передобробки: збільшення інформативності прикладів для підвищення ефективності навчання.

1.1.1 Кодування входів-виходів

На відміну від звичайних комп'ютерів, здатних обробляти будь-яку символьну інформацію, нейромережеві алгоритми працюють тільки з числами, бо їх робота базується на арифметичних операціях множення і складання. Саме таким чином набір синаптичних ваг визначає хід обробки даних.

Тим часом, не всяка вхідна або вихідна змінна в початковому вигляді може мати чисельне вираження. Відповідно, всі такі змінні слід закодувати - перевести в чисельну форму, перш ніж почати власне нейромережеву обробку. Розглянемо, перш за все основний керівний принцип, загальний для всіх етапів попередньої обробки даних.

1.1.2 Максимізація ентропії як мета передобробки

Припустимо, що в результаті переведення всіх даних в числову форму і подальшого нормування всі вхідні і вихідні змінні відображаються в одиничному кубі. Завдання нейромережевого моделювання - знайти статистично достовірні залежності між вхідними та вихідними змінними. Єдиним джерелом інформації для статистичного моделювання є приклади з навчальної вибірки. Чим більше біт інформації принесе кожен приклад - тим краще використовуються наявні в нашому розпорядження дані.

Розглянемо довільну компоненту нормованих (предоброблених) даних: $\tilde{x}_i$ Середня кількість інформації, принесеної кожним прикладом $\tilde{x}_i^a$, так само ентропії розподілу значень цієї компоненти $H(\tilde{x}_i)$. Якщо ці значення зосереджені у відносно невеликій області одиничного інтервалу, інформаційного змісту такої компоненти мало. У межі нульової ентропії, коли всі значення змінної збігаються, ця змінна не несе ніякої інформації. Навпаки, якщо значення змінної $\tilde{x}_i^a$ рівномірно розподілені в одиничному інтервалі, інформація такої змінної максимальна [32].

Загальний принцип передобробки даних для навчання, таким чином, полягає в максимізації ентропії входів і виходів. Цим принципом слід керуватися і на етапі кодування нечислових змінних.

1.1.3 Типи нечислових змінних

Можна виділити два основних типи нечислових змінних: впорядковані (звані також ординальні - від англ. Order - порядок) і категоріальні. В обох випадках змінна відноситься до одного з дискретного набору класів $\{c_1, \ldots, c_n\}$. Але в першому випадку ці класи впорядковані - їх можна ранжувати: $c_1 \succ c_2 \succ \ldots \succ c_n$, тоді як у другому така впорядкованість відсутня [30]. Як приклад упорядкованих змінних можна привести порівняльні категорії: погано - добре - відмінно, або повільно - швидко. Категоріальні змінні просто позначають один з класів, є іменами категорій. Наприклад, це можуть бути імена людей або назви кольорів: білий, синій, червоний.

1.1.4 Кодування ординальних змінних

Ординальні змінні ближчі до числової формі, тому що числовий ряд також впорядкований. Відповідно, для кодування таких змінних залишається лише поставити у відповідність номерам категорій такі числові значення, які зберігали б існуючу впорядкованість. Природно, при цьому є велика свобода вибору - будь-яка монотонна функція від номера класу породжує свій спосіб кодування. Яка ж з нескінченного різноманіття монотонних функцій - найкраща?

Відповідно до викладеного вище загального принципу, ми повинні прагнути до того, щоб максимізувати ентропію закодованих даних. При використанні сигмоідної функцій активації всі вихідні значення лежать в кінцевому інтервалі - зазвичай [0,1] або [-1,1]. З усіх статистичних функцій розподілу, визначених на кінцевому інтервалі, максимальну ентропію має рівномірний розподіл.

Що стосується даного випадку, кодування змінних числовими значеннями повинно призводити, по можливості, до рівномірного заповнення одиничного інтервалу закодованими прикладами. (Захоплюючи заодно і етап нормування) При такому способі "оцифровування" все приклади будуть нести приблизно однакове інформаційне навантаження.

Виходячи з цих міркувань, можна запропонувати наступний практичний рецепт кодування ординальних змінних. Одиничний інтервал розбивається на n відрізків - по кількості класів - з довжинами пропорційними числу прикладів кожного класу в навчальній вибірці:

$\Delta x_k = P_k/P$, де P_k - число прикладів класу k, а P - загальне число прикладів. Центр кожного такого відрізка буде чисельним значенням для відповідного ординального класу (див. Рис. 1.1) [30].

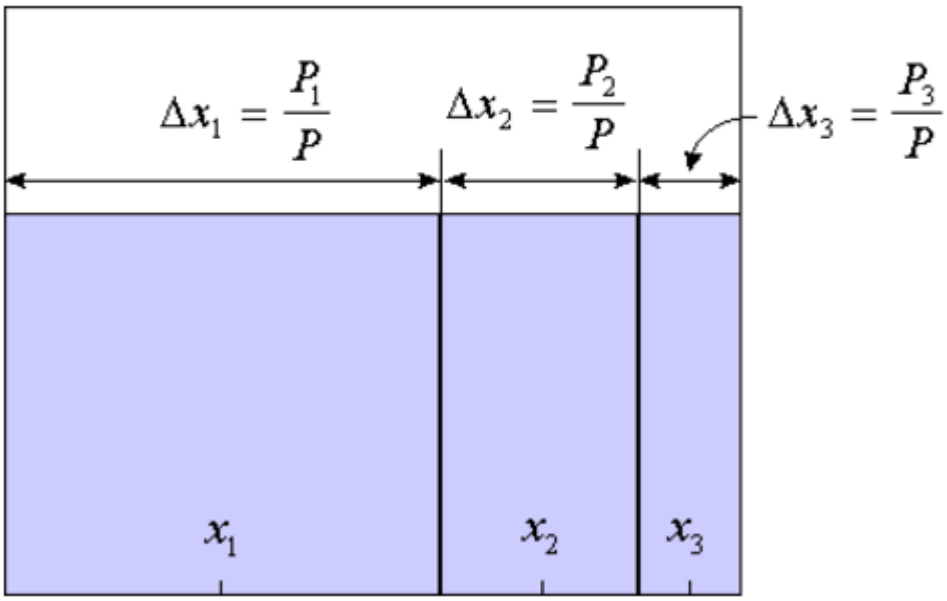


Рис 1.1 Ілюстрація способу кодування кардинальних змінних з урахуванням кількості прикладів кожної категорії.

1.1.5 Кодування категоріальних змінних

В принципі, категоріальні змінні також можна закодувати описаним вище способом, пронумерувавши їх довільним чином. Однак, таке нав'язування

неіснуючої впорядкованості тільки ускладнить вирішення завдання. Оптимальне кодування не повинно спотворювати структури співвідношень між класами. Якщо класи не впорядковані, така ж повинна бути і схема кодування.

Найбільш природною виглядає і найчастіше використовується на практиці двійкове кодування типу "n -> n", коли імена n категорій кодуються значеннями n бінарних нейронів, причому перша категорія кодується як (1,0,0,…,0) , друга, відповідно – (0,1,0,…,0) і т.д. аж до n-ної: (0,0,0,…,1). (Можна використовувати біполярну систему кодування, в якій нулі замінюються на "-1".) [30]. Легко переконатися, що в такому симетричному кодуванні відстані між усіма векторами-категоріями рівні.

Таке кодування, однак, не оптимально в разі, коли представлені класи істотно розрізняються числом прикладів. В цьому випадку, функція розподілу значень змінної вкрай неоднорідна, що істотно знижує інформативність цієї змінної. Тоді має сенс використовувати більш компактний, але симетричний код n -> m , коли імена n класів кодуються m-бітний двійковим кодом. Причому, в новому кодуванні активність кодуючих нейронів повинна бути рівномірна: мати приблизно однакове середнє за прикладами значення активації. Це гарантує однакову значимість ваг, що відповідають різним нейронам.

Як приклад розглянемо ситуацію, коли один з чотирьох класів (наприклад, клас C_1) якоїсь категоріальної змінної представлений набагато більшим числом прикладів, ніж інші: $P_1 \gg P_2 \sim P_3 \sim P_4$. . Просте кодування n -> n призвело б до того, що перший нейрон активувався б набагато частіше за інших. Відповідно, ваги нейронів що залишилися мали б менше можливостей для навчання. Цієї ситуації можна уникнути, закодувавши чотири класи двома бінарними нейронами наступним чином: $c_1 = (0,0), c_2 = (1,0), c_3 = (0,1), c_4 = (1,1)$, що забезпечує рівномірне "завантаження" кодуючих нейронів [31].

1.1.6 Відмінність між вхідними та вихідними змінними

Наприкінці даного розділу відзначимо одну істотну відмінність способів кодування вхідних і вихідних змінних, що випливає з визначення градієнта помилки. А саме, входи беруть участь в навчанні безпосередньо, тоді як виходи - лише опосередковано - через помилку верхнього шару. Тому при кодуванні категорій в якості вихідних нейронів можна використовувати як логістичну функцію активації $f(a) = 1/(e^{-a} + 1)$, визначену на відрізку [0,1], так і її антисиметричний аналог для відрізка [-1,1], наприклад: $f(a) = \tanh(a)$. При цьому кодування вихідних змінних з навчальної вибірки буде або {0,1} , або {-1,1} [31]. Вибір того чи іншого варіанту ніяк не позначиться на навчанні.

У випадку зі вхідними змінними справа йде по-іншому: навчання ваг нижнього шару мережі визначається безпосередньо значеннями входів: на них множаться невязки, що залежать від виходів. Тим часом, якщо з точки зору операції множення значення 1 та -1 рівноправні, між 0 і 1 є істотна асиметрія: нульові значення не дають ніякого вкладу в градієнт помилки. Таким чином, вибір схеми кодування входів впливає на процес навчання. В силу логічної рівноправності обох значень входів, більш кращою виглядає симетрична кодування: {-1,1}, яке зберігає цю рівноправність в процесі навчання.

Як входами, так і виходами нейромережі можуть бути абсолютно різнорідні величини. Очевидно, що результати нейромережевого моделювання не повинні залежати від одиниць виміру цих величин. А саме, щоб мережа трактувала їх значення одноманітно, всі вхідні і вихідні величини повинні бути приведені до єдиного одиничного масштабу. Крім того, для підвищення швидкості і якості навчання корисно провести додаткову передобробку даних, яка вирівнює розподіл значень ще до етапу навчання.

1.1.7 Індивідуальне нормування даних

Приведення даних до одиничного масштабу забезпечується нормуванням кожної змінної на діапазон розкиду її значень. У найпростішому варіанті це - лінійне перетворення:

$$\widetilde{x}_i = \frac{x_i - x_{i,\min}}{x_{i,\max} - x_{i,\min}}$$

в одиничний відрізок $\widetilde{x}_i \in [0, 1]$. Узагальнення для відображення вхідних даних в інтервал [-1,1], що рекомендується для вхідних даних, тривіально [31].

Лінійне нормування оптимальне, коли значення змінної щільно заповнюють певний інтервал. Але подібний "прямолінійний" підхід застосуємо далеко не завжди. Так, якщо в даних є відносно рідкісні викиди, які набагато перевищують типовий розкид (Рис. 1.2), саме ці викиди визначать згідно попередньої формули масштаб нормування. Це призведе до того, що основна маса значень нормованої змінної зосередиться поблизу нуля.

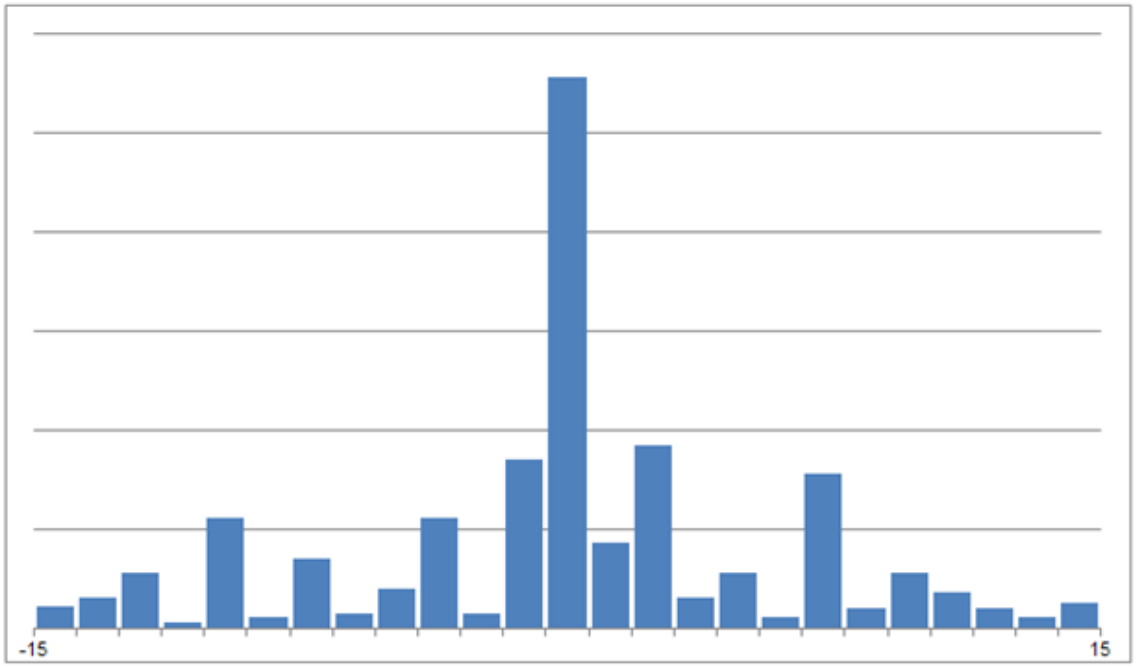


Рис 1.2 Гістограма значень змінної при наявності рідкісних, але великих за амплітудою відхилень від середнього

Набагато надійніше, тому, орієнтуватися при нормуванні не на екстремальні значення, а на типові, тобто статистичні характеристики даних, такі як середнє і дисперсія:

$$\tilde{x}_i = \frac{x_i - \bar{x}_i}{\sigma_i}, \qquad \bar{x}_i \equiv \frac{1}{P}\sum_{\alpha=1}^{P} x_i^{\alpha}, \quad \sigma_i^2 \equiv \frac{1}{P-1}\sum_{\alpha=1}^{P}\left(x_i^{\alpha} - \bar{x}_i\right)^2$$

У цьому випадку основна маса даних матиме одиничний масштаб, тобто типові значення всіх змінних будуть порівнюванні.

Однак, тепер нормовані величини не належать гарантовано одиничному інтервалу, більш того, максимальний розкид значень заздалегідь не відомий. Для вхідних даних це може бути і не важливо, але вихідні змінні будуть використовуватися в якості еталонів для вихідних нейронів. У разі, якщо вихідні нейрони – сігмоїдної функції, вони можуть набувати значень лише в одиничному діапазоні. Щоб встановити відповідність між навчальною вибіркою і нейромережею в цьому випадку необхідно обмежити діапазон зміни змінних.

Лінійне перетворення, як ми переконалися, не здатне віднормувати основну масу даних і одночасно обмежити діапазон можливих значень цих даних. Природний вихід із цієї ситуації - використовувати для попередньої обробки даних функцію активації тих же нейронів. Наприклад, нелінійне перетворення

$$\tilde{x}_i = f\left(\frac{x_i - \bar{x}_i}{\sigma_i}\right), \qquad f(a) = \frac{1}{1+e^{-a}}$$

нормує основну масу даних одночасно гарантуючи, що $\tilde{x}_i \in [0, 1]$ [31] (див. Рис. 1.3).

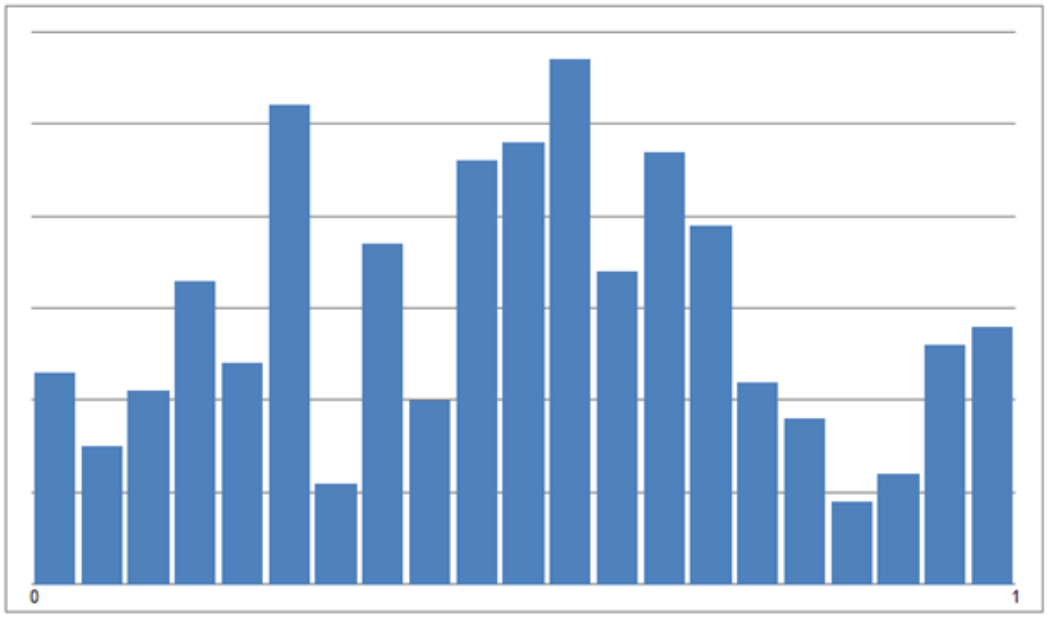


Рис. 1.3 Нелінійне нормування, що використовує логістичну функцію активації

Як видно з наведеного вище рисунка, розподіл значень після такого нелінійного перетворення набагато ближче до рівномірного.

До сих пір ми намагалися максимізувати ентропію кожного входу (виходу) окремо. Але, взагалі кажучи, можна домогтися набагато більшого максимізуючи їх спільну ентропію. Розглянемо цю техніку на прикладі спільного нормування входів, маючи на увазі, що з таким же успіхом її можна застосовувати і для виходів а також для всієї сукупності входів-виходів.

1.1.8 Спільне нормування: «вибілювання» входів

Якщо два входи статистично не незалежні, то їх спільна ентропія менше суми індивідуальних ентропій: $H(\tilde{x}_i \tilde{x}_j) \leq H(\tilde{x}_i) + H(\tilde{x}_i)$

Тому домігшись статистичної незалежності входів ми, тим самим, підвищимо інформаційну насиченість вхідної інформації. Це, однак, потребує більш складної процедури спільного нормування входів.

Замість того, щоб використовувати для нормування індивідуальні дисперсії, будемо розглядати вхідні дані в сукупності. Ми хочемо знайти таке лінійне перетворення, яке максимізувало б їх спільну ентропію. Для спрощення

завдання замість складнішої умови статистичної незалежності вимагатимемо, щоб нові входи після такого перетворення були декорельовані. Для цього розрахуємо середній вектор і коваріаційну матрицю даних за формулами:

$$\bar{\mathbf{x}} \equiv \frac{1}{P}\sum_{\alpha=1}^{P}\mathbf{x}^{\alpha}, \quad \Sigma_{ij}^{X} \equiv \frac{1}{P-1}\sum_{\alpha=1}^{P}\left(x_i^{\alpha} - \bar{x}_i\right)\left(x_j^{\alpha} - \bar{x}_j\right)$$

Потім знайдемо лінійне перетворення, що діагоналізує коваріаційну матрицю. Відповідна матриця складена з стовпців - власних векторів коваріаційної матриці:

$$\sum_j \Sigma_{ij}^{X} U_{jk} = \lambda_k U_{ik}$$

Легко переконатися, що лінійне перетворення, зване вибілюванням (whitening)

$$\tilde{x}_i = \left(x_k - \bar{x}_k\right) U_{ki} / \sqrt{\lambda_i}$$

перетворить всі входи в некорельовані величини з нульовим середнім і одиничною дисперсією [30].

Якщо вхідні дані представляють собою багатовимірний еліпсоїд, то графічно вибілювання виглядає як розтягнення цього еліпсоїда по його головним осях (див. Рис. 4).

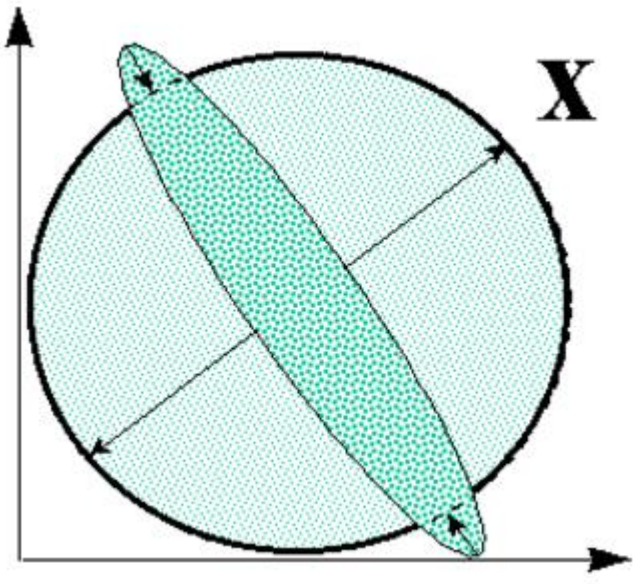


Рис. 1.4 Вибілювання вхідної інформації: підвищення інформативності входів за рахунок вирівнювання функції розподілу

1.2 Проблема незбалансованих датасетів

Нерідко виникають ситуації, коли в навчальному наборі даних частка прикладів деякого класу занадто мала (цей клас називають міноритарним, а інший, сильно представлений, - мажоритарним). Такі тенденції добре помітні в кредитному скорингу, в медицині, в маркетингу. Побудований на таких наборах даних класифікатор може виявитися абсолютно неефективним [6].

Слід зазначити те, що можуть відрізнятися і витрати помилкової класифікації. Причому невірна класифікація прикладів міноритарного класу, як правило, обходиться в рази дорожче, ніж помилкова класифікація прикладу мажоритарного класу.

Одним з підходів для вирішення зазначеної проблеми є застосування різних стратегій семплінгу, які можна розділити на дві групи: випадкові і спеціальні.

Відновлення балансу класів може проходити двома шляхами. У першому випадку видаляють деяку кількість прикладів мажоритарного класу (undersampling), у другому - збільшують кількість прикладів міноритарного (oversampling) [11].

Перейдемо до коротким теоретичним відомостями про найбільш поширених стратегіях семплінгу, а потім деякі з них можна порівняти, застосувавши на наборі даних з незбалансованими класами.

Набір даних незбалансований, якщо класи в ньому представлені не приблизно однаково. Дисбаланс на порядок 100 до 1 є звичайним явищем при отриманні початкового необробленного набору даних, в той час як в інших випадках може бути присутній дисбаланс 100000 до 1 [7].

Продуктивність алгоритмів машинного навчання, як правило, оцінюється за допомогою точності прогнозування (predictive accuracy). Тим не менш, це не підходить, коли дані незбалансованої і / або вартість різних помилок помітно різниться. Як приклад, розглянемо класифікацію пікселів в зображеннях мамографії, як тих що свідчать про ракову пухлину [15]. Типовий набір даних

мамографії може містити 98% нормальних пікселів і 2% аномальних пікселів. Проста стратегія вгадування класу більшості за замовчуванням буде давати прогностичну точність 98%. Проте, характер застосування вимагає досить високої точності правильного виявлення в міноритарному класі і дозволяє малу точність в мажоритарному класі. Звична прогностична точність явно не підходить в таких ситуаціях. Характеристика ROC є стандартним методом для оцінки продуктивності класифікатора в діапазоні між позитивними і помилковими позитивними частотами помилок [23]. Площа під кривою (AUC) є визнаною традиційною метрикою продуктивності для кривої ROC. Опукла ROC оболонка може бути також використана в якості надійного методу ідентифікації потенційно оптимальних класифікаторів. Якщо лінія проходить через точку на опуклій оболонці, то немає ніякої іншої лінії з тим же нахилом, що проходить через іншу точку з більшим істинним позитивним (TP). Таким чином, класифікатор в цій точці є оптимальним при будь-яких припущеннях розподілу з цим нахилом.

В питаннях машинного навчання проблема з класовим дисбалансом вирішується в двох напрямках. Одним з них є призначення різної ваги навчальним прикладам [17]. Інший метод полягає в зміні вибірки оригінального набору даних, або шляхом збільшення (over-sampling) міноритарного класу і / або зменшення (under-sampling) мажоритарного класу [17]. Найбільш передовим підходом є поєднання over-sampling міноритарного класу та under-sampling мажоритарного класу.

1.3 Збільшення навчальної множини

Додавання даних є одним з найпростіших і ефективних способів поліпшити якість навчальної множини та вирішення описаних проблем. При цьому просте додавання даних довільного виду не завжди ефективно, часто потрібно додати дані певного різновиду для підвищення якості розпізнавання. Поширеним підходом є програмна генерація. У разі використання синтетичних

навчальних даних зручніше всього згенерувати відсутні навчальні приклади. Однак не у всіх завданнях допустимо використання програмно згенерованих даних. У таких випадках доводиться застосовувати складніші методи додавання даних. Більш ефективним підходом є так зване “збільшення даних” (data augmentation). Згідно дослідженної літератури, аугментація (роздуття, збільшення) тренувальних даних можлива як в просторі даних, так і в просторі ознак [11, 24].

1.3.1 Аугментація в просторі даних

Модифікація наявних зображень з метою розширити навчальної множини активно застосовується при навчанні глибоких нейронних мереж, а також в умовах дефіциту розмічених даних. Застосовуються стиснення, розтягування, горизонтальне відображення, поворот, випадковий зсув в колірному просторі, випадкова або закономірна зміна деяких пікселів, обрізання частини зображення. Вважається, що додавання повністю випадкового шуму неефективно, слід додавати шум, обумовлений даними (який потенційно можливий в реальних даних). Цей метод дозволяє гнучко доповнювати простір навчальних даних саме тими значеннями, які є дефіцитними, тобто заповнювати “пробіли” (“sparse areas”) [9].

Оскільки глибинні мережі повинні бути навчені на величезній кількості тренувальних даних для досягнення задовільних результатів, якщо початковий набір зображень досить обмежений, то є доцільним застосувати методи збільшення даних для підвищення продуктивності. Збільшення даних стає звичним і навіть необхідним етапом роботи при підготовці сучасної глибокої мережі.

Є багато способів виконати збільшення даних, найпопулярніші з яких вже були згадані вище. Крім того, ефективним є поєднання декількох різних обробок, наприклад, роблячи обертання і випадкове масштабування одночасно. Крім того, існують більш складні техніки, наприклад можна підняти

насиченість і значення (компоненти S і V колірного простору HSV) всіх пікселів зображення в интервалі між 0,25 і 4, помножити ці значення на коефіцієнт від 0,7 і 1.4, і додати до них значення від -0,1 до 0,1. Крім того, можна додати значення між -0.1, 0.1 до компоненти відтінку (H компонента HSV) всіх пікселів в зображенні / контурі [2].

A. Krizhevsky і ін. запропонували техніку “fancy PCA” при підготовці знаменитого Alex-Net в 2012 році. Fancy PCA змінює інтенсивності RGB каналів в тренувальних зразках. На практиці, по-перше виконується PCA на множині піксельних значень RGB над множиною тренувальних образів. І потім, для кожного тренувального образу, додається деяка величина до кожного пікселя зображення RGB, яка є випадковою величиною взятою з гауссового розподілу з нульовим середнім і стандартним відхиленням 0,1. Важливо, що кожне відхилення використовується тільки один раз для всіх пікселів конкретного тренувального зображення та до тих пір, поки зображення не буде використовуватися для навчання знову. Тобто, коли модель отримуватиме те ж навчальне зображення знову, для нього буде випадковим чином згенероване інше відхилення для збільшення даних. При використанні цієї техніки на конкурсі ImageNet 2012 року, було отримано зниження помилки “топ-1” більш ніж на 1% [2].

Для нових графічних даних можна створювати правдоподібні перетворення існуючих зразків, які зберігають інформацію, необхідну для навчання, з перевіркою цілісності зображення, виконуваною людиною спостерігачем (чи може все ще людина розпізнати об'єкт). Одним із значних поліпшень в продуктивності класифікаторів на базі MNIST був шляхом введення пружних деформацій [24], в доповненні до існуючих афінних перетворень, для збільшення даних. Пружна деформація (elastic deformation) виконується шляхом визначення нормованого поля випадкових зсувів u(x, y), що для кожного місця розташування пікселя (x, y) в зображенні задає вектор зміщення, таким чином, що $Rw = Po + \alpha u$, де Rw і Po описують розташування пікселів в оригінальних і викривлених зображеннях відповідно. Сила зсуву в

пікселях задається альфа. Гладкість поля зсувів контролюються параметром u, який є стандартним відхиленням гаусового розподілу, який згорнутий з матрицями рівномірно розподілених випадкових величин, які утворюють виміри x і y поля зміщень u.

Для відомого датасету MNIST було виявлено, що великі деформації, зі зміщенням альфа ≥ 8 пікселів, можуть призвести до символів, які важко розпізнати за допомогою спостерігача-людини, що означає що інформація для навчання не була збережена. Ця втрата цілісності зображення викликається зсувом критичної частини символу за межі кордону зображення, або шляхом введення «зламу», що вносить суттєву нерозбірливість, як показано на рисунку 1.5. Були емпірично встановлені зналення $\alpha = 1{,}2$ пікселів з $\sigma = 20$ засновані на виконанні алгоритму CELM.

Рис. 1.5 Штучні приклади, створені за рахунок еластичних деформацій, в яких присутні викривлення що призводять до втрати цілісності. Оригінальний датасет MNIST в порівнянні з зразками з значеннями $\alpha = 1.2$ (зліва) та $\alpha = 8$ pixels (зправа).

1.3.2 Аугментація в просторі ознак

V. Chawla і W. Bowyer [6] запропонували підхід до збільшення кількісті прикладів міноритарного класу (oversampling), створюючи «синтетичні»

приклади, а не реплікуючи вже наявні, як у відомих до цього підходах. Він був названий SMOTE. Такий підхід навіяний методом, який виявився успішним в розпізнаванні рукописних символів [4]. Були створені додаткові дані для навчання шляхом виконання певних операцій на реальних даних. У їхньому випадку, такі операції, як обертання і перекіс були природним способом для покращення якості навчальних даних. За підходом SMOTE синтетичні приклади генеруються в менш специфічній для додатку манері, при роботі в «просторі ознак», а не «просторі даних».

Відбувається збільшення кількісті прикладів міноритарного класу (oversampling), ураховуючи кожен зразок міноритарного класу з введенням синтетичних прикладів уздовж відрізків ліній, що з'єднують деякі або всі найближчі сусідні міноритарні класи. Сусіди з найближчих класів обираються випадковим чином, залежно від кількості потрібних надлишкових прикладів. Класична реалізація в даний час використовує п'ять найближчих сусідів. Наприклад, якщо кількість необхідних надлишкових прикладів становить 200%, беруться тільки два з п'яти найближчих сусідів і один зразок генерується в напрямку кожного (для кожного оригінального зразка поточного класу).

Синтетичні зразки генеруються в такий спосіб: береться різницю між вектором ознак поточного класу і його найближчим сусідом. Множиться ця різницю на випадкове число між 0 і 1, і додається до вектору ознак на стадії розгляду. Це призводить до вибору випадкової точки вздовж відрізка між двома специфічними особливостями. Такий підхід досить ефективно згладжує регіон ознак міноритарного класу,робить його менш специфічним з точки зору усієї навчальної множини. Псевдокод алгоритму SMOTE [6] показаний нижче.

Algorithm SMOTE(T, N, k)

Input:

Number of minority class samples T;

Amount of SMOTE N%;

Number of nearest neighbors k

Output:

(N/100)* T synthetic minority class samples

1. (∗ If N is less than 100%, randomize the minority class samples as only a random
percent of them will be SMOTEd. ∗)
2. if N < 100
3. then Randomize the T minority class samples
4. T = (N/100) ∗ T
5. N = 100
6. endif
7. N = (int)(N/100)(∗ The amount of SMOTE is assumed to be in integral multiples of 100. ∗)
8. k = Number of nearest neighbors
9. numattrs = Number of attributes
10. Sample[ ][ ]: array for original minority class samples
11. newindex: keeps a count of number of synthetic samples generated, initialized to 0
12. Synthetic[ ][ ]: array for synthetic samples
(∗ Compute k nearest neighbors for each minority class sample only. ∗)
13. for i ← 1 to T
14. Compute k nearest neighbors for i, and save the indices in the nnarray
15. Populate(N, i, nnarray)
16. endfor
Populate(N, i, nnarray) (∗ Function to generate the synthetic samples. ∗)
17. while N != 0
18. Choose a random number between 1 and k, call it nn. This step chooses one of
the k nearest neighbors of i.
19. for attr ← 1 to numattrs
20. Compute: dif = Sample[nnarray[nn]][attr] − Sample[i][attr]
21. Compute: gap = random number between 0 and 1
22. Synthetic[newindex][attr] = Sample[i][attr] + gap ∗ dif
23. endfor
24. newindex++
25. N = N − 1
26. endwhile
27. return (∗ End of Populate. ∗)

Але в даному підході були недоліки, які стали поштовхом до створення нових алгоритмів. Множина навчальних даних (датасет) має проблему дисбалансу класу, коли цей клас має дуже невелику кількість прикладів по відношенню до інших класів. Простим класифікаторам зазвичай не вдається виявити міноритарний клас через надзвичайно низьку частоту зустрічання у виборці. Згідно The Synthetic Minority Over-Sampling Technique (SMOTE) (Техніка синтетичного збільшення прикладів міноритарного класу) новий синтетичний зразок створюється шляхом вибору випадкової точки в просторі ознак уздовж лінії, яка перетинає декілька випадково вибраних зразків одного і того ж класу. Пізніша з'явився похідний від SMOTE алгоритм Density Based SMOTE (DBSMOTE) [5], який генерує нові синтетичні зразки в межах відстані від центру кластера для класу. DBSMOTE генерує синтетичні екземпляри уздовж найкоротшого шляху від кожного позитивного приклада до псевдо-центра ваги кластера міноритарного класу. Отже, ці синтетичні екземпляри розташовуються щільно поблизу центроїда і рідко далеко від цього центроїда. Оскільки алгоритм DBSMOTE генерує свої синтетичні зразки навколо центру кожного класу, то можна було б очікувати, що ці синтетичні зразки не можуть сприяти скороченню ступеня перенавчання (overfit). Дійсно, часто виявиляють, що використання DBSMOTE фактично зменшує межу перенавчання.

Зазвичай в просунутих системах розпізнавання використовують як збільшення навчальної множини в просторі ознак, так і в просторі даних. На рисунку 6 показане поєднання обох підходів [28].

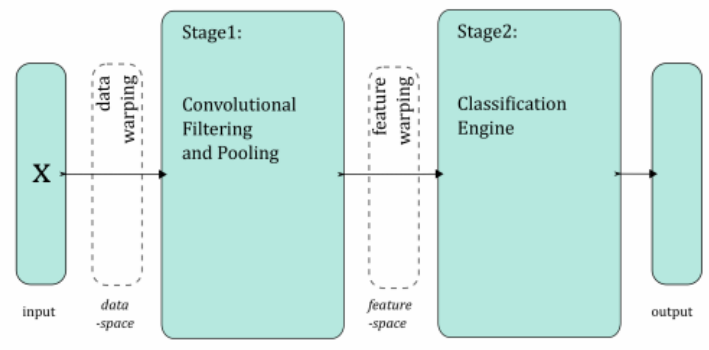


Рис.1.6 Послідовне поєднання підходів збільшення навчальної множинив просторі даних та ознак

1.4 Висновок

Нейронна мережа працює тільки з числовими вхідними даними, тому важливим етапом при підготовці даних є перетворення і кодування даних. При використанні на вхід нейронної мережі слід подавати значення з того діапазону, на якому вона навчалася. Існує поняття нормалізації даних. Метою нормалізації значень є перетворення даних до виду, який найбільш підходить для обробки, тобто дані, що надходять на вхід, повинні мати числовий тип, а їх значення повинні бути розподілені в певному діапазоні. На практиці ж саме предобробка даних може стати найбільш трудомістким елементом нейромережевого аналізу. Причому, знання основних принципів і прийомів передобробки даних не менше, а може бути навіть більш важливе, ніж знання власне нейромережевих алгоритмів. Весь технологічний ланцюжок, тобто необхідні етапи нейромережевого аналізу включають: кодування входів-виходів, нормування даних, передобробка даних, навчання та оцінка мережі. Успіх навчання нейромережі може вирішальним чином залежати від того, в якому вигляді представлена інформація для її навчання. Були розглянуті можливі проблеми при створенні навчального набору даних, а також додавання даних, як одним з найпростіших і ефективних способів поліпшити якість навчальної множини та вирішення описаних проблем. Поширеним підходом є програмна генерація. У разі використання синтетичних навчальних даних зручніше всього згенерувати відсутні навчальні приклади. Однак не у всіх завданнях допустимо використання програмно згенерованих даних. У таких випадках доводиться застосовувати складніші методи додавання даних. Більш ефективним підходом є так зване “збільшення даних”, тобто аугментація. Розглянута аугментація в просторі даних та в просторі ознак. Згідно з розглянутими джерелами, аугментація в просторі даних дає кращі результати, менш ресурсоємка, простіша в імплементації. Саме цей підхід є більш перспективним для використання в мережах IoT для вирішення проблем недостатнього розміру навчальних даних або класового розбалансування датасету.

2 РОЗМІР НАВЧАЛЬНОЇ МНОЖИНИ ТА КОНФІГУРАЦІЯ НЕЙРОННОЇ МЕРЕЖІ

2.1 Вступ

В цьому розділі буде описане дослідження впливу набору навчальних даних на результати класифікації нейронною мережею. Розглядатиметься кореляція між об’ємом навчальної вибірки та точністю класифікації нейронної мережі. Для цього будуть розроблені тестові моделі двох найпопулярніших архітектур нейронних мереж – згорткова нейронна мережа (CNN) та багатошаровий персептрон (MLP). Буде експериментально обрано оптимальну конфігурацію багатошарового персептрону. В якості тестового датасету обрано MNIST dataset, так як він містить графічні дані, що відповідають усім критеріям щодо збалансування та нормалізації, що були описані в попередньому розділі.

2.2 Перенавчання нейронної мережі

При навчанні нейронних мереж часто виникає проблема перенавчання (overfitting). Перенавчання, або надмірно близька підгонка - зайва точна відповідність нейронної мережі до конкретного набору навчальних прикладів, при якому мережу втрачає здатність до узагальнення.

Перенавчання виникає в разі занадто довгого навчання, недостатньої кількості навчальних прикладів або переускладненої структури нейронної мережі.

Перенавчання пов'язано з тим, що вибір навчальної (тренувальної) множини є випадковим. З перших кроків навчання відбувається зменшення похибки. На наступних кроках з метою зменшення похибки (цільової функції) параметри підлаштовуються під особливості навчальної множини. Однак при цьому відбувається "підлаштування" не під загальні закономірності ряду, а під особливості його частини - навчальної підмножини. При цьому точність прогнозу зменшується [25].

Один з варіантів боротьби з перенавчанням мережі - поділ навчальної вибірки на дві множини (навчальну і тестову).

На навчальній множині відбувається навчання нейронної мережі. На тестовій множині здійснюється перевірка побудованої моделі. Ці множини не повинні перетинатися.

З кожним кроком параметри моделі змінюються, однак постійне зменшення значення цільової функції відбувається саме на навчальній множині. При розбитті множини на дві можна спостерігати зміну похибки прогнозу на тестовій множині паралельно зі спостереженнями над навчальною множиною. Певну кількість кроків похибка прогнозу зменшується на обох множинах. Однак на певному етапі похибка на тестовій множині починає зростати, при цьому похибка на навчальній множині продовжує зменшуватися. Цей момент вважається кінцем реального або справжнього навчання, з нього і починається перенавчання.

У статистиці та машинному навчанні одним із найпоширеніших завдань є пристосовування «моделі» до набору тренувальних даних таким чином, щоби уможливити здійснення надійних передбачень на загальних даних, на яких не здійснювалося тренування. При перенавчанні статистична модель описує випадкову похибку або шум, замість взаємозв'язку, що лежить в основі даних [22]. Перенавчання виникає тоді, коли модель є занадто складною, такою, що має занадто багато параметрів відносно числа спостережень. Перенавчена модель має погану передбачувальну продуктивність, оскільки вона занадто сильно реагує на другорядні відхилення в тренувальних даних. Цю проблему добре ілюструє рисунок 2.1 та 2.2 [21].

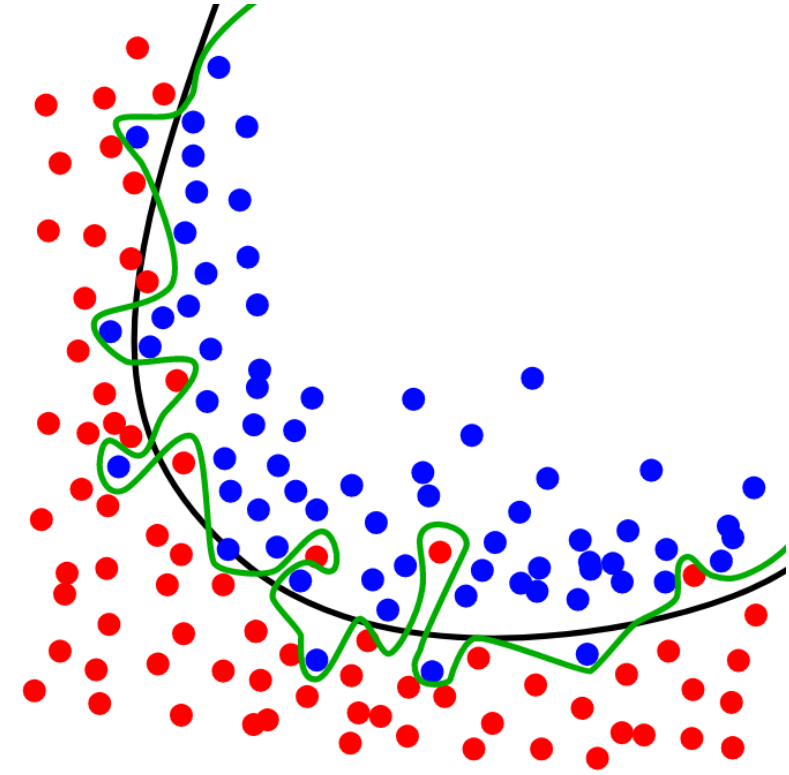


Рис. 2.1 Графічне зображення перенавчання, де синім та червоним показані об’єкти різних класів, а чорною та зеленою лініями - робота різних класифікаторів

Зелена лінія представляє перенавчену модель, а чорна — регуляризовану. В той час як зелена лінія найкраще слідує тренувальним даним, вона занадто залежить від них, і, ймовірно, матиме вищий рівень похибки на нових небачених даних у порівнянні з чорною лінією.

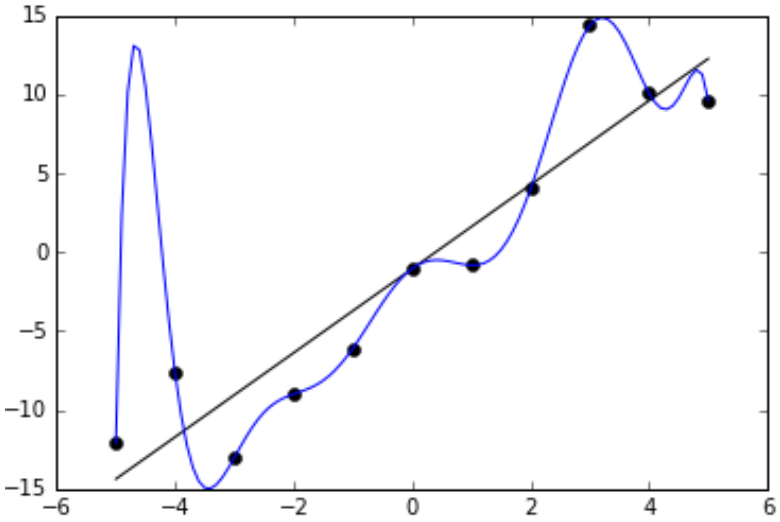


Рис. 2.2 Приблизно лінійні дані, до яких пристосовано лінійну та поліноміальну функції

На рисунку 2.2 зображені зашумлені (приблизно лінійні) дані, до яких пристосовано лінійну та поліноміальну функції. Хоч поліноміальна функція й пристосована ідеально, від лінійної можна очікувати кращого узагальнення. Іншими словами, якби ці дві функції застосовувалися для екстраполювання даних за межами тих, до яких здійснювалося пристосовування, то лінійна функція робила би кращі передбачення.

Можливість перенавчання існує тому, що критерій, який застосовується для тренування моделі, відрізняється від критерію, який застосовується для оцінки її ефективності. Зокрема, модель зазвичай тренують шляхом максимізації її продуктивності на якомусь наборі тренувальних даних. Проте її ефективність визначається не її продуктивністю на тренувальних даних, а її здатністю працювати добре на даних небачених. Перенавчання стається тоді, коли модель починає «запам'ятовувати» тренувальні дані, замість того, щоби «вчитися» узагальненню з тенденції [25]. Як крайній приклад, якщо число параметрів є таким же, або більшим, як число спостережень, то проста модель або процес навчання може відмінно передбачувати тренувальні дані, просто запам'ятовуючи їх повністю, але така модель зазвичай зазнаватиме рішучої невдачі при здійсненні передбачень про нові або небачені дані, оскільки ця проста модель взагалі не навчилася узагальнювати.

Потенціал перенавчання залежить не лише від кількостей параметрів та даних, але й від відповідності структури моделі формі даних, та величини похибки моделі в порівнянні з очікуваним рівнем шуму або похибки в даних. Навіть коли пристосована модель не має надмірного числа параметрів, слід очікувати, що пристосований взаємозв'язок працюватиме на новому наборі даних не так добре, як на наборі, використаному для пристосовування. Зокрема, значення коефіцієнту детермінації відносно первинних тренувальних даних скорочуватиметься.

Щоби уникати перенавчання, необхідно використовувати додаткові методики (наприклад, перехресну перевірку, регуляризацію, ранню зупинку, обрізку, баєсові апріорні параметрів або порівняння моделей), які можуть вказувати, коли подальше тренування не даватиме кращого узагальнення. Основою деяких методик є або явно штрафувати занадто складні моделі, або перевіряти здатність моделі до узагальнення шляхом оцінки її продуктивності на наборі даних, не використаному для тренування, який вважається наближенням типових небачених даних, з якими стикатиметься модель.

Гарною аналогією перенавчання задачі є уявити дитину, яка намагається вивчити, що є вікном, а що не є вікном, ми починаємо показувати їй вікна, і вона виявляє на початковому етапі, що всі вікна мають скло та раму, і через них можна дивитися назовні, деякі з них може бути відчинено. Якщо ми продовжимо показувати ті самі вікна, то дитина може також зробити помилковий висновок, що всі вікна є зеленими, і що всі зелені рами є вікнами. Перенавчаючись таким чином цієї задачі.

Зазвичай алгоритм навчання тренується з використанням деякого набору «тренувальних даних»: зразкових ситуацій, для яких бажаний вихід є відомим. Метою є, щоби алгоритм також добре працював над передбаченням виходу при подаванні «перевірних даних», які не траплялися під час його тренування.

Перенавчання є застосуванням моделей або процедур, які порушують лезо Оккама, наприклад, включаючи більше регульованих параметрів, ніж є зрештою оптимально, або використовуючи складніший підхід, ніж є зрештою

оптимально. Для прикладу завеликого числа регульованих параметрів розгляньмо такий набір даних, де тренувальні дані для у може бути адекватно передбачено лінійною функцією двох залежних змінних. Така функція вимагає лише трьох параметрів (відсікання та двох нахилів). Заміна цієї простої функції новою, складнішою квадратичною функцією, або новою, складнішою лінійною функцією від понад двох залежних змінних, несе ризик: лезо Оккама значить, що будь-яка задана складна функція є апріорі менш імовірною за будь-яку задану просту функцію. Якщо цю нову, складнішу функцію обрано замість простої функції, і якщо не було достатньо великої користі для пристосовування до тренувальних даних, щоби протиставити її підвищенню складності, то нова складніша функція «перенавчається» даних, і складна перенавчена функція, швидше за все, працюватиме гірше на перевірних даних за межами тренувального набору, ніж простіша функція, навіть якщо складніша функція працювала добре, або навіть краще, на наборі тренувальному.

При порівнянні різних типів моделей складність не можна вимірювати виключно підрахунком того, скільки параметрів існує в кожній з моделей; мусить також розглядатися й виразність кожного з параметрів. Наприклад, є нетривіальним порівнювати безпосередньо складності нейронної мережі (яка може відстежувати криволінійні взаємозв'язки) з m параметрами, та регресійної моделі з n параметрами.

Перенавчання є особливо ймовірним в тих випадках, коли навчання виконувалося занадто довго, або коли тренувальні зразки є рідкісними, що спричиняє пристосовування до дуже особливих випадкових ознак тренувальних даних, які не мають причинного взаємозв'язку з цільовою функцією [25]. В процесі цього перенавчання продуктивність на тренувальних зразках продовжує підвищуватися, тоді як продуктивність на небачених даних стає гіршою.

Як простий приклад розгляньмо базу даних роздрібних купівель, яка включає придбану позицію, покупця, та дату й час купівлі. Нескладно побудувати модель, яка ідеально пристосовується до тренувального набору із

застосуванням дати й часу купівлі, щоби передбачувати інші ознаки; але ця модель взагалі не узагальнюватиметься на нові дані, оскільки ті минулі часи вже ніколи не настануть.

Як правило, кажуть, що алгоритм навчання перенавчається відносно простішого, якщо він є точнішим у пристосовуванні до відомих даних (розумність заднім числом), але менш точним у передбачуванні нових даних (далекоглядність). Перенавчання можна інтуїтивно розуміти з точки зору тієї обставини, що інформацію з усього минулого досвіду може бути поділено на дві групи: інформацію, яка стосується майбутнього, і недоречну інформацію («шум»). За всіх інших рівних умов, що складнішим для передбачування є критерій (тобто, що вищою є невизначеність), то більше шуму, який треба ігнорувати, міститься в минулій інформації. Задача полягає у визначенні того, яку частину ігнорувати. Алгоритм навчання, який знижує шанси пристосовування до шуму, називається надійним.

Найочевиднішим наслідком перенавчання є погана продуктивність на перевірному наборі даних. До інших негативних наслідків належать:

- Перенавчена функція схильна вимагати більше інформації про кожен елемент перевірного набору даних, ніж функція оптимальна; збирання цих додаткових непотрібних даних може бути витратним або схильним до помилок, особливо якщо кожну окрему частину інформації потрібно збирати за допомогою людського спостереження та введення даних вручну.
- Складніша, перенавчена функція схильна бути менш переносною, ніж проста. Як одна з крайностей, лінійна регресія з однією змінною є настільки переносною, що, за потреби, може навіть здійснюватися вручну. На протилежній крайності знаходяться моделі, які може бути відтворено лише точним дублюванням цілісної постановки первинного розробника, що ускладнює повторне використання або наукове відтворення.

2.3 Оцінка продуктивності нейронної мережі

Продуктивність алгоритмів машинного навчання, як правило, оцінюють за допомогою матриці неточностей (confusion matrix), як показано на рисунку 2.3. Стовпці відповідають за передбачений клас, а рядки за фактичний клас. У матриці неточностей, TN це число правильно класифікованих негативних прикладів (True Negative), FP (False Positive) є число негативних прикладів неправильно класифікованих як позитивні (помилкових спрацьовувань), FN є число позитивних прикладів неправильно класифікованих як негативні (False Negative) і TP число позитивних прикладів, які правильно класифіковані (True Positive).

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Рис. 2.3 Матриця неточностей (Confusion Matrix)

Прогностична точність є мірою продуктивності якою, як правило, оцінюють алгоритми машинного навчання і визначається як точність:

$$\text{Accuracy} = (TP + TN) / (TP + FP + TN + FN)$$

В контексті збалансованих наборів даних і рівних витрат помилок, розумно використовувати частоту помилок в якості показника продуктивності.

2.4 Згорткові нейроннні мережі (CNN)

При застосуванні для розпізнавання зображень, згорткові нейронні мережі складаються з декількох шарів невеликих збірок нейронів, які

обробляють частини вхідного зображення, що називаються рецептивними полями. Виходи цих збірок потім укладаються таким чином, щоби їхні вхідні області перекривалися, для отримання кращого представлення первинного зображення; це повторюється для кожного такого шару. Укладання з перекриттям дозволяє ЗНМ бути терпимими до паралельних перенесень вхідного зображення [1].

Вони також складаються з різних комбінацій згорткових та повноз'єднаних шарів, із застосуванням шару поточкової нелінійності в кінці кожного шару [8]. Для зниження числа вільних параметрів та покращення узагальнення вводиться операція згортки на малих областях входу. Однією з головних переваг згорткових мереж є використання спільної ваги у згорткових шарах, що означає, що для кожного пікселя шару використовується один і той же фільтр (банк ваги); це як зменшує обсяг необхідної пам'яті, так і поліпшує продуктивність.

Архітектура CNN формується стосом різних шарів, що перетворюють вхідний об'єм на вихідний об'єм (що, наприклад, зберігає рівні відношення до класів) за допомогою диференційовної функції. Зазвичай застосовується декілька різних типів шарів.

2.4.1 Вибір фреймворка

Розробка програмного рішення для класифікації об'єктів за допомогою штучних нейронних мереж типу CNN потребує здійснити вибір фреймворку. У міру того як популярність CNN збільшилася, виникли різні реалізації. Найбільш важливими факторами фреймворків для глибокого навчання є їх продуктивність, зручність і простота використання, яка включає наявність інтерфейсів API для різних мов програмування, якість документації та діяльність спільноти користувачів та розробників фреймворку. Продуктивність є критично важливою вимогою, оскільки CNN є складними моделями з

багатьма значущими параметрами. Популярними фреймворками для глибокого навчання є Torch, Theano, Caffe і TensorFlow.

Torch є фреймворком для наукових обчислень, який реалізує CNN, а також інші алгоритми машинного навчання з використанням мов програмування C і Lua, де Lua використовується в якості мови сценаріїв. Він розроблений і використовується для Facebook AI, Google Search і Twitter DeepMind.

Caffe є фреймворком спеціалізованим для глибокого навчання з використанням C ++ для реалізації і протоколу Google Protobuf для визначення мережевих архітектур і контролю навчання. Крім того, Caffe надає інтерфейси API для Python і Matlab. Обидва фреймворка забезпечують реалізацію для навчання CNN на CPU і GPU.

Theano значно відрізняється від Torch і Caffe. Це в основному бібліотека Python і таким чином, також реалізована на Python. Theano не є безпосередньо бібліотекою для реалізацією тільки CNN, а більш загальної бібліотекою, яка може бути легко використана для реалізації моделі CNN.

TensorFlow це відкрита програмна бібліотека для машинного навчання цілій низці задач, розроблена компанією Google для задоволення її потреб у системах, здатних будувати та тренувати нейронні мережі для виявлення та розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди. Її наразі застосовують як для досліджень, так і для розробки продуктів Google. TensorFlow було початково розроблено командою Google Brain для внутрішнього використання в Google, поки її не було випущено під відкритою ліцензією Apache 2.0, 9 листопада 2015 року.

У даній роботі Caffe був обраний в якості фреймворка для глибокого навчання, через його високу продуктивність, хорошу документованість, простоту використання і розгортання. Саме за допомогою цього фреймворку проведемо тестування описаних вище підходів до збільшення об’єму навчальних даних для мережі типу CNN (Рис. 2.4).

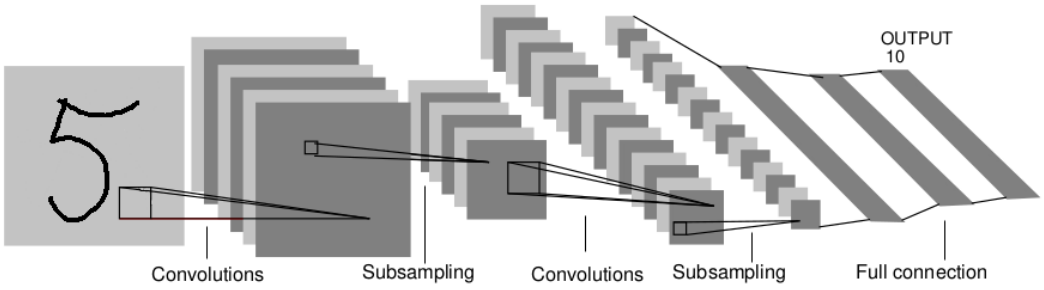


Рис. 2.4 Графічне зображення класичної архітектури згорткової нейронної мережі

2.4.2 Побудова архітектури згорткової нейронної мережі

Встановлення та компілювання фреймворку Caffe не є складним завданням, також весь функціонал доступний вже “з коробки”. В наведених далі прикладах вважатимемо, що встановлений Caffe знаходиться в CAFFE_ROOT.

Спочатку потрібно завантажити дані з сайту MNIST і перетворити їх в потрібний формат. Щоб зробити це, виконуємо наступні команди:

```
cd $CAFFE_ROOT
./data/mnist/get_mnist.sh
./mnist/create_mnist.sh
```

Для виконання скриптів повинні бути встановленні додаткові залежності, такі як wget та gunzip. Після успішного виконання скриптів отримаємо `mnist_train_lmdb`, та `mnist_test_lmdb`.

Перед тим як приступити до наступних етапів підготовки до навчання нейронної мережі, вияснимо деякі моменти. Ми будемо використовувати мережу LeNet, яка, як відомо, добре працює в завданнях класифікації, зокрема в класифікації цифр. Версія, яку ми використовуватимемо, відрізнятиметься від оригінальної реалізації LeNet, заміною сигмовидної активації на Rectified Linear Unit (ReLU) активацію нейронів. Цей крок вже став звичним для покращення властивостей цієї архітектури нейронної мережі.

Конструкція LeNet по суті своїй є класичною CNN (convolutional neural network), що активно використовується в великих моделях, таких як ImageNet. Загалом, вона складається з згорткового шару (convolutional layer - звідки і отримала свою назву) з подальшим шаром пулінгу (pooling layer), ще одного шару згортки з подальшим шаром пулінгу, а потім двома повністю з'єднаними шарами (fully connected layers), подібними до традиційних багатошарових персептронів. Порядок та конфігурація шарів визначається у файлі $CAFFE_ROOT/mnist/lenet_train_test.prototxt.

Опис архітектури здійснюється в файлі lenet_train_test.prototxt, який визначає модель LeNet для класифікації рукописних цифр MNIST. Для створення такого опису мережі в фреймворці Caffe використовується формат Google Protocol Buffers, або просто Google Protobuf. Це формат серіалізації даних, запропонований корпорацією Google, як альтернатива XML. Оригінальна реалізація Google для C++, Java та Python доступна під вільною ліцензією. Google стверджує, що protocol buffers в декілька раз збільшує швидкість обробки даних та суттєво зменшує обсяги передаваної інформації. Усі дефініції формату, що є специфічними для Caffe можуть бути знайдені у $CAFFE_ROOT/src/caffe/proto/caffe.proto.

Опис починається, даючи ім'я мережі:

```
name: "LeNet"
```

2.4.3 Рівень даних

В даному шарі відбувається читання даних MNIST з lmdb файлу, що ми створили раніше. Цей процес визначається за допомогою шару даних:

```
layer {
  name: "mnist"
  type: "Data"
  data_param {
    source: "mnist_train_lmdb"
    backend: LMDB
    batch_size: 64
    scale: 0.00390625
```

```
  }
  top: "data"
  top: "label"
}
```

Зокрема, цей шар має ім'я mnist, тип - дані, і зчитує дані з вказаного джерела lmdb. Ми будемо використовувати розмір пакету 64 (`batch_size`), що говорить про те, що використовуватиметься не класичний стохастичний градієнтний спуск, а пакетний. Для пакетного градієнтного спуску функція втрат обчислюється для всіх зразків разом узятих після закінчення епохи, і потім вводяться поправки вагових коефіцієнтів нейрона відповідно до методу зворотного поширення помилки. Стохастичний метод відразу після обчислення виходу мережі на одному зразку вводить поправки в вагові коефіцієнти. Пакетний метод показує себе як більш швидкий і стабільний. Також вказується параметр масштабування вхідних пікселів (`scale`), так що вони знаходяться в діапазоні [0,1). Число 0.00390625 є 1 поділеним на 256. І, нарешті, цей шар дає дві мітки, одна мітка даних, і одна мітка ярликів даних.

2.4.4 Рівень згортки

Визначимо перший рівень згортки наступним чином:

```
layer {
  name: "conv1"
  type: "Convolution"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
```

```
  bottom: "data"
  top: "conv1"
}
```

Цей шар приймає двійковий об'єкт даних, що надходить з рівня даних, описаний відповідною міткою. Створюється шар Conv1. Він містить вихід на 20 каналів, з розміром згорткового ядра 5 і здійснюється з кроком 1.

Заповнювачі (`weight_filler`) дозволяють випадковим чином форматувати значення ваг і зміщення. Для вагового заповнювача, ми будемо використовувати алгоритм Xavier, який автоматично визначає масштаб ініціалізації на основі кількості вхідних і вихідних нейронів. Для заповнювача зміщень, ми будемо формувати його як константу, з значення заповнення за замовчуванням 0.

lr_mults визначають параметр learning rate для поточного шару. У цьому випадку, ми встановимо learning rate ваги таким, яким він і буде під час виконання, і learning rate зміщення буде вдвічі більшим - це, як правило, призводить до збільшення швидкості збіжності.

2.4.5 Рівень субдискретизації (Pooling Layer)

Визначення шару Pooling виглядає наступним чином:

```
layer {
  name: "pool1"
  type: "Pooling"
  pooling_param {
    kernel_size: 2
    stride: 2
    pool: MAX
  }
  bottom: "conv1"
  top: "pool1"
}
```

Як видно, виконуються максимальні пули з розміром ядра 2 і кроком 2 (таким чином немає перекриття між сусідніми регіонами). Параметри:

```
bottom: "conv1"
  top: "pool1"
```

визначають послідовність з’єднання шарів. По аналогії пишуться другий шар згортки та субдискретизації.

Conv2:

```
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

Pool2:

```
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

2.4.6 Рівень повного з’єднання (Fully Connected Layer)

Вихід з шарів згортки з логічної сторони являє собою деякі закономірності даних досить високого рівня. Незважаючи на те, що вихід може бути з'єднаний з вихідним шаром, додавання шару повного з’єднання є (зазвичай) ефективним способом вивчення нелінійних комбінацій цих закономірностей [8].

По суті шари згортки виявляють значущі, високорівневі дані простору ознак, а шар повного з’єднання відповідає за вивчення функцій (можливо, нелінійних) в цьому просторі.

Налаштування Fully Connected Layer виглядає наступним чином:

```
layer {
  name: "ip1"
  type: "InnerProduct"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "pool2"
  top: "ip1"
}
```

Шар InnerProduct (у фреймворку Caffe відповідає шару повного з’єднання) обробляє вхідний сигнал як простий вектор, і видає вихідний сигнал у вигляді одного вектора (з висотою і шириною встановленими в значення 1). Крім вже відомих параметрів, в даному описі зазначається кількість вихідних каналів.

2.4.7 Рівень ReLU

```
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
```

ReLU є поелементною операцією. Дане вхідне значення x, шар ReLU обчислює вихід, як x, якщо x> 0 і negative_slope * x, якщо x <= 0. Коли параметр negative_slope не встановлено, це еквівалентно стандартній функції ReLU взяття max(x , 0). Також підтримується “in-place”обчислення, а це означає, що нижня і верхня мітка може бути одна й та ж, що зменшує споживання пам'яті.

Далі пишемо другий шар `InnerProduct`

```
layer {
  name: "ip2"
  type: "InnerProduct"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "ip1"
  top: "ip2"
}
```

2.4.8 Рівень Loss Layer

Шар втрат визначає, як тренування мережі штрафує відхилення між передбаченими та справжніми мітками, і є, як правило, останнім шаром у мережі. В ньому можуть використовуватися різні функції втрат для різних завдань. Loss Layer відповідає за співставлення вихідних значень мережі та цільових (target values) та вирахування значень для мінімізації.

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
}
```

Шар Softmax_loss реалізує як SoftMax (normalized exponential) так і поліноміальну логістичну функцію втрат (loss function)(що економить час і покращує чисельну стійкість). Він приймає дві мітки, перша з яких є передбаченням мережі, а друга `"label"` отримується з рівня даних. Цей шар не дає на вихід мітки - він виконує обчислення значення функції втрат, та повідомляє про це під час запуску зворотного поширення (backpropagation), і ініціює градієнт по відношенню до ip2. Саме в цьому процесі закладена основна частина алгоритму навчання нейронної мережі.

2.4.9 Розділення архітектури мережі на навчальну та тестову

Визначення шару може містити особливі правила, що говорять, що даний шар включається у архітектуру мережі лише на певному етапі - навчальному або тестовому.

```
layer {
  ......
  include: { phase: TRAIN }
}
```

Якщо таке правило не зазначене, шар по замовчуванню включається в архітектуру мережі на обох етапах. Якщо зазначити в правилі `phase: TEST`, то шар буде включено лише на фазі тестування.

В обраній архітектурі згорткової нейронної мережі присутні два визначення шару даних. Визначення для тренувальної фази було розглянуто вище. Визначення шару даних для фази тестування виглядає наступним чином:

```
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "mnist/mnist_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
```

Також присутній шар `Accuracy` , що зазначений лише для фази тестування. Його задача полягає в повідомленні точності класифікації мережею кожні 100 ітерацій.

```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
```

2.4.10 Конфігурація навчання мережі

Крім файлу архітектури мережі `lenet_train_test.prototxt`, обов'язковим у фреймворці Caffe є створення файлу конфігурацій `lenet_solver.prototxt`.

Файл з визначенням архітектури мережі:

```
net: "mnist/lenet_train_test.prototxt"
```

Параметр test_iter зазначає скільки прямих проходів буде здійснено на етапі тестування.

У випадку датасету MNIST, є 10,000 тестових зображень. Було обрано значення параметру batch size 100 в тестовому визначенні шару даних. Отож для покриття усіх тестових зображень, варто обрати значення параметру test_iter 100.

```
test_iter: 100
```

Зазначаємо, що тестування буде здійснюватися кожні 500 ітерацій тренування.

```
test_interval: 500
```

Зазначення швидкості навчання (learning rate), імпульсу (momentum) та спаду ваги (weight decay) мережі.

```
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
```

Імпульс використовується, щоб зменшити коливання в зміні ваги між послідовними ітераціями. Спад ваги нормалізує зміни ваги.

По суті параметр weight decay змінює функцію, яка оптимізується, в той час як momentum змінює шлях пошуку оптимуму.

Weight decay, приближує коефіцієнти до нуля, гарантуючи, що знайдено локальний оптимум з параметрами малої величини. Це, як правило, має вирішальне значення для запобігання перенавчання (хоча інші види обмежень ваг можуть працювати теж). Як побічний ефект, він може також зробити модель легшою для оптимізації, зробивши цільову функцію більш опуклою.

Так як є цільова функція, необхідно вирішити, як пересуватися по ній. Найшвидший спуск по градієнту є найпростішим підходом, але коливання

(fluctuations) можуть бути великою проблемою. Додавання імпульсу (momentum) допомагає вирішити цю проблему.

```
lr_policy: "inv"
```

Значення параметру `lr_policy` говорить про те, як змінюватиметься параметр Learning rate (на графіку по осі ординат) з ходом навчання (на графіку по осі абсцис від 0% до 99%). Значення `"inv"` говорить, що зміна йтиме з залежністю 1/T. Цю залежність можна візуалізувати за допомогою інтерфейсу NVIDIA/DIGITS для Caffe (Рис. 5).

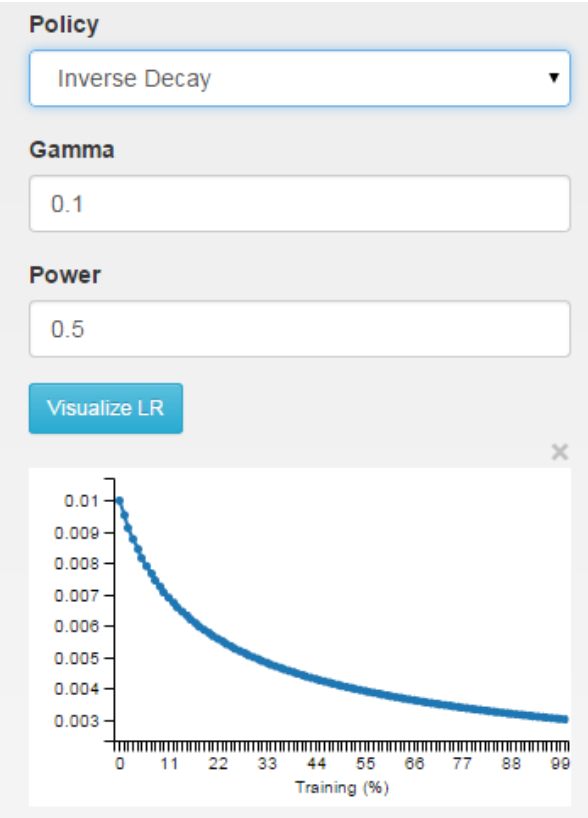


Рис.5 Візуалізація зміни парметру learning rate з ходом навчання нейронної мережі за допомогою інтерфейсу NVIDIA/DIGITS для Caffe

Можливий вибір значення `"fixed"` , при якому значення Learning rate залишалося б сталим протягом усього процесу навчання, це дало б більшу швидкість збіжності, але був би ризик не знайти глобального оптимуму.

```
gamma: 0.0001
power: 0.75
```

Зазначаємо параметр для показу статистики кожні 100 ітерацій

```
display: 100
```

Зазначаємо параметр максимальної кількості ітерацій

```
max_iter: 10000
```

Зберігаємо проміжкові результати кожні 5000 ітерацій, з можливістю відкотитися до кожного збереження а продовжити навчання з іншими параметрами

```
snapshot: 5000
snapshot_prefix: "mnist/lenet"
```

Вибираємо платформу для здійснення обрахунків (CPU або GPU)

```
solver_mode: GPU
```

Файл конфігурації навчання для мережі LeNet у фреймворку Caffe виглядає наступним чином:

```
net: "mnist/lenet_train_test.prototxt"
test_iter: 100
test_interval: 500
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
lr_policy: "inv"
gamma: 0.0001
power: 0.75
display: 100
max_iter: 10000
snapshot: 5000
snapshot_prefix: "mnist/lenet"
solver_mode: GPU
```

2.4.11 Тренування та тестування моделі

Після створення файлів архітектури моделі та конфігурації навчання, необхідно запустити команду `caffe` з параметром `train` та вказаним шляхом до файлу конфігурацій `lenet_solver.prototxt`.

```
caffe train --solver=mnist/lenet_solver.prototxt
```

Після чого фреймворк Caffe розпочне процес навчання та тестування мережі. Весь процес якісно та детально логується в консоль як показано на рисунку 2.5.

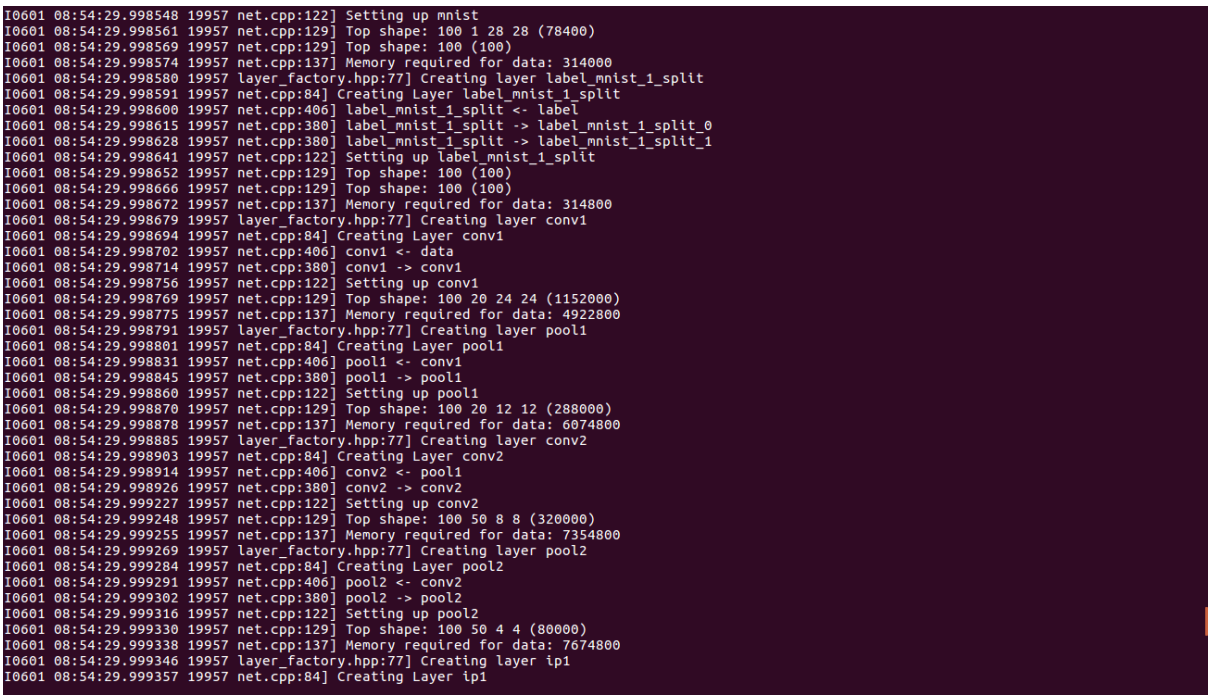


Рис. 2.5 Процес навчання та тестування згорткової нейронної мережі у фреймворці Caffe

Інформація, що виводиться на екран, корисна для розуміння процесів, що відбуваються в момент навчання, а також для подальшої відладки архітектури або конфігурації.

Прогрес навчання логується відповідними записами:

I0601 08:56:30.692229 20017 net.cpp:255] Network initialization done.

I0601 08:56:30.692283 20017 solver.cpp:56] Solver scaffolding done.

I0601 08:56:30.692322 20017 caffe.cpp:248] Starting Optimization

I0601 08:56:30.692330 20017 solver.cpp:273] Solving LeNet

I0601 08:56:30.692337 20017 solver.cpp:274] Learning Rate Policy: inv

I0601 08:56:30.693008 20017 solver.cpp:331] Iteration 0, Testing net (#0)

I0601 08:56:35.482034 20020 data_layer.cpp:73] Restarting data prefetching from start.

I0601 08:56:35.682839 20017 solver.cpp:398] Test net output #0: accuracy = 0.1177

...

I0601 08:56:37.256662 20017 solver.cpp:331] Iteration 20, Testing net (#0)

I0601 08:56:41.912153 20020 data_layer.cpp:73] Restarting data prefetching from start.

I0601 08:56:42.108898 20017 solver.cpp:398] Test net output #0: accuracy = 0.7179

...

I0601 08:56:43.675174 20017 solver.cpp:331] Iteration 40, Testing net (#0)

I0601 08:56:48.337430 20020 data_layer.cpp:73] Restarting data prefetching from start.

I0601 08:56:48.530983 20017 solver.cpp:398] Test net output #0: accuracy = 0.8435

...

```
I0601 08:56:50.098110 20017 solver.cpp:331] Iteration 60, Testing net (#0)
I0601 08:56:54.750447 20020 data_layer.cpp:73] Restarting data prefetching from start.
I0601 08:56:54.947866 20017 solver.cpp:398]     Test net output #0: accuracy = 0.872
...
I0601 08:56:56.523301 20017 solver.cpp:331] Iteration 80, Testing net (#0)
I0601 08:57:01.167789 20020 data_layer.cpp:73] Restarting data prefetching from start.
I0601 08:57:01.363181 20017 solver.cpp:398]     Test net output #0: accuracy = 0.9119
...
I0601 08:57:02.929152 20017 solver.cpp:331] Iteration 100, Testing net (#0)
I0601 08:57:07.571240 20020 data_layer.cpp:73] Restarting data prefetching from start.
I0601 08:57:07.765524 20017 solver.cpp:398]     Test net output #0: accuracy = 0.9146
...
I0601 08:58:07.144680 20017 solver.cpp:331] Iteration 300, Testing net (#0)
I0601 08:58:11.803262 20020 data_layer.cpp:73] Restarting data prefetching from start.
I0601 08:58:11.999020 20017 solver.cpp:398]     Test net output #0: accuracy = 0.9672
...
I0601 08:59:11.848783 20017 solver.cpp:331] Iteration 500, Testing net (#0)
I0601 08:59:16.492518 20020 data_layer.cpp:73] Restarting data prefetching from start.
I0601 08:59:16.684715 20017 solver.cpp:398]     Test net output #0: accuracy = 0.9747
…
I0601 09:14:54.054831 20017 solver.cpp:331] Iteration 10000, Testing net (#0)
I0601 09:15:01.054885 20020 data_layer.cpp:73] Restarting data prefetching from start.
I0601 09:15:01.054889 20017 solver.cpp:398]     Test net output #0: accuracy = 0.991
```

2.4.12 Зміна набору навчальних даних

В даному випадку було досягнуто рівня помилки 0.9%. Набір даних містить 60,000 навчальних прикладів та 10,000 тестових. Кожен клас містить приблизно 6000 навчальних та 1000 тестових прикладів (класів 10, по одному на кожну цифру від 0 до 9).

Наступним етапом є дослідження впливу кількості навчальних прикладів на результуючу помилку.

Весь датасет представлений у форматі LMDB (Lightning Memory-Mapped Database), що забезпечує швидкий доступ до даних при навчанні у фреймворці Caffe. LMDB це програмна бібліотека, яка забезпечує високу швидкодію для вбудованих баз даних які призначені для підтримання багатопоточності та великої частоти транзакцій. База зберігається у вигляді пар ключ-значення в

бінарному вігляді. LMDB написаний на мові прграмування C з прив’язкою API для декількох мов програмування. LMDB зберігає довільні пари ключ-дані як масив байт, має можливість пошуку в діапазоні, підтримує декілька елементів даних для одного ключа і має спеціальний режим для додавання записів в кінці бази даних (MDB_APPEND), що дає відчутне збільшення продуктивності запису в порівнянні з іншими аналогічними базами [https://lmdb.readthedocs.io/en/release/]. LMDB не є реляційної базою даних, а строго послідовністю ключ-значення.

LMDB також може одночасно використовуватися як в багатопотоковому так і в багатопроцесному середовищі, в яких при масштабуванні продуктивність читання зростає лінійно відповідно архітектури середовища. В базах даних LMDB лише один процес може здійснювати запис в одну одиницю часу, однак, на відміну від багатьох подібних баз даних типу ключ-значення, процес що здійснює транзакцію запису, не блокує процеси-читачі, як і процеси, що здійснює транзакцію на читання не блокують процеси, що здійснюють запис. LMDB також незвичайна в тому, що кілька додатків на одній і тій же системі можуть одночасно відкрити і використовувати ту ж саму базу LMDB, що приводить до масштабування продуктивності. Крім того, LMDB не вимагає ведення журналу транзакцій (тим самим збільшуючи продуктивність запису), оскільки підтримує цілісність даних по суті своєї конструкції.

По замовчуванню оригінальний датасет MNIST представлений у форматі LMDB.

Для того щоб внести зміни у навчальну множину, що знаходиться в бінарному вигляді, необхідно провести конвертацію. Для цих цілей був написаний скрипт на мові Python (див. Додаток А).

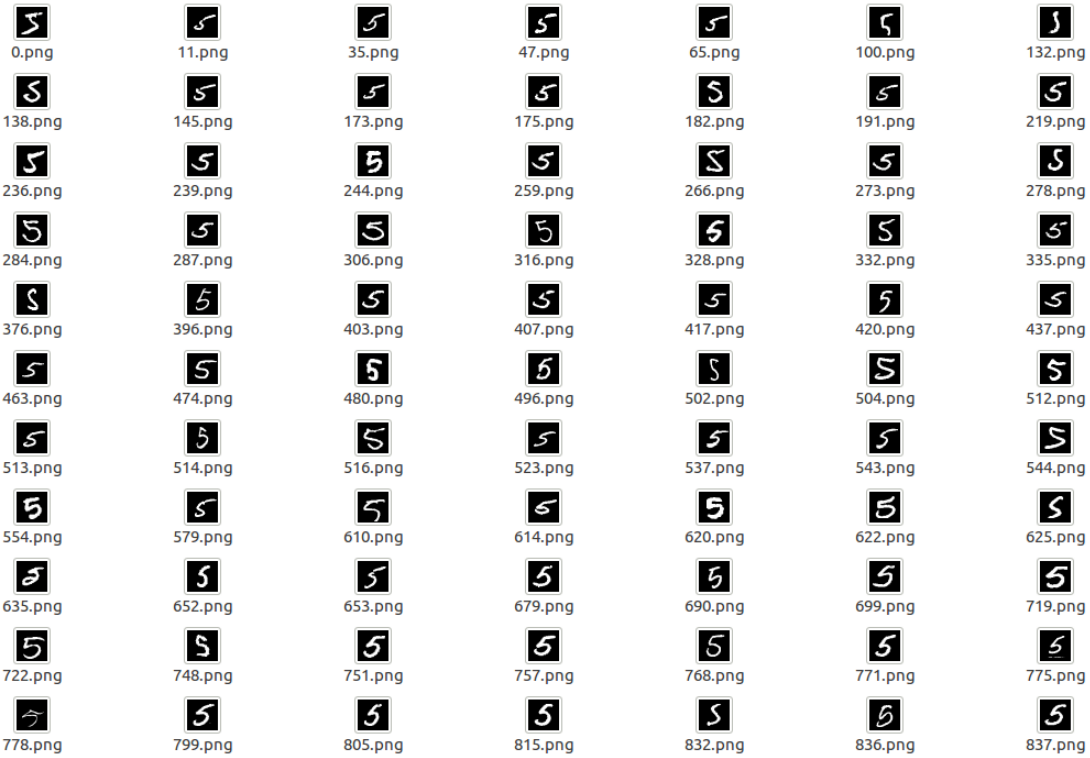


Рис. 2.6 Приклад вихідних зображень навчальної множини після конвертації з формату LMDB

Отримуємо каталоги з зображеннями в форматі PNG, як показано на рисунку 2.6. Наступним етапом є дослідження залежності помилки класифікації від розміру датасету.Для цього змінимо кількісно наповнення кожного класу.

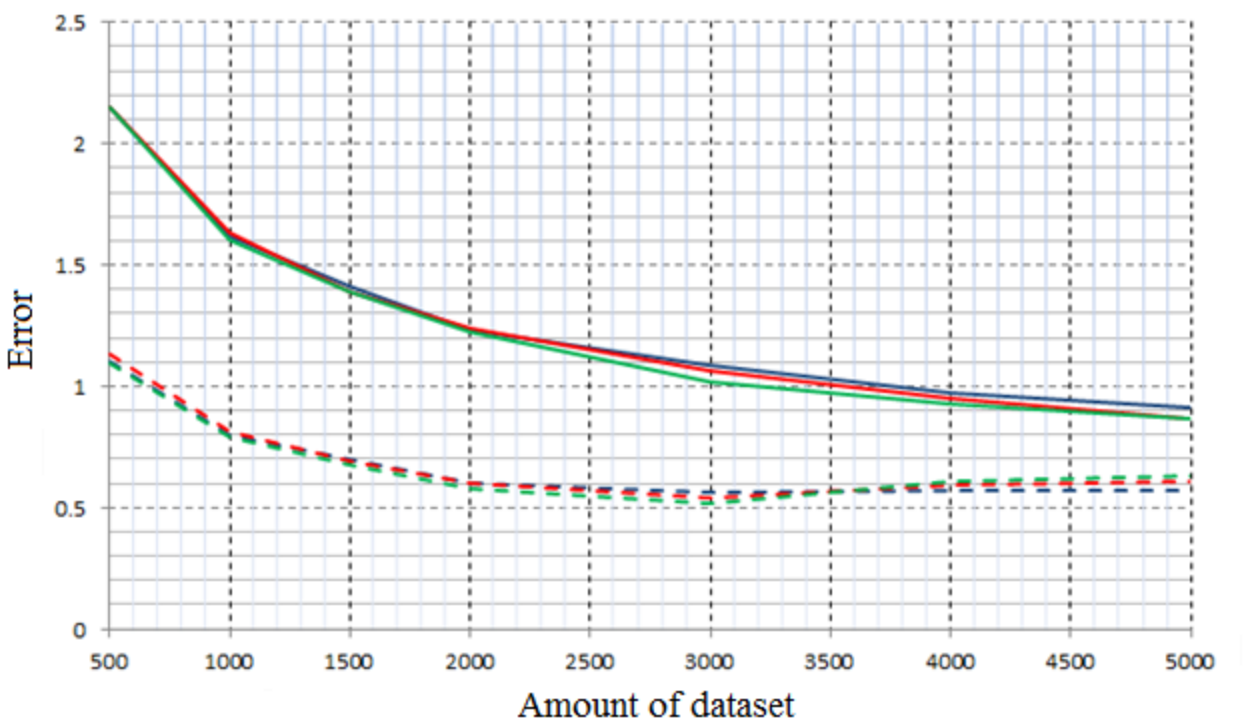
Для конвертації графічних зображень в формат бази даних LMDB, який необхідний для роботи фреймворку Caffe, використаємо скрипт, написаний на мові Python (див. Додаток Б).

2.4.13 Залежність тестової похибки нейронної мережі типу CNN від об’єму навчальної множини

Проведемо навчання мережі на поступово збільшуваному об’ємі навчальних даних, починаючи з 500 прикладів для кожного з 10 класів. Важливим є те, щоб у кожному класі кількість зображень була однаковою, для уникнення проблем з незбалансованим набором даним, що було детально описано в попередніх розділах.

Експеримент проводився три рази при однакових умовах (синя, червона та зелена лінії відповідно кожному експерименту). Суцільна лінія показує похибку на тестовій вибірці для кожного з експериментів, пунктирна - на навчальній.

Збільшення кількості навчальних даних в усіх випадках веде до збільшення кількості ітерацій алгоритму зворотнього розповсюдження для незмінної кількості епох.



Мал. 2.7 Залежність тестової (суцільна лінія) та тренувальної (пунктирна лінія) похибки нейронної мережі типу CNN від кількості елементів в тренувальній множині

Як бачимо на рисунку 2.7, зі збільшенням кількості навчальних прикладів, різниця між тренувальною та тестовою похибкою зменшується, що показує зменшення рівня перенавчання. Це показує найголовнішу перевагу збільшення навчальної вибірки - зменшення рівня перенавчання при сталих параметрах мережі.

Спостереження, яке характерне саме для CNN - збільшення розмірів датасету призводить до зменшення як тренувальної, так і тестової похибок. Для більшості інших архітектур нейронних мереж характерна особливість, при якій збільшення датасету пизводить до покращення тестової похибки, але погіршення навчальної, так як мережа краще пристосовується для генералізації на нові приклади, але гірше працює для запам’ятовування збільшеної кількості навчальних прикладів.

2.5 Багатошаровий персептрон (MLP)

Проведемо навчання також й багатошарового персептрона для виявлення закономірностей, пов’язаних з розміром навчального датасету. Для роботи з MLP був обраний фреймворк Chainer що працює на основі Python.

Chainer є досить зручним фреймворком для нейронних мереж. Основною метою його створення була зручність, тобто можливість написання комплексної архітектури мережі просто та інтуїтивно. Більшість сучасних фреймворків побудовані на схемі “Define-and-Run”, за якої мережа є визначеною та зафіксованою. Натомість, фреймворк Chainer побудований на схемі “Define-by-Run”, що дозволяє мережі адаптивно підлаштовуватись під поточні розрахунки. В фреймворці зберігається історяія обчислень, а не програмна логіка. Для прикладу, в Chainer є досить простим написання умовних циклів в дифініціях мережі, що дає можливість адаптувати мережу в ході навчання. Таким чином виглядає визначення класу багатошарового персептрона з двома прихованими шарами:

```
class MLP(chainer.Chain):
    def __init__(self, n_units, n_out):
        super(MLP, self).__init__(
            # the size of the inputs to each layer will be inferred
            l1=L.Linear(None, n_units),  # n_in -> n_units
            l2=L.Linear(None, n_units),  # n_units -> n_units
            l3=L.Linear(None, n_out),  # n_units -> n_out
        )
    def __call__(self, x):
        h1 = F.relu(self.l1(x))
        h2 = F.relu(self.l2(h1))
        y = self.l3(h2)
        return y
```

Всі ноди є повнозв’язними, перша з яких - вектор вхідних параметрів (input layer), останній - вихідні результати класифікації (output layer). Проміжны шари називаються “прихованими” (hidden layer). Отриману архітектуру можна зобразити графічно (рис. 2.8)

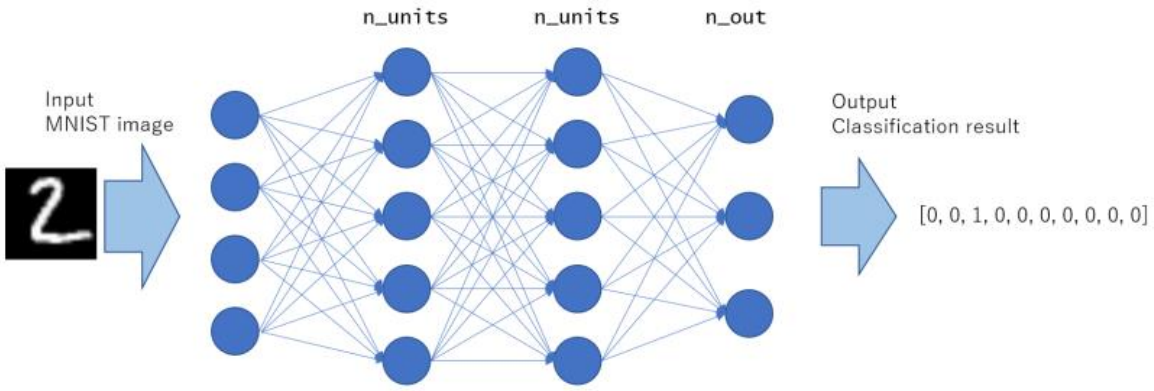


Рис. 2.8 Багатошаровий персептрон з двома прихованими шарами однакової розмірності.

Як бачимо з декларування функції __call__ , на вхід приймається параметр X, що є масивом який предствляє вхідні зображення. Функція повертає У, що показує передбачену ймовірність для кожного класу. Але цього недостатньо для навчання моделі. Необхідним є розрахунок функції втрат (loss

function), що стане об’єктом оптимізації під час навчання. В задачах класифікаціїї зазвичай використовують функцію softmax cross entropy loss. Вихід функції Linear, що можелює шари в фреймворці Chainer може давати довільне число, а задача softmax функції є конвертація її в проміжок [0,1], таким чином показуючи передбачену ймовірність для кожного класу.

Задача функції cross entropy є розрахунок функції втрат між двома розподілами ймовірностей. Фреймворк Chainer має робочу функцію F.softmax_cross_entropy(y, t) для розрахунку softmax від **y** та cross entropy з **t.** Функція втрат буде меншою, якщо передбачений розподіл ймовірностей **y** буде близьким до дійсного розподілу ймовірностей **t.** Інтуїтивно кажучи, функція втрат має низьке значення, якщо модель змогла передбачити правильний клас зображення.

Для вирахування softmax cross entropy loss, ми об’являємо наступний клас в фреймворці Chainer:

```
class SoftmaxClassifier(chainer.Chain):
    def __init__(self, predictor):
        super(SoftmaxClassifier, self).__init__(
            predictor=predictor
        )
     def __call__(self, x, t):
        y = self.predictor(x)
        self.loss = F.softmax_cross_entropy(y, t)
        self.accuracy = F.accuracy(y, t)
        return self.loss
```

Наступним кроком є ініціація моделі з використанням написаних класів:

```
unit = 20
model = MLP(unit, 10)
classifier_model = SoftmaxClassifier(model)
```

Значення unit говорить про кількість елементів (нейронів) у кожному прихованому шарі мережі. При ініціації MLP, значення n_out було виставлене в 10, по кількості вихідних класів (цифри від 0 до 9 відповідно). Далі створюється classifier_model для емелемнта MLP model в якості предиктора. Як можна побачити, елементи були з’єднанні між собою в модель classifier_model.

Елементи і далі можуть поєднуватися, створюючи нові моделі. Саме тому фремворк і був названий Chainer від англійського “chain” - поєднуавати.

Так як функція втрат описана в блоці __call__, можна передати створену модель на вхід оптимізатору для проведення тренування.

```
optimizer = chainer.optimizers.Adam()
optimizer.setup(classifier_model)
```

Тренування проводиться через оновлення параметрів через функцію update:

```
optimizer.update(classifier_model, x, t)
```

Таким чином буде проводитися обрахунок функції втрат у моделі classifier_model та оптимізовуватися її внутрішні параметри за допомогою обраного алгоритма оптимізації, в нашому випадку - `Adam()`.

Алгоритм зворотнього розповсюдження виконується всередині функції update, отож немає необхідності реалізовувати його в явному вигляді.

Також для значного пришвидшення навчання в фреймворці Chainer присутня підтримка обрахунків на відеокарті. Для цього повинна бути присутня NVIDIA GPU та встановлена CUDA. Після цього може бути дописаний відповідний код:

```
if gpu >= 0:
        chainer.cuda.get_device(gpu).use()  # Make a specified GPU current
        classifier_model.to_gpu()             # Copy the model to the GPU
    xp = np if gpu < 0 else cuda.cupy
```

Змінна gpu повинна містити ідентифікатор вашого фізичного пристрою відеокарти. Для виконання обрахунків на центральному процесорі, слід виставити цей параметр в -1. Для використання відеокарти по замовчуванню, параметр gpu виставляється в 0. Виклик chainer.cuda.get_device(gpu).use() виконує визначення використовуваного пристрою, а classifier_model.to_gpu() передає внутрішні параметри моделі до відеокарти. Після чого cuda.cupy починає розрахунки.

2.5.1 Фази тренування та тестування

Написаний скрипт складається з двох фаз (див. додаток Б) - тренування та тестування. В задачах класифікації в машинному навчанні є досить важливою перевірка моделі на можливість генералізації на нові приклади. Навіть якщо фукція втрат продовжує зменшуватися для тренувальної вибірки, то це не завжди означає що покращується показник на тестовій вибірці. Це виливається в проблему перенавчання, яка була описана в одному з попередніх розділів. Для правильного навчання мережі, необхідно перевіряти функцію втрат як на тренувальній, так і на тестовій вибірках.

Тренування

optimizer.update() оновлює внутрішні параметри моделі під час навчання з метою мінімізації функції втрат.

```
perm = np.random.permutation(N)
        sum_accuracy = 0
        sum_loss = 0
        start = time.time()
        for i in six.moves.range(0, N, batchsize):
            x = chainer.Variable(xp.asarray(train[perm[i:i + batchsize]][0]))
            t = chainer.Variable(xp.asarray(train[perm[i:i + batchsize]][1]))
             # Pass the loss function (Classifier defines it) and its arguments
            optimizer.update(classifier_model, x, t)
             sum_loss += float(classifier_model.loss.data) * len(t.data)
            sum_accuracy += float(classifier_model.accuracy.data) * len(t.data)
        end = time.time()
        elapsed_time = end - start
        throughput = N / elapsed_time
        print('train mean loss={}, accuracy={}, throughput={} images/sec'.format(
            sum_loss / N, sum_accuracy / N, throughput))
```

Функція np.random.permutation(N) здійснює рандомізацію навчальних прикладів для створення навчальних підвибірок (mini-batches). Якщо функція втрат навчальної множини не починається зменшуватися з початку навчання, це може означати, про невірно зазначені параметри моделі. Якщо функція втрат

навчальної множини перестає зменшуватися, то це зазвичай означає що модель знаходиться в стані насичення і нормальним є припинити процес навчання.

Тестування

Важливою відмінністю від попередньої фази (навчання) є те, що модель є сформованою та не підлягає зміні (тобто не відбувається виклик optimizer.update()). Тестова підвибірка розцінюється як дані, що ні разу не проганялися через модель (та формується виходячи з цього).

```
sum_accuracy = 0
        sum_loss = 0
        for i in six.moves.range(0, N_test, batchsize):
            index = np.asarray(list(range(i, i + batchsize)))
            x = chainer.Variable(xp.asarray(test[index][0]))
            t = chainer.Variable(xp.asarray(test[index][1]))

            loss = classifier_model(x, t)
            sum_loss += float(loss.data) * len(t.data)
            sum_accuracy += float(classifier_model.accuracy.data) * len(t.data)

        print('test  mean loss={}, accuracy={}'.format(
            sum_loss / N_test, sum_accuracy / N_test))
```

Дана фаза не зачіпає внутрішні параметри моделі, а лише вираховує функцію втрат та точність. В ході навчання функція втрат тестової множини повинна знижуватися, а точність на тестовій множині рости, але це не завжди так.

Якщо тестова функція втрат не зменшується, в той час як навчальна функція втрат продовжує зменшуватися, це може свідчити про такі проблеми як:

- недостатня кількість навчальних даних (рекомендованим є використання штучного збільшення навчальних даних (data augmentation))
- мережа надто складана для поточної задачі (потрібно зменшити кількість внутрішніх параметрів мережі, наприклад нейронів в прихованих шарах)

2.5.2 Виконання навчання та тестування в середовищі Pycharm

Так як фреймворк Chainer працює на Python, було обрано середовище розробки для цієї мови програмування, а саме PyCharm від Get Brains (рис. 2.9). Воно відрізняється від інших просунутою системою мета-контролю коду та допомоги корисувачу під час роботи.

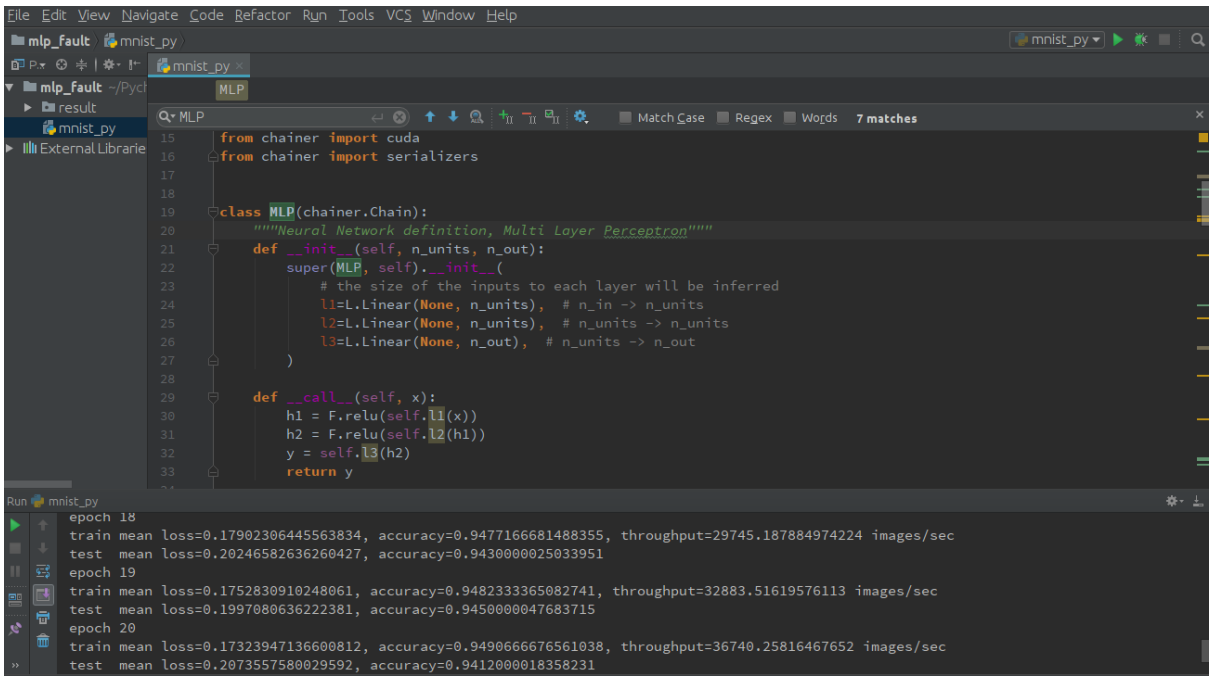


Рис. 2.9 Інтерфейс середовища PyCharm з виконуваним скриптом по навчанню багатошарового персептрону

2.5.3 Вибір оптимальної складності моделі MLP

Базова конфігурація MLP з двома прихованими шарами по 20 нейронів дала точність 0.9601 на тестовій вибірці. Це досить непогана точність для такої відносно примітивної моделі. Значення яке рахується state of the art на даний момент дорівнює 0.997.

Аби обрати оптимальну складність моделі, проведемо навчання на поступово збільшуваній кількості нейронів в прихованих шарах, починаючи з 10. Отриманий графік можна побачити на рисунку 2.10. Результати для тестової вибірки показані в таблиці 1.

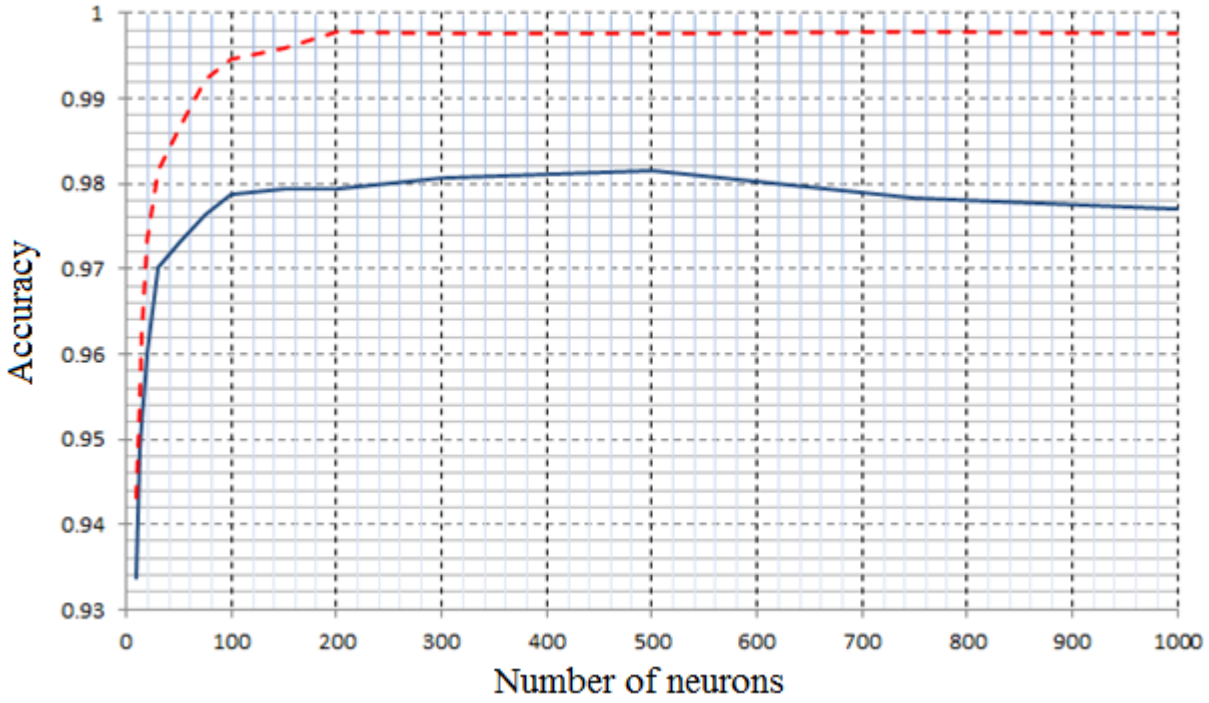


Рис. 2.10 Графік залежності точності прогнозування нейронної мережі типу MLP в залежності від кількості нейронів в прихованому шарі. Пунктирною лінією показана точність на тренувальній вибірці, а суцільною - точність на тестовій вибірці

Таблиця 2.1 Точність на тестовій вибірці нейронної мережі типу CNN для різної кількості нейронів в прихованому шарі

Кількість елементів в кожному прихованому шарі	Точність на тестовій вибірці
10	0.9337
12	0.9488
15	0.9519
20	0.9601
30	0.9702
50	0.9731
75	0.9763
100	0.9787

150	0.9794
200	0.9794
300	0.9806
500	0.9816
750	0.9783
1000	0.9771

Як бачимо, точність класифікації тренувальноі вибірки майже одразу досягла асимптотичного рівня та не змінювалася з подальшим ростом кількості елементів в прихованому шарі. Натомість, точність класифікації тестової вибірки почала падати з набором кількості елементів в прихованому шарі, тобто для більш складних моделей.

В ході навчання мережі було відмічено що для випадку з 750 та 1000 елементами в прихованих шарах, найкращий результат класифікації на тренувальній вибірці був присутній не на останній епосі навчання, а раніше, що говорить, що мало місце явище перенавчання.

Мережа з кількість елементів 750 показала найкращий результат на епосі 12, який дорівнював 0.9845, в порівнянні з 0.9783 на останній епосі.

В свою чергу, мережа з кількість елементів 1000 показала найкращий результат на епосі 14, який дорівнював 0.9834, в порівнянні з 0.9771 на останній епосі. Що говорить про те, що простіша модель показує кращу точність на поточній задачі.

Також здійснювався замір часу навчання для кожної конфігурації моделі. Графік залежності часу навчання від кількості елементів в прихованому шарі зображено на рис. 2.11.

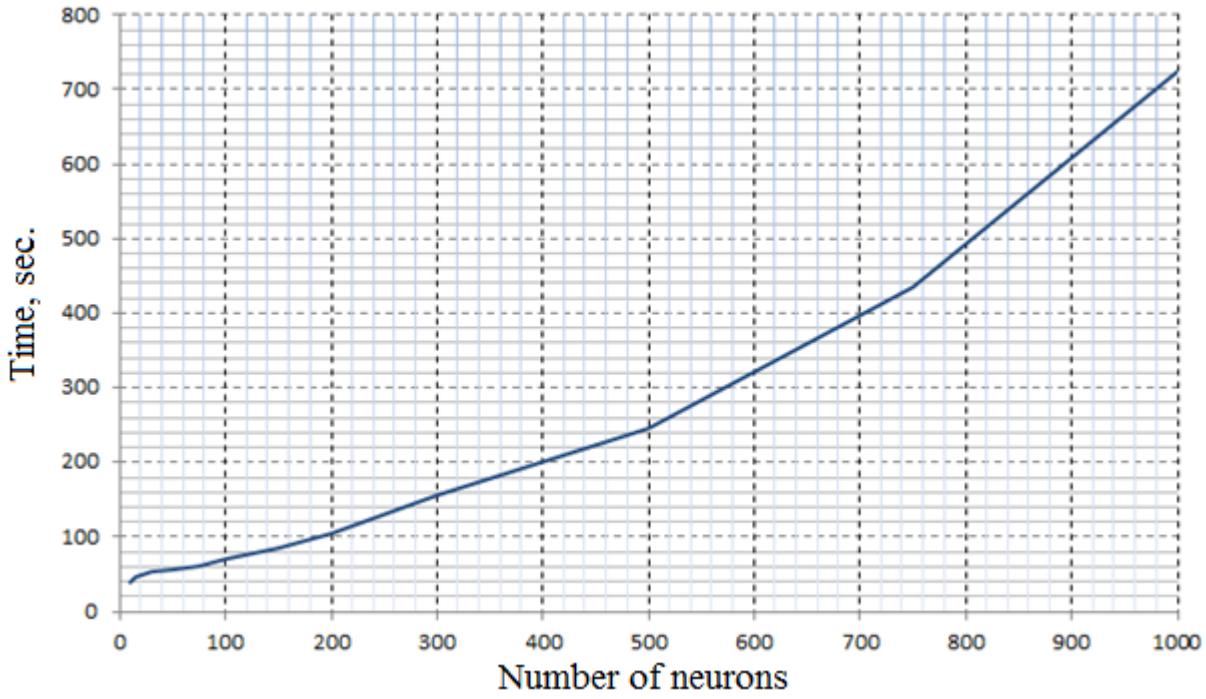
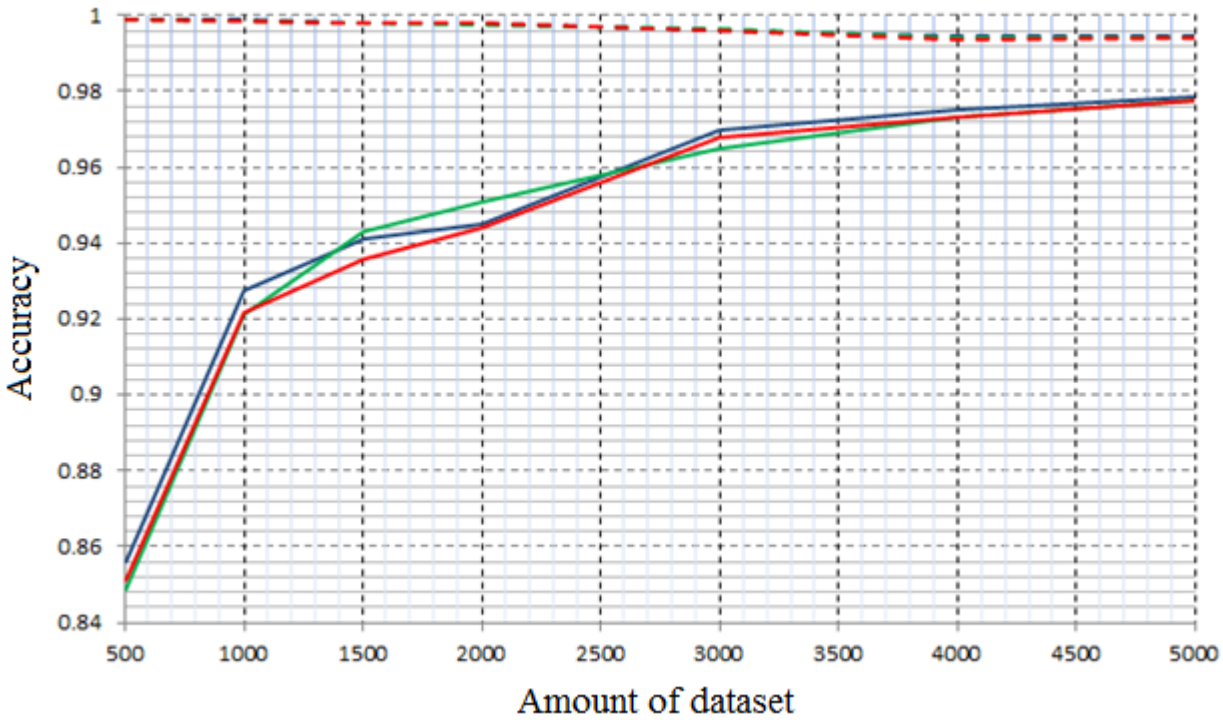


Рис. 2.11 Залежність часу навчання (сек.) від кількості елементів в прихованому шарі для нейронної мережі типу MLP

Виходячи з отриманих результатів, є оптимальним обрати кількість елементів прихованому шарі, що дорівнюватиме 100, так як точність прогнозування з подальшим збільшенням елементів зростає досить незначно, потребуючи в свою чергу значного збільшення використовуваних ресурсів.

2.5.4 Залежність тестової похибки нейронної мережі типу MLP від об’єму навчальної множини

Проведемо навчання мережі на поступово збільшуваному об’ємі навчальних даних, починаючи з 500 прикладів для кожного з 10 класів. Важливим є те, щоб у кожному класі кількість зображень була однаковою, для уникнення проблем з незбалансованим набором даним, що було детально описано в попередніх розділах.



Мал. 2.12 Залежність точності класифікації на тестовій (суцільна лінія) та тренувальній (пунктирна лінія) вибірках для нейронної мережі типу MLP від кількості елементів в тренувальній множині

Експеримент проводився три рази при однакових умовах (синя, червона та зелена лінії відповідно кожному експерименту). Суцільна лінія показує похибку на тестовій вибірці для кожного з експериментів, пунктирна - на навчальній.

Як бачимо на рисунку 2.12 зі збільшенням кількості навчальних прикладів, різниця між тренувальною та тестовою точністю зменшується, що показує зменшення рівня перенавчання. Це показує найголовнішу перевагу збільшення навчальної вибірки - зменшення рівня перенавчання при сталих параметрах мережі.

Спостереження, яке характерне саме для MLP - збільшення розмірів датасету призводить до збільшення точності на тестовій вибірці та до незначного зменшення точності на тренувальній. Це явище в спеціалізованій літературі пояснюється тим, що мережа краще пристосовується для

генералізації на нові приклади, але гірше працює для запам’ятовування збільшеної кількості навчальних прикладів.

2.6 Висновки

Було описане дослідження впливу набору навчальних даних на результати класифікації нейронною мережею. Розглянулася кореляція між об’ємом навчальної вибірки та точністю класифікації на двох рохроблених тестових моделях – згортковій нейронній мережі (CNN) та багатошаровому персептроні (MLP). Для багатошарового персептрона обрано оптимальну конфігурацію, а саме кількість нейронів в кожному прихованому шарі. Були проведені експерименти по навчанню нейронних мереж на постійно збільшуваній кількості навчальних даних, за результатами яких побудовані графіки. Як видно з побудованих графіків, точність класифікації мережі досить значно залежить від об’єму навчальних даних, особливо на початкових етапах. Як бачимо, зі збільшенням кількості навчальних прикладів, різниця між тренувальною та тестовою точністю зменшується, що показує зменшення рівня перенавчання. Це показує найголовнішу перевагу збільшення навчальної вибірки - зменшення рівня перенавчання при сталих параметрах мережі, що характерне для обох тестових моделей.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ШТУЧНОГО ЗБІЛЬШЕННЯ НАВЧАЛЬНОЇ МНОЖИНИ

3.1 Вступ

Проаналізований раніше процес створення датасету та навчання нейронної мережі, опис особливостей роботи контроллерів Internet of things, а також експериментально підтверджена висока залежність між якістю навчання нейронної мережі та об’ємом датасету говорять про перспективність рішення про штучне збільшення об’ємів навчальних даних за допомогою спеціалізованих програмних засобів. Метою цієї роботи було обрано оптимізацію процесу навчання штучної нейронної мережі для задоволення особливостей роботи в технології інтернету речей через зменшення необхідної кількості навчальних даних, тому буде реалізовано та досліджено ефективність підходу збільшення навчальної вибірки в просторі даних (augmentation in data-space). Роздуття даних в просторі ознак (augmentation in feature-space) розглядатися на практиці не буде, так як виходячи з існуючих сучасних досліджень [9, 23], цей підхід не дає значних результатів в порівнянні з збільшення навчальної вибірки в просторі даних, а також потребує значно більших ресурсів, що критично для використання в IoT. Тестування буде проводитися на двох найбільш поширених архітектурах нейронних мереж - CNN (convolutional neural network - згорткова нейронна мережа) та MLP (multi-layer perceptron - багатошаровий персептрон), моделі яких вже були імплементовані та протестовані в попередньому розділі. Основний виграш, який планується отримати в результаті використання підходу augmentation - це зменшення перенавчання мережі в умовах, коли подальше збільшення реальних даних неможливе.

3.2 Розробка програмного додатку на мові Python

Був розроблений додаток на мові Python, задачею якого є створення штучних зображень на основі вже існуючих. Для цього були використані потужності бібліотек skimage та matplotlib , які призначені для просунутої роботи з зображеннями, а також середовище PyCharm від JetBrains, яке вже описувалося раніше.

```
from skimage import transform
from skimage.transform import rotate
from matplotlib import pyplot as plt
from matplotlib import image as img
import glob
import random
import os
for nums in range(0, 10): # ітерування по кількості класів в датасеті
    augm_from_dir = "/home/den/PycharmProjects/mlp_fault/mnist_png/training/"
        + str(nums) + "/*.png"  # вибір директорії з датасетом
    filelist = glob.glob(augm_from_dir) # створення масиву доступних навчальних зображень
    augm_from = 500 # вибір кількості оригінальних зображень кожного класу
    make_num = 4500 # вибір необхідної кількості штучних зображень, яка буде створена
    if make_num < augm_from:
        make_num = augm_from
    multiplyer = make_num / augm_from
    for fname in filelist: # ітерування по масиву доступних навчальних зображень
        if augm_from != 0:
            augm_from -= 1
            print(fname)
            image = img.imread(fname) # зчитування зображення
            mul = multiplyer # показник розмноження
            while (mul > 0):
                mul -= 1
                rand_angle = random.randint(0, 360) # поворот на випадковий кут
                print("Angle = " + str(rand_angle))
                image_rotated = rotate(image, rand_angle)
                # plt.imshow(image_rotated, cmap=plt.cm.gray)
                # plt.show()
```

```
rand_scale = random.randint(80, 150) / 100
print("Scale = " + str(rand_scale))
tf_sc = transform.SimilarityTransform(scale=rand_scale) # масштабування на
                                                        # коефіцієнт від 0.8 до 1.5
image_scaled = transform.warp(image_rotated, tf_sc, output_shape=(28, 28))
rand_translX = random.randint(0, 10) - 5
rand_translY = random.randint(0, 10) - 5 # зрушення по осі абсцис та ординат на
                                         # випадкову кількість пікселів в межах -5;5
print("Shift = [" + str(rand_translX) + ", " + str(rand_translY) + "]")
tf_shift = transform.SimilarityTransform(translation=[rand_translX, rand_translY])
image_shifted = transform.warp(image_scaled, tf_shift, output_shape=(28, 28))
# plt.imshow(image_shifted, cmap=plt.cm.gray)
# plt.show()
aug_dir = "/home/den/PycharmProjects/mlp_fault/augmented/" + str(nums) + "/"
if not os.path.exists(aug_dir):
    os.makedirs(aug_dir) # створення директорії для збереження штучно
                         # створених зображень
img.imsave(str(aug_dir) + "aug" + str(nums) + "_"
        + str(augm_from) + "_" + str(int(mul)) + ".png",
          image_shifted, cmap=plt.cm.gray)  # збереження штучно створених
                                            # зображень
```

Як бачимо з коду, проводиться ітерування по директоріям оригінальної навчальної вибірки та береться augm_from зображень кожного класу. Далі створюється make_num штучних зображень для кожного класу. Для прикладу ми вказали, що для кожного класу з 500 наявних зображень необхідно створити ще 4,500. Це говорить що на основі кожного елементу навчальної вибірки буде створено 9 штучних. У випадку датасету MNIST, є 10 класів, що відповідають цифрам від 0 до 9, тобто буде створено 45,000 штучних зображень.

Для аугментації даних використовуються методи skimage.transform.rotate(), transform.SimilarityTransform(scale), а також transform.SimilarityTransform(translation). Ці методи виконують поворот на випадкову величину від 0 до 360 градусів, масштабування від 80% до 150%, та зсув по осям абсцис та ординат на величину від -5 до 5 пікселів (нагадаємо, що розмір зображень датасету складає 28 на 28 пікселів).

Для візуалізації роботи алгоритму, приведено рисунок 16 та 17. На першому з них (рис. 3.1) результат роботи алгоритму при створенні 32 штучних зображень з усього лише одного оригінального зображення. На другому рисунку (рис. 3.2) зображено результати аугментації з 8 оригінальних зображень. Як бачимо, навчальні дані у другому випадку більш різноманітні та такі, що дадуть кращі результати при навчанні нейронної мережі через значно більшу ентропію.

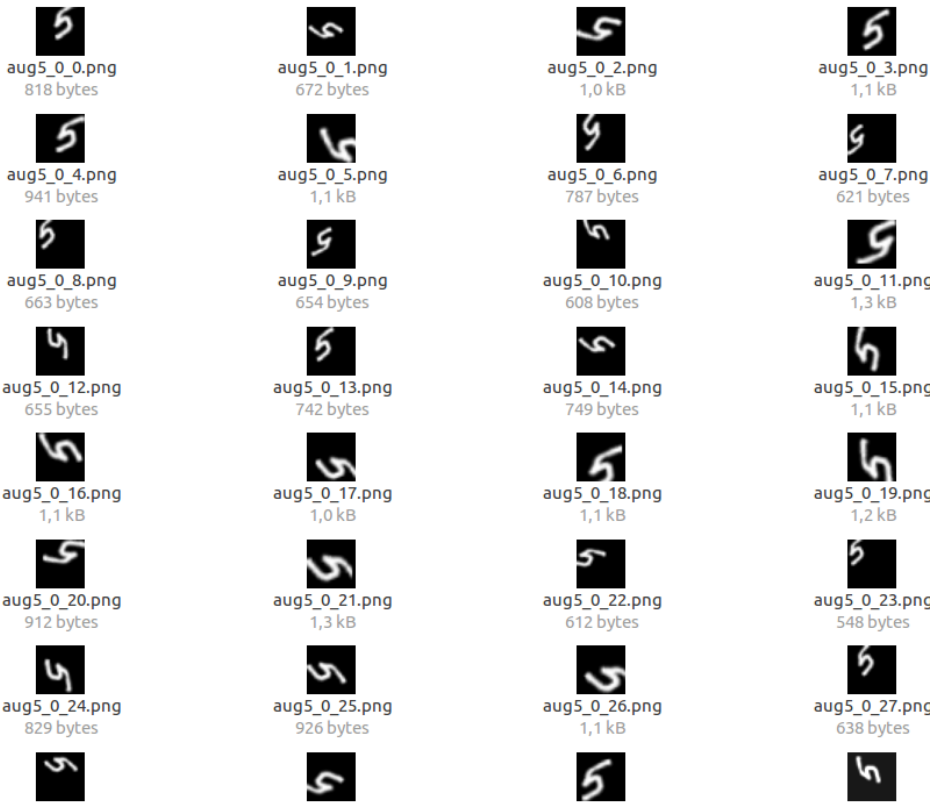


Рис. 3.1 Результат роботи алгоритму аугментації для створення 32 штучних зображень на основі одного оригінального

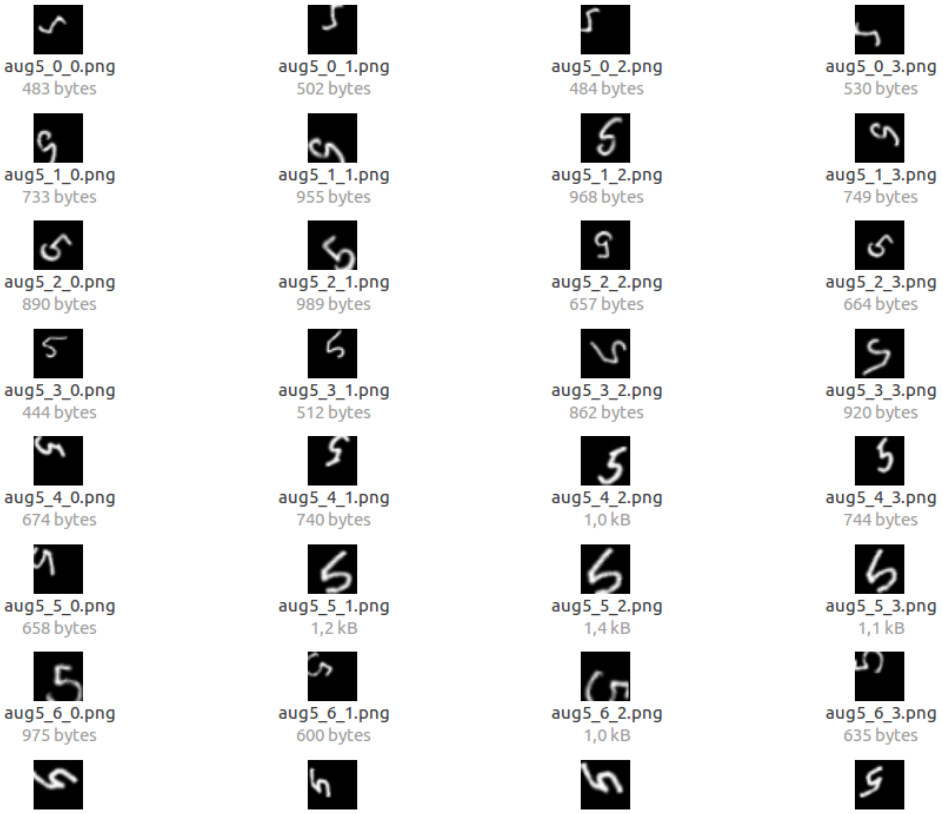


Рис. 3.2 Результат роботи алгоритму аугментації для створення 32 штучних зображень на основі 8 оригінальних

3.3 Дослідження підходу для мереж типу CNN та MLP

Для дослідження ефективності штучного збільшення навчальної вибірки, було взято 500 зображень кожного класу оригінального датасету. Цим моделюється ситуація, при якій доступна лише досить обмежена кількість оригінальних зображень для навчання нейронної мережі. Основною ідеєю є те, що штучне збільшення доступної множини зображень за допомогою методів аугментації з обмеженої кількості оригінальних зображень, дасть приріст в якості класифікації нових прикладів, які не пропускалися через мережу під час навчання. Для цього було проведено штучне збільшення датасету з 500 зображень для кожного класу до 1000 зображень для кожного класу (500 оригінальних та 500 штучно створених) та проведене навчання нейронної мережі типу CNN. Цей дослід було проведено тричі для отримання трьох результатів. Далі таким же чином було проведено навчання на вибірках в 1500

елементів (500 оригінальних та 1000 штучних), 2000 та так далі до 5000 (500 оригінальних та 4500 штучних).

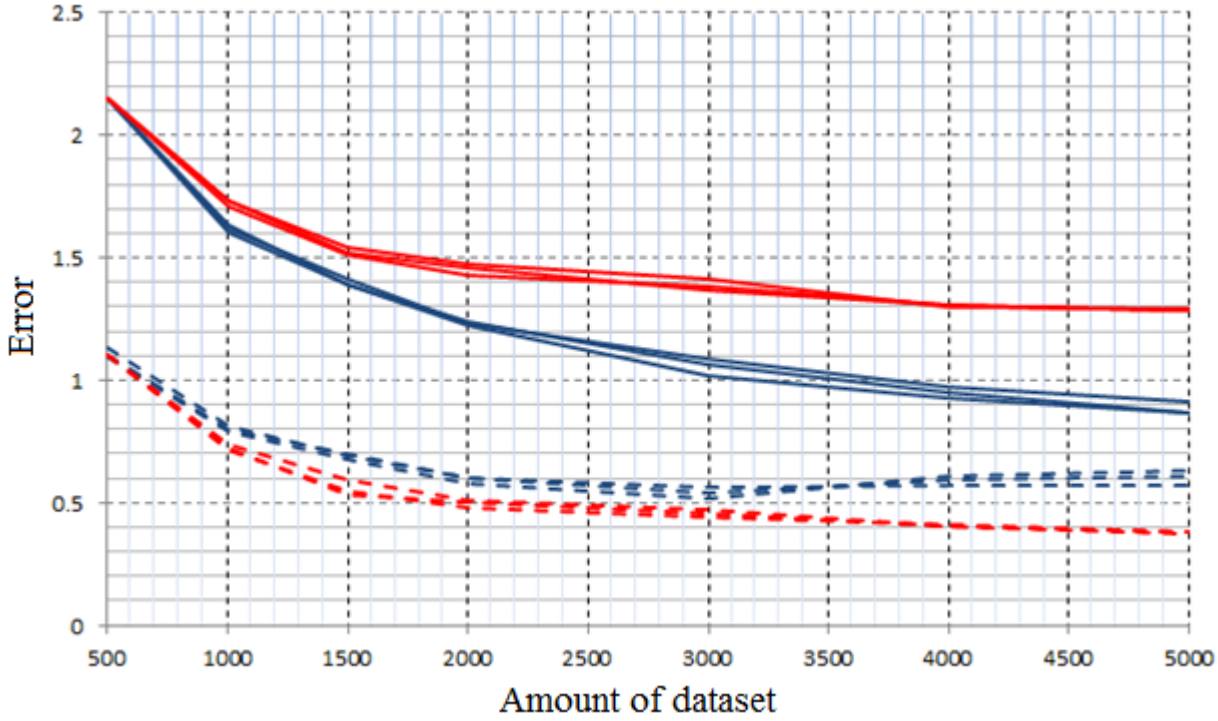


Рис. 3.3 Залежність тестової (суцільна лінія) та тренувальної (пунктирна лінія) похибки нейронної мережі типу CNN від кількості елементів в тренувальній множині. Синя лінія показує оригінальну навчальну вибірку. Червона лінія показує вибірку, що була отримана шляхом штучного збільшення навчальних даних.

На рисунку 3.3 показана продуктивність CNN при навчанні на датасеті, що містить частину штучно створених даних, а саме залежність похибки на навчальній та тренувальній множині в залежності від об’єму датасету. Як і в попередньому випадку, ми бачимо, що збільшення кількості навчальних даних покращує результати класифікації мережі. Найважливішим є те, що величина похибки на тестовій множині спадає протягом усього процесу збільшення доступних навчальних даних за допомогою штучних зображень. Декілька послідовних експериментів, як бачимо, показали досить схожі та стабільні результати. На відміну від навчання на повністю оригінальних даних, навчання

на частково штучних даних привело до того, що похибка на тренувальній множині продовжує спадати з збільшенням розмірів датасету, так само як похибка на тестовій множині.

Як результат, бачимо, що похибка на тестовій множині для навчального набору 500+4500 вийшла в результат приблизно 1,3%, що, в свою чергу, відповідає набору повністю реальних даних в кількості приблизно 1750 елементів. Таким чином, за допомогою методів аугментації даних, ми зменшили кількість необхідних реальних прикладів приблизно в 3.5 рази для досягнення того ж результату навчання мережі.

Далі таким же чином було проведено навчання на вибірках в 500, 1000, 1500, 2000 елементів та так далі до 5000 (500 оригінальних та 4500 штучних) для багатошарового персептрону (MLP).

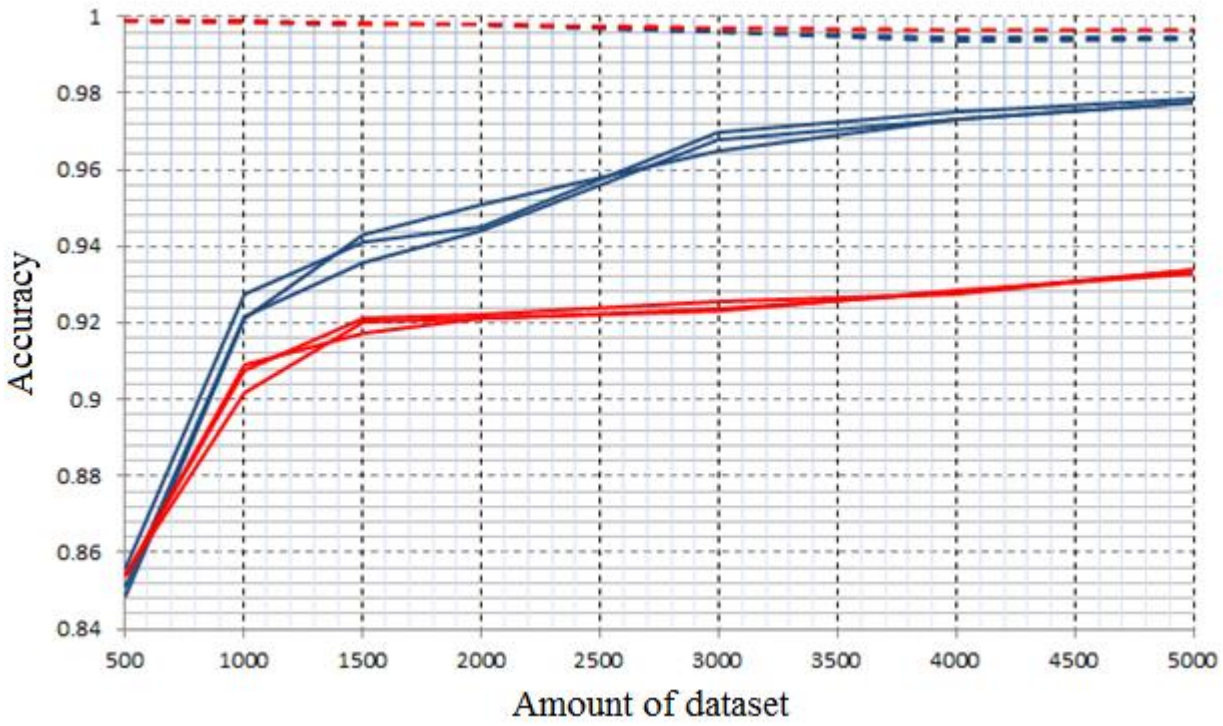


Рис. 3.4 Залежність точності класифікації тестової (суцільна лінія) та тренувальної (пунктирна лінія) вибірки для нейронної мережі типу MLP від кількості елементів в тренувальній множині. Синя лінія показує оригінальну навчальну вибірку. Червона лінія показує вибірку, що була отримана шляхом штучного збільшення навчальних даних.

На рисунку 3.4 показана продуктивність MLP при навчанні на множині, що містить частину штучно створених даних, а саме залежність точності класифікації на навчальній та тренувальній множині в залежності від об’єму датасету. Як і в попередньому випадку, ми бачимо, що збільшення кількості навчальних даних покращує результати класифікації мережі. Найважливішим є те, що величина точності класифікації на тестовій множині спадає протягом усього процесу збільшення доступних навчальних даних за допомогою штучних зображень. Декілька послідовних експериментів, як бачимо, показали досить схожі та стабільні результати. Від експерименту до експерименту, мережа демонструє більш передбачувані результати на штучних даних, ніж на реальних. На відміну від навчання на повністю оригінальних даних, навчання на частково штучних даних привело до того, що точність класифікації на тренувальній множині практично не змінюється з збільшенням розмірів датасету.

Як результат, бачимо, що точність на тестовій множині для навчального набору 500+4500 вийшла в результат приблизно 0.933, що, в свою чергу, відповідає набору повністю реальних даних в кількості приблизно 1300 елементів. Таким чином, за допомогою методів аугментації даних, ми зменшили кількість необхідних реальних прикладів приблизно в 2.6 рази для досягнення того ж результату навчання мережі. Зауважимо, що точність класифікації, на відміну від CNN, не досягла асимптотичного рівня, а продовжувала зростати з набором штучних даних та, можливо, вийшла б на коефіцієнт 3.5, як і згорткова мережа.

4 РОЗГЛЯД МІКРОКОНТРОЛЛЕРІВ МЕРЕЖ INTERNET OF THINGS

4.1 Вступ

Описані вище алгоритми оптимізації навчання нейронної мережі націлені на зменшення використовуваних для цього процесу ресурсів, як процесорних, так і затрачуваних об’ємів оперативної та статичної пам’яті. Так чи інакше можливості IoT пристроїв залежать від апаратної платформи на якій вони побудовані. Для вирішення сучасних завдань, які ставляться перед такими пристроями, доцільно використовувати мікроконтролери або міні комп’ютери. У цьому розділі розглянемо популярні рішення, які існують на ринку, а також їх можливості.

Обрані пристрої цілком підходять для використання в поєднанні з штучними нейронними мережами, як з точки зору доступних апаратних засобів та їх потужності, так і з точки зору програмних засобів, що дозволяють успішне виконання обраного алгоритму штучного збільшення навчальних даних.

5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

5.1 Вступ

Стартап як форма малого ризикового (венчурного) підприємництва впродовж останнього десятиліття набула широкого розповсюдження у світі через зниження бар'єрів входу в ринок (із появою Інтернету як інструменту комунікацій та збуту стало простіше знаходити споживачів та інвесторів, займатись пошуком ресурсів, перетинати кордони між ринками різних країн), і вважається однією із наріжних складових інноваційної економіки, оскільки за рахунок мобільності, гнучкості та великої кількості стартап-проектів загальна маса інноваційних ідей зростає.

Проте створення та ринкове впровадження стартап-проектів відзначається підвищеною мірою ризику, ринково успішними стає лише невелика частка, що за різними оцінками складає від 10% до 20%. Ідея стартап-проекту, взята окремо, не вартує майже нічого: головним завданням керівника проекту на початковому етапі його існування є перетворення ідеї проекту у працюючу бізнес-модель, що починається із формування концепції товару (послуги) для визначеної клієнтської групи за наявних ринкових умов.

Розроблення та виведення стартап-проекту на ринок передбачає здійснення низки кроків, в межах яких визначають ринкові перспективи проекту, графік та принципи організації виробництва, фінансовий аналіз та аналіз ризиків і заходи з просування пропозиції для інвесторів. Узагальнено етапи розроблення стартап-проекту можна подати таким чином:

1. Маркетинговий аналіз стартап-проекту

В межах цього етапу:

- розробляється опис самої ідеї проекту та визначаються загальні напрями використання потенційного товару чи послуги, а також їх відмінність від конкурентів;
- аналізуються ринкові можливості щодо його реалізації;
- на базі аналізу ринкового середовища розробляється стратегія

ринкового впровадження потенційного товару в межах проекту.

2. Організація стартап-проекту

В межах цього етапу:

- складається календарний план-графік реалізації стартап-проекту;
- розраховується потреба в основних засобах та нематеріальних активах;
- визначається плановий обсяг виробництва потенційного товару, на основі чого формулюється потреба у матеріальних ресурсах та персоналі;
- розраховуються загальні початкові витрати на запуск проекту та планові загальногосподарські витрати, необхідні для реалізації проекту.

3. Фінансово-економічний аналіз та оцінка ризиків проекту

В межах цього етапу:

- визначається обсяг інвестиційних витрат;
- розраховуються основні фінансово-економічні показники проекту (обсяг виробництва продукції, собівартість виробництва, ціна реалізації, податкове навантаження та чистий прибуток) та визначаються показники інвестиційної привабливості проекту (запас фінансової міцності, рентабельність продажів та інвестицій, період окупності проекту);
- визначається рівень ризикованості проекту, визначаються основні ризики проекту та шляхи їх запобігання (реагування на ризики).

4. Заходи з комерціалізації проекту

Цей етап спрямовано на пошук інвесторів та просування інвести-ційної пропозиції (оферти). Він передбачає:

- визначення цільової групи інвесторів та опису їх ділових інтересів;
- складання інвест-пропозиції (оферти): стислої характеристики проекту для попереднього ознайомлення інвестора із проектом;
- планування заходів з просування оферти: визначення

комунікаційних каналів та площадок та планування системи заходів з просування в межах обраних каналів;

- планування ресурсів для реалізації заходів з просування оферти.\

Означені етапи, реалізовані послідовно та вчасно – створюють передумови для успішного ринкового старту. Проте фахівці зі створення та розвитку стартаппроектів окремо відзначають, що відсутність маркетингових знань та умінь, що уможливлюють розробку ринково затребуваного проекту із вихідної ідеї, є основною причиною високого рівня банкрутств стартап-компаній, і ця проблема може бути вирішена за рахунок навчання винахідників. Відповідно, основним призначенням даних Методичних рекомендацій є надання студентам знань щодо суті, основних принципів розроблення стратегії ринкового впровадження та маркетингового управління інноваційними стартаппроектами у промислових галузях економіки, використання ефективних маркетингових інструментів просування високотехнологічних продуктів виробництва та послуг.

5.2 Опис ідеї проекту

В межах підпункту було проаналізовано і подано у вигляді таблиць:

- зміст ідеї (що пропонується);
- можливі напрямки застосування;
- основні вигоди, що може отримати користувач товару (за кожним напрямком застосування);
- чим відрізняється від існуючих аналогів та замінників;

Перші три пункти подані у вигляді таблиці (табл. 1) і дають цілісне уявлення про зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Таблиця 5.1 Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*

Програмне забезпечення для штучного покращення якості датасету для навчання нейронних мереж	1. Використання при проектуванні промислових нейронних мереж	1. Покращення якості датасету
	2. Використання для нейронних мереж в концепції Інтернет речей	2. Зменшення необхідної кількості навчального матеріалу для нейронної мережі для тієї ж якості навчання
		3. Пришвидшення процесу збору навчальних даних

Таблиця 5.2 Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	*Техніко-економічні характеристики ідеї*	*(потенційні) товари/концепції конкурентів*		*W (слабка сторона)*	*N (нейтральна сторона)*	*S (сильна сторона)*
		Мій проект	*Контроллер*			
1.	Кросплатформеність	Можливість використання на будь-яких пристроях, що підтримують Python	Вузька заточеність під архітектуру			+
2.	Собівартість	Низька	Висока			+

3.	Зручність використання	Повна автоматизація роботи, гнучке налаштування	Можливість віддаленного налаштування	+		
4.	Ефект розширення датасету	Високий	Середній			+

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

5.3 Технологічний аудит ідеї проекту.

В межах даного підрозділу було проведено аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару).

Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових (табл. 3):

- за якою технологією буде виготовлено товар згідно ідеї проекту?
- чи існують такі технології, чи їх потрібно розробити/доробити?
- чи доступні такі технології авторам проекту?

Таблиця 5.3 Технологічна здійсненність ідеї проекту

№ п/п	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
	Dataset augmentation	Python Image Library (PIL)	Є у наявності	Доступно на більшості платформах.

	Dataset augmentation	Skimage library	Є у наявності	Доступно на більшості платформах.
	Dataset augmentation	Matplotlib	Є у наявності	Доступно на більшості платформах
	Usability improvement	Glob Python library	Є у наявності	Доступно на усіх платформах на базі Linux.
Обрана технологія реалізації ідеї проекту: Python Image Library (PIL)+ Skimage library + Glob Python library				

За результатами аналізу таблиці зроблено висновок, про можливість реалізації проекту.

5.4 Аналіз ринкових можливостей запуску стартап-проекту.

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Спочатку було проведено аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (табл. 5.4).

Таблиця 5.4 Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1	Кількість головних гравців, од	4
2	Загальний обсяг продаж, грн/ум.од	8 350 000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі , %	R = 41%

Середню норму рентабельності в галузі було порівняно із банківським відсотком на вкладення. Останній є меншим, тому є сенс вкладати гроші саме у цей проект.

За результатами аналізу таблиці було зроблено висновок, що ринок є привабливим для входження.

Надалі були визначені потенційні групи клієнтів, їх характеристики, та зформовано орієнтовний перелік вимог до товару для кожної групи (табл. 5).

Таблиця 5.5 Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*

	Потреба в покращенні якості датасету для навчання штучних нейронних мереж	Розробникик проектів та програмних додатків з використання нейронних мереж, що спеціалізуються на класифікації зображень	Використання нейронних мереж для класифікації інших об’єктів, окрім графічних	Рішення має бути крос-платформеним та інтуїтивно-зрозумілим

Після визначення потенційних груп клієнтів було проведено аналіз ринкового середовища: складено таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. 5.6-5.7).

Таблиця 5.6 Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
	Конкуренція	Вихід на ринок одного з гігантів сумісних областей з комплексним програмним рішенням, що міститиме у собі аналог нашого продукту	1. Передбачити додаткові переваги власного проекту для того, щоб повідомити про них саме після виходу міжнародної компанії на ринок. 2. Обрати нову цільову аудиторію і зосередитися на ній
	Економічний	Подорожчання спеціалізованого обладнання для роботи з штучними	Оптимізація програмного продукту, для можливості його запуску на більш бюджетних пристроях.

		нейронними мережами в концепії Інтернету речей	

Таблиця 5.7 Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
	Науково-технічний	Тенденція до випуску покращенного спеціалізованого обладнання для роботи в обраній галузі	Адаптація існуючого рішення і алгоритмів під нову технологію.
	Попит	Більш широке розповсюдження тезнології нейронних мереж	Постійна підтримка продукту.

Надалі було проведено аналіз пропозиції: визначили загальні риси конкуренції на ринку (табл. 5.8).

Таблиця 5.8 Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Вказати тип конкуренції:	Існує декілька фірм-конкурентів.	Підтримка якості продукту та постійні

монополістична конкуренція.		нововведення.
2. За рівнем конкурентної боротьби: Міжнародний.	Фірми-конкуренти знаходяться в інших країнах.	Адаптація продукту як для вітчизняних так і для зарубіжних клієнтів.
3. За галузевою ознакою: внутрішньогалузева.	Продукт використовується лише всередині даної галузі.	Постійне вдосконалення продукту.
4. Конкуренція за видами товарів: товарно-видова.	Види товарів однакові.	Створити продукт, враховуючи сильні і слабкі сторони конкурентів.
5. За характером конкурентних переваг: нецінова.	Вдосконалення технології створення контроллерів.	Зниження ціни на продукт та підтримка його якості.
6. За інтенсивністю: марочна.	Бренди існують і конкурують.	PR, реклама, просування бренду.

Таблиця 5.20 Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники*	*Рівень цін на товари-аналоги*	*Рівень доходів цільової групи споживачів*	*Верхня та нижня межі встановлення ціни на товар/послугу*
	150-500$	170-550$	7000$	10-80$

Наступним кроком є визначення оптимальної системи збуту, в межах якого було прийняте рішення (табл. 5.21):

- проводити збут власними силами і залучати сторонніх посередників.
- користуватися однорівневим каналом збуту;

Таблиця 5.21 Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
	Одна одиниця на особу	Роздрібна торгівля	Однорівневий	Власні сили та через посередників

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 5.22).

Таблиця 5.22 Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуються цільові клієнти*	*Ключові позиції, обрані для позиціонування*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
	Клієнти обиратимуть зручніший товар з потрібними функціями.	Соціальні мережі, електронна пошта, мобільні телефони	Ціна, простота використання, кросплатформеність, більш зрчне та гнучке налаштування	Показати переваги продукту, низьку ціну, ефективність роботи для великої кількості випадків.	Демо ролик з використанням, реклама.

Результатом підрозділу стала ринкова (маркетингова) програма, що

включає в себе концепції товару, збуту, просування та попередній аналіз можливостей ціноутворення, спирається на цінності та потреби потенційних клієнтів, конкурентні переваги ідеї, стан та динаміку ринкового середовища, в межах якого впроваджено проект, та відповідну обрану альтернативу ринкової поведінки.

5.7 Висновки.

В даному розділі було проведено аналіз програмного продукту у якості стартап проекту. Можна зазначити що у проекта є можливість комерціалізації, адже ринок технологій нейронних мереж динамічно розвивається, створюються нові додатки які, в свою чергу, стимулють попит на різноманітні допоміжні засоби для навчання нейронних мереж.

На ринку наявна монополістична конкуренція, існує декілька фірм-конкурентів, тому вихід на нього не буде легким. Проте проект є доволі конкурентноспроможним завдяки, в першу чергу, своїй низькій собівартості. Через те, що він є повністю програмним, його розробка не потребує витрат на різноманітні матеріали та обладнання, необхідні для виготовлення корпусу, схем, тощо.

Для впровадження ринкової реалізації проекту слід обрати альтернативу, яка передбачає розробку програмного продукту, а потім якісну рекламу та PR, сконцентровану навколо позитивних характеристиках даного програмного продукту, таких як низька ціна, більш істотний ефект покращення якості датасету, кросплатформеність, екологічність і т.д.

З огляду на проведений аналіз, можна чітко сказати, що подальша імплементація проекту є доцільною, адже він може знайти свою цільову аудиторію та зайняти місце на ринку.

ВИСНОВКИ

Технології машинного навчання та Інтернету речей вже набули широкої популярності та розповсюдження, все більше використовуються в різноманітних областях та з кожним днем збільшують спектр застосування у різних галузях. Велика кількість задач може успішно вирішуватися за допомогою нейронних мереж, а технологія IoT має значний потенціал та постійно розвивається. Поєднання цих технологій є досить ефективним та перспективним.

В ході дипломної роботи проводився аналіз процесу навчання нейронної мережі, з огляду на її використання на пристроях мереж Інтернету речей. Зазвичай ці контроллери мають досить обмежені ресурси, як в плані можливих обсягів здійснюваних обрахунків, так і доступної пам’яті. Тому була звернена особлива увага на кожен етап навчання, починаючи з найважливішого - збір та обробка навчальних даних. Виявлені та розглянуті основні проблеми, що притаманні датасетам, такі як незбалансованість, нерепрезентативність, низька ентропія.

Були створені тестові моделі найпопулярніших на сьогодні архітектур нейронних мереж, а саме convolutional neural network (CNN), тобто згорткової нейронної мережі та multilayer perceptron (MLP), тобто багатошарового персептрона. В якості тестового датасету було обрано MNIST, так як він вважається класичним набором для тестування можливостей класифікації графічних зображень та має обширну базу результатів для різноманітних моделей та архітектур нейронних мереж. Детально досліджена залежність точності класифікації кожною нейронною мережею від розмірів доступного набору навчальних даних. Зроблені висновки щодо високої залежності якості роботи мережі від об’єму навчальної вибірки.

Виходячи з цього провелося дослідження підходів до зменшення необхідної кількості навчальних даних для нейронних мереж різних архітектур для полегшення їх використання в технології Інтернету речей. Було поставлене

завдання виявлення способу ефективної мінімізації необхідної кількості навчальних даних для отримання задовільних результатів класифікації.

Була створена програмна реалізація алгоритму на мові Python для штучного збільшення навчального набору графічних зображень для нейронних мереж. Показані приклади створення набору штучних зображень з одного або декількох оригінальних.

Для перевірки спроможності реалізованого програмного засобу досягти поставленої цілі, тобто ефективної мінімізації необхідної кількості навчальних даних для отримання задовільних результатів класифікації нейронною мережею, було проведене навчання тестових моделей на датасеті, що складався з штучно створених елементів та лише з незначної кількості оригінальних зображень. Знайдена межа, при якій подальше додавання штучних даних не покращує якості класифікації мережі на тестовому наборі оригінальних зображень. Було виявлено значний ріст результатів класифікації мережею при незмінній кількості початкової оригінальної навчальної множини. Співставлення результатів попередніх експериментів та поточного показало, що результат при навчанні на такому наборі відповідає результату для значно більшого набору оригінальних зображень, ніж знадобилося для створення штучного набору.

Реалізувавши програмний засіб, були виявлені деякі місця для покращення додатку. Є можливості для оптимізації швидкості виконання, а також збільшення ентропії в створюваних штучних зображеннях.

Ціль дослідження була досягнута, результатом роботи є програмний засіб, за допомогою якого можливо значно зменшити необхідну кількість оригінальних навчальних даних для отримання задовільних результатів класифікації нейронною мережею.

Був зроблений детальний огляд контроллерів, що використовуються в мережах IoT та зроблений висновок, що їх ресурсів достатньо для роботи розробленого програмного засобу.

Технологія Інтернету речей розвивається дуже швидко, а її синтез з технологією нейронних мереж є логічним етапом цього розвитку. Це поєднання є ефективним та перспективним, а отже програмні засоби, подібні до розробленного в даній роботі, задачою яких є оптимізація взаємної роботи цих технологій, будуть необхідні в майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A. G. Howard, “Some improvements on deep convolutional neural network based image classification,” arXiv preprint arXiv:1312.5402, 2013.
2. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.
3. Arduino products official documentation; Інтернет-ресурс. Режим доступу: https://www.arduino.cc/en/Main/Products
4. Bhowan U. Genetic Programming for Classification with Unbalanced Data. Victoria University of Wellington. 2012, 270 pp.
5. C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, “Dbsmote: density-based synthetic minority over-sampling technique,” Applied Intelligence, vol. 36, no. 3, pp. 664–684, 2012.
6. Chawla N., Bowyer, K., Hall, L., Kegelmeyer, W. SMOTE: Synthetic Minority Over-sampling Technique // Journal of Artificial Intelligence Research, 2002, 16. – P. 341-378.
7. Chawla N. C4.5 and imbalanced datasets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure / N. Chawla // ICML 2003 Workshop on Imbalanced Datasets.
8. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.
9. F. Provost and T. Fawcett, ``Robust Classification for Imprecise Environments," Machine Learning, vol. 42/3, pp. 203-231, 2001.
10. Hara K., Nakayama K. Selection of minimum training data for generalization and on-line training by multilayer neural networks / Proc. IEEE ICNN'1996, Washington, DC, USA, 1996, Vol.1. pp.436-441.

11. He H., Garcia A. Learning from Imbalanced Data // IEEE transactions on knowledge and data engineering, vol. 21, no. 9, September 2009. – P. 1263-1284.
12. I. Sato, H. Nishimura, and K. Yokoi, “Apac: Augmented pattern classi- fication with neural networks,” arXiv preprint arXiv:1505.03229, 2015.
13. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” arXiv preprint arXiv:1512.03385, 2015.
14. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.
15. K. Woods, C. Doss, K. Bowyer, J. Solka, C. Priebe, and P. Kegelmeyer, ``Comparative Evaluation of Pattern Recognition Techniques for Detection of Microcalcifications in Mammography," International Journal of Pattern Recognition and Artificial Intelligence, vol. 7(6), pp. 1417-1436, 1993.
16. L. Yaeger, R. Lyon, and B. Webb, “Effective training of a neural network character classifier for word recognition.”
17. M. Kubat and S. Matwin, ``Addressing the Curse of Imbalanced Training Sets: One Sided Selection," in Proceedings of the Fourteenth International Conference on Machine Learning, (Nashville, Tennesse), pp. 179-186, Morgan Kaufmann, 1997.
18. M. D. McDonnell and T. Vladusich, “Enhanced image classification with a fast-learning shallow convolutional neural network,” arXiv preprint arXiv:1503.04596, 2015.
19. Nano Pi official documentation; Інтернет-ресурс. Режим доступу: http://www.nanopi.org/NanoPi-NEO-Air_Feature.html
20. Orange Pi official documentation; Інтернет-ресурс. Режим доступу: http://www.orangepi.org/
21. Overfitting in machine learning; Інтернет-ресурс. Режим доступу: https://en.wikipedia.org/wiki/Overfitting
22. Provost, F., & Fawcett, T. (2001). Robust Classification for Imprecise Environments. Machine Learning, 42/3, 203–231.

23. P. Domingos, ``Metacost: A General Method for Making Classifiers Cost-sensitive," in Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (San Diego, CA), pp. 155-164, ACM Press, 1999.
24. P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best practices for convolutional neural networks applied to visual document analysis,” in 2013 12th International Conference on Document Analysis and Recognition, vol. 2. IEEE Computer Society, 2003, pp. 958–958.
25. P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” arXiv preprint arXiv:1312.6229, 2013.
26. Rapsberry Pi official documentation; Інтернет-ресурс. Режим доступу: https://www.raspberrypi.org/
27. Receiver operating characteristic; Інтернет-ресурс. Режим доступу: https://en.wikipedia.org/wiki/Receiver_operating_characteristic
28. Sebastien C. Wong, Adam Gatt, Victor Stamatescu, Mark D. McDonnell, “Understanding data augmentation for classification: when to warp?”, in 2016, ArXiv: 1609.08764v2, 26 Nov 2016
29. X. Zhang, Y. Fu, A. Zang, L. Sigal, and G. Agam, “Learning classifiers from synthetic data using a multichannel autoencoder,” arXiv preprint arXiv:1503.03163, 2015.
30. Качановский Ю.П., Коротков Е.А. Предобработка данных для обучения нейронной сети // Фундаментальные исследования. – 2011. – № 12-1. – С. 117-120;
31. Прикладная статистика: Классификация и снижение размерности: Справ. Изд. / С. А. Айвазян, В. М. Бухштабер, И. С. Енюков, Л. Д. Мешалкин; Под ред. С. А. Айвазяна. – М.: Финансы и статистика, 1989. – 607с.: ил. с. 339
32. Технология Data Mining: Интеллектуальный Анализ Данных, Степанов Р. Г. – Казань, 2008. – с. 14