НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

"На правах рукопису"
УДК_004.93'12_

«До захисту допущено»
Завідувач кафедри

_____ В.П. Тарасенко
(підпис)

“___” _____ 2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 “Комп'ютерна інженерія”
спеціалізації “Спеціалізовані комп'ютерні системи”

на тему:
ПРОГРАМНІ ЗАСОБИ РОЗПІЗНАВАННЯ АКУСТИЧНОЇ ІНФОРМАЦІЇ В СЕНСОРНИХ МЕРЕЖАХ

Виконав: студент 6 курсу, групи КМ-63м
Гончаров Данил Андрійович _____
(підпис)

Науковий керівник доц., к.т.н. Петрашенко А.В. _____
(підпис)

Консультант з _____ доцент., к.т.н. Клятченко Я.М. _____
(підпис)

Рецензент доц., к.т.н. Волокита А.М. _____
(підпис)

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

Київ – 2018

Національний технічний університет України
“Київський політехнічний інститут”

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність 8.05010203 “Спеціалізовані комп‘ютерні системи”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.П. Тарасенко
“___” _____ _____ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

ГОНЧАРОВУ ДАНИЛУ АНДРІЙОВИЧУ

1. Тема дисертації: ПРОГРАМНІ ЗАСОБИ РОЗПІЗНАВАННЯ АКУСТИЧНОЇ ІНФОРМАЦІЇ В СЕНСОРНИХ МЕРЕЖАХ,
науковий керівник: Петрашенко Андрій Васильович, к.т.н., доцент,
затверджені наказом по університету від “22” березня 2018 року №986-с.

2. Термін подання студентом дисертації: “13” травня 2018 р.

3. Об’єкт дослідження: програмні засоби класифікації звукових сигналів із використанням нейронних мереж..

4. Предмет дослідження: алгоритми для розпізнавання звуку за допомогою машинного навчання.

5. Перелік задач, які потрібно вирішити:
- провести дослідження характеристик звуку, які можуть бути використаними для класифікації аудіосигналів;
- дослідити алгоритми машинного навчання, які можуть бути використаними для цієї задачі;
- дослідити можливість використання підходу перенесення навчання для класифікації аудіосигналів;
- розробити структуру нейромережі для розв'язання задачі класифікації звуку;
- провести тестування отриманого алгоритму;
- дослідити результати роботи алгоритму;

6. Перелік обов’язкового ілюстративного матеріалу:
- структура нейромережі;
- демонстраційні таблиці результатів роботи алгоритму;

- демонстраційна схема роботи нейромережі;
- структура схема програмного модулю;
- графіки ілюстрації процесу навчання алгоритму.

7. Перелік обов’язкових публікацій:
   - Стаття “Використання tranfer learning для класифікації звуку”

8. Дата видачі завдання: “__” _____ 2018 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	15.12.2016	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	01.03.2017	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.05.2017	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	15.10.2017	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.12.2017	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації; підготовка матеріалів доповіді на конференції ПМК-2016.	01.03.2018	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу	15.04.2018	
8.	Оформлення текстової і графічної частини магістерської дисертації	20.04.2018	
9.	Попередній розгляд магістерської дисертації	26.04.2018	

Студент _____ Гончаров Д. А.

Науковий керівник _____ Петрашенко А. В.

РЕФЕРАТ

Актуальність теми.

З урахуванням швидкого розвитку мікроконтролерів та появи дешевої елементної бази для датчиків та подальшого розповсюдження смартфонів кількість наявної аудіоінформації стрімко зростає. Як результат, зростає необхідність в розвитку систем, спрямованих на обробку цієї інформації в автоматизованому режимі. Однією з важливих задач є задача класифікації звукових сигналів, яка важлива для сенсорних систем, які мають обробляти інформацію, що поступає з датчиків в реальному часі. Наразі існує досить велика кількість алгоритмів, що направлені на вирішення цієї задачі, проте дослідження теми лишається актуальним.

Об’єктом дослідження є програмні засоби класифікації звукових сигналів із використанням нейронних мереж.

Предметом дослідження є алгоритми для розпізнавання звуку за допомогою машинного навчання, особливо нейронні мережі, використання підходу transfer learning для вирішення задачі класифікації звуку, шляхи знаходження характерних ознак звуку (feature extraction), що найкращим чином підходять до цих алгоритмів в цій предметній області.

Мета дослідження:

Підвищення ефективності розв'язання задачі класифікації звуку за допомогою зменшення часу навчання алгоритму згорткових нейронних мереж на задачі класифікації.

Задачі дослідження:

1. Аналіз наявних алгоритмів класифікації звуку.
2. Розробка алгоритму на базі згорткових нейронних мереж з використанням підходу transfer learning для розв’язання задачі класифікації звуку.
3. Дослідження отриманого алгоритму.

Методи дослідження базуються на використанні методів теорії ймовірності.

Наукова новизна одержаних результатів полягає в тому що запропонований метод дає змогу зменшити час навчання нейронної мережі для класифікації звуку, зберігаючи достатньо високу точність (близько 80%).

Практична цінність отриманих в роботі результатів полягає в тому, що розроблено модуль на основі запропонованого алгоритму для класифікації звуку, який має перевагу в швидкості навчання в порівнянні з альтернативними підходами.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на X науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2018 (Київ, 21-23 квітня 2018 р.).

Публікації. Результати роботи опубліковані в 1 науковій праці, з яких 1 – тези доповідей.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їхнє впровадження.

У першому розділі розглянуто наявні методи класифікації аудіосигналів та проведений огляд наявних досліджень, направлених на використання transfer learning стосовно звукових сигналів.

У другому розділі наведено поглиблений огляд згорткових нейронних мереж та огляд підходу transfer learning.

У *третьому* розділі розглянуто деталі реалізації алгоритму мовою Python, наведена структура нейромережі, що використовується, та

детальна інформація про дані, які використовувались для тестування алгоритму.

У четвертому розділі подані результати тестування алгоритму та аналіз цих результатів.

У висновках представлені результати проведеної роботи.

Робота представлена на 83 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: штучні нейронні мережі, transfer learning, аудіосигнали.

ABSTRACT

Theme urgency. Considering fast development of micro-controllers and cheap element base for sensors and further extension of smartphones the amount of available audio information rapidly growing. As a result, there is a growing need for the development of systems aimed to processing this information in an automated mode. One of the important tasks is the classification of audio signals, which is important for sensory systems that need to handle the information coming from the sensors in real time. There are currently quite a large number of algorithms for solving this problem, but the topic research remains relevant.

Object of research is software tools for classifying sound signals using neural networks.

Subject of research is the algorithms for recognition of sound through machine learning, especially neural networks, the use of the transfer learning approach for solving the problem of sound classification, the ways of finding the characteristic features of sound (feature extraction) that best suit these algorithms in this subject area.

Research objective: Increasing the efficiency of solving the problem of classification of sound by reducing the learning time of the algorithm of convolutional neural networks on the classification problem.

Specifically:

1. Analysis of available sound classification algorithms.
2. Development of algorithm based on convolutional neural networks using the transfer learning approach for solving the problem of sound classification.
3. Research of the algorithm.

Research methods. Methods in research are based on the probability theory methods.

Scientific novelty of the obtained results is that the proposed method makes it possible to reduce the training time of the neural network for the classification of sound, while preserving a sufficiently high accuracy (about 80%).

Practical value of the results obtained in the work is that the module is developed on the basis of the proposed algorithm for the classification of sound, which has the advantage of learning speed compared with alternative approaches.

Approbation. The basic points and results of the work were presented and discussed at the 10nd Conference of Masters and Postgraduate Students "Applied Mathematics and Computer", PMK-2018 (Kyiv, April 21-23, 2018).

Structure and content of the thesis. The master's thesis consists of an introduction, four chapters, conclusions and appendixes.

The introduction gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the research, shows the scientific novelty of the results obtained and the practical value of the work, provides information on the approbation of the results and their implementation.

In the first chapter the existing methods for the classification of audio signals and are analysed and provided an overview of existing studies aimed at using transfer learning in relation to audio signals.

The second chapter provides an in-depth review of convolutional neural networks and an overview of the transfer learning approach.

In the third chapter formulated the implementation details of the Python algorithm, describes the structure of the used neural network, and provides detailed information about the data used to test the algorithm.

The fourth chapter presents the results of testing the algorithm and analysis of these results.

In the conclusions the general conclusions on the presented thesis are given; the obtained results are analyzed.

The thesis is presented in 83 pages, it contains 30 references to the used information sources. 14 figures and 4 tables are given in the thesis.

Key words: artificial neural networks, transfer learning, audio signals.

РЕФЕРАТ

Актуальность темы. С учетом быстрого развития микроконтроллеров и появления дешевой элементной базы для датчиков и дальнейшего распространения смартфонов количество имеющейся аудиоинформации стремительно растет. Как результат, возрастает необходимость в развитии систем, направленных на обработку этой информации в автоматизированном режиме. Одной из важных задач является задача классификации звуковых сигналов, которая важна для сенсорных систем, обладающих необходимостью обрабатывать информацию, поступающую с датчиков в реальном времени. Сейчас существует достаточно большое количество алгоритмов, направленных на решение этой задачи, однако исследование темы остается актуальным.

Объектом исследования являются программные средства классификации звуковых сигналов с использованием нейронных сетей.

Предметом исследования являются алгоритмы для распознавания звука с помощью машинного обучения, в первую очередь нейронные сети, использование подхода transfer learning для решения задачи классификации звука, способы нахождения характерных признаков звука (feature extraction), наилучшим образом подходят к этим алгоритмов в этой предметной области.

Цель работы: повышение эффективности решения задачи классификации звука посредством уменьшения времени обучения алгоритма сверточных нейронных сетей на задачах классификации.

Задачи исследования:

1. Анализ существующих алгоритмов классификации звука.

2. Разработка алгоритма на базе сверточных нейронных сетей с использованием подхода transfer learning для решения задачи классификации звука.

3. Исследование полученного алгоритма.

Методы исследования базируются на использовании методов теории вероятности.

Научная новизна исследования заключается в том, что предложенный метод позволяет уменьшить время обучения нейронной сети для классификации звука, сохраняя достаточно высокую точность (около 80%).

Практическая ценность полученных в работе результатов заключается в том, что разработан модуль на основе предложенного алгоритма для классификации звука, который имеет преимущество в скорости обучения по сравнению с альтернативными подходами.

Апробация работы. Основные положения и результаты работы были представлены и обсуждались на X научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2018 (Киев, 21-23 апреля 2018).

Публикации. Результаты работы опубликованы в 1 научной работе, из которых 1 - тезисы докладов.

Структура и объем работы. Магистерская диссертация состоит из введения, четырех глав и выводов.

Во введении представлена общая характеристика работы, произведена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов и практическую ценность работы, приведены сведения об апробации результатов и их внедрение.

В первом разделе рассмотрены существующие методы классификации аудиосигналов и проведен обзор имеющихся исследований, направленных на использование transfer learning применительно к звуковым сигналам.

Во втором разделе приведен углубленный обзор сверточных нейронных сетей и обзор подхода transfer learning.

В третьем разделе рассмотрены детали реализации алгоритма на языке программирования Python, приведена структура нейросети, и подробная информация о данных, которые использовались для тестирования алгоритма.

В четвертом разделе представлены результаты тестирования алгоритма и анализ этих результатов.

В выводах представлены результаты проведенной работы.

Работа представлена на 83 листах, содержит ссылки на список использованных литературных источников.

Ключевые слова: искусственные нейронные сети, transfer learning, аудиосигналы.

4. Таблиця значень точності модифікації алгоритму з картами активації класів по епохам.
5. Структурна схема програми.
6. Блок-схема програми.
7. Графік функції втрат для модифікації алгоритму без карт активації класів.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

1. ШНМ — штучна нейронна мережа.
2. ЗНМ — згорткова нейронна мережа.
3. SVM — машина опорних векторів.
4. ILSVRC — ImageNet Large Scale Visual Recognition Challenge.
5. ReLU — rectified Linear Units.
6. ROC — receiver operating characteristic.
7. MFCC — мел-кепстральні характеристики звуку.

ВСТУП

З урахуванням швидкого розвитку мікроконтролерів та появи дешевої елементної бази для датчиків та подальшого розповсюдження смартфонів кількість наявної аудіоінформації стрімко зростає, відстаючи в об'ємах лише від відеоінформації. Музика, диктофонні записи, сигнали з датчиків, звук з відеозаписів, записи телефонних розмов — різноманітні аудіосигнали присутні в усьому навколишньому світі, та їх кількість зростає з кожною годиною. Навіть більше, ця кількість не просто лінійно зростає, а зростає скоріш експоненційно, вибухово, якщо подивитись на колосальні зміни, що прийшли в повсякденне життя за останні два десятки років.

Як результат, зростає необхідність в розвитку систем, спрямованих на обробку цієї інформації в автоматизованому режимі, без залучення людей. Системи з розпізнавання мови, системи з локалізації джерела звуку, системи автоматичної класифікації музики по жанрам, системи автоматичного розпізнавання небезпечних звуків (наприклад на виробництві) набувають все більшого розповсюдження. До того ж все більш поширеними (і ця тенденція навряд зміниться) стають такі розробки, що відносяться до інтернету речей, і багато з них використовують керування мовою.

Однією з важливих задач в області обробки аудіоінформації є задача класифікації звукових сигналів, яка важлива як для сенсорних систем, які мають обробляти інформацію, що поступає з датчиків в реальному часі, так і для систем, що мають обробляти велику кількість попередньо збереженої інформації.

Розв'язання цієї задачі є досить важливим для значного полегшення життя людей, наприклад за рахунок пришвидшення реакції відповідальних служб у випадку небезпечних ситуацій, або впровадження певних систем “розумного дому”, що реагують на певні типи звукової інформації, наприклад, на дзвінок у двері, або дзвін розбитого вікна, або гавкіт собаки. Не менш важливою галуззю є

технології, направлені на збереження природи та захист тварин від браконьєрів — адже система, що використовує інформацію, зібрану датчиками, що розташовані на певній природній області, що потребує охорони, може внести неоцінимий вклад через вчасну локалізацію і детекцію таких звуків, як звуки пострілу або вибухів. Також застосування системи класифікації аудіосигналів можуть знайти в охоронних системах, які теж можуть більш надійно працювати, обробляючи інформацію з мікрофонів, розташованих по периметру об'єкту, що охороняється. Інших можливих способів застосувати подібні технології — безліч.

Технології машинного навчання в цілому — це один з найцікавіших і найбільш ефективних способів вирішення подібних задач. З часів створення перших обчислювальних машин люди мріяли про те створення машини, яка буде навчатися та розв'язувати задачі. Ця мрія призвела до розвитку цілої галузі науки, відомої зараз як наука про штучний інтелект.

Штучні нейронні мережі наразі — один із найбільш популярних і вживаних розділів штучного інтелекту, що показав свою високу результативність на багатьох задачах, таких як розпізнавання зображень, машинний переклад, розпізнавання мови, та інших, не менш складних. Якісне розв'язання цих задач ще кілька десятків років тому вважалось вкрай важким, до моменту стрімкого вибухового розвитку глибоких штучних мереж. Дослідження їх можливостей та підходів, що базуються на таких мережах — також є вкрай важливим з точки зору подальшого розвитку науки, адже це сприяє їх подальшому розвитку.

У даній роботі пропонується використання підходу transfer learning, що показав свою результативність на задачах розпізнавання зображень стосовно задачі класифікації звуку. Цей підхід базується на переносі та застосуванні певних апріорних знань з однієї області до області, до якої ці знання початково не відносились. Цей підхід використовується для випадків, коли з певних причин достатня кількість знань з області, в якій розв'язується задача, відсутня. Іншою

причиною використання цього підходу може бути необхідність пришвидшити навчання алгоритму, що може мати сенс у випадку відсутності апаратних ресурсів на навчання алгоритму.

За допомогою такого підходу може бути значно пришвидшений процес навчання мережі для розв'язання задачі класифікації звуку за допомогою використання ваг нейромережі, що навчалась на зображеннях.

1.ПІДХОДИ ДО КЛАСИФІКАЦІЇ ЗВУКУ

1.1 Характеристики звуку для класифікації аудіосигналів

1.1.1 Огляд характеристик звуку

Наразі була розроблена досить велика кількість різних характеристик звуку, що можуть використовуватись для роботи з акустичними даними [1].

Прикладами задач, в яких ці характеристики можуть використовуватись, можуть виступати задачі класифікації звуку, задачі розпізнавання мови, задачі генерації подібного до заданого мовного сигналу, генерації заголовків для аудіосигналів, тощо.

1. Спектрально-часові ознаки:

1.1 Спектральні ознаки:

- середнє значення спектра аналізованого сигналу;
- нормалізовані середні значення спектра;
- відносний час перебування сигналу в смугах спектра;
- нормалізований час перебування сигналу в смугах спектра;
- медіанне значення спектра сигналу в смугах;
-відносна потужність спектра сигналу в смугах;
- будь-які інші ознаки сигналу, отримані безпосередньо з спектру сигналу

1.2 Часові ознаки:

- тривалість сегменту, звуку;
- висота сегменту;

Спектрально-часові ознаки характеризують сигнал в його фізико-математичної суті, виходячи з наявності компонент трьох видів:

1. Періодичних (тональних) ділянок звукової хвилі;
2. Неперіодичних ділянок звукової хвилі (шумових);

ділянок, що не містять пауз.

2. Кепстральні ознаки

-мел-частотні кепстральні коефіцієнти;

-коефіцієнти лінійного передбачення з корекцією на нерівномірність чутливості людського вуха;

-коефіцієнти потужності частоти реєстрації;

-коефіцієнти спектра лінійного передбачення;

-коефіцієнти кепстра лінійного передбачення.

3. Амплітудно-частотні ознаки:

- інтенсивність, амплітуда;

- енергія;

- частота основного тону (чот);

- формантні частоти;

- джиттер (jitter) - тремтіння частотна модуляція основного тону (шумовий параметр);

- шиммер (shimmer) - амплітудна модуляція на основному тоні (шумовий параметр);

- радіальна базисна ядерна функція;

- нелінійний оператор Тигер;

Амплітудно-частотні ознаки дозволяють отримувати значення, які можуть змінюватися в залежності від параметрів дискретного перетворення Фур'є (виду і ширини вікна) або незначних зрушень вікна по вибірці.

З точки зору акустики, сигнал є поширюваними в повітряному середовищі складними за своєю структурою звуковими коливаннями, які характеризуються частотою (числом коливань в секунду), інтенсивністю (амплітудою коливань) і тривалістю. Амплітудно-частотні ознаки несуть необхідну і достатню інформацію для розпізнавання звукового сигналу.

4. Ознаки нелінійної динаміки:

- відображення Пуанкаре;

- рекурентний графік;
- максимальний характеристичний показник Ляпунова;
- фазовий портрет (атрактор);

Для групи ознак нелінійної динаміки мовної сигнал розглядається як скалярна величина, яка спостерігається в системі голосового тракту людини.

Процес мовоутворення можна вважати нелінійним і аналізувати його методами нелінійної динаміки[1]. Завдання нелінійної динаміки полягає в знаходженні і дослідженні базових математичних моделей і реальних систем, які виходять з найбільш типових ідей про властивості окремих елементів, що складають систему, і законів взаємодії між ними.

В даний час методи нелінійної динаміки базуються на фундаментальної математичної теорії, в основі якої лежить теорема Такенса, яка підводить математичну основу під ідеї нелінійної авторегресії і доводить можливість відновлення фазового портрета атрактора по часовому ряду або по одній його координаті. (Під атрактором розуміють множину точок або підпростір в фазовому просторі, до якого наближається фазова траєкторія після загасання перехідних процесів.)

Оцінки характеристик сигналу з отриманих мовних траєкторій використовуються в побудові нелінійних детермінованих фазово-просторових моделей спостережуваного часового ряду. Виявлені відмінності в формі атракторів можна використовувати для діагностичних правил і ознак, що дозволяють, наприклад, розпізнати і правильно ідентифікувати різні емоції в емоційно забарвленому мовному сигналі.

Найчастіше використовуються спектральні ознаки та мел-кепстральні характеристики (MFCC).

MFCC – це представлення енергії спектра сигналу, яка базується на особливостях сприйняття людським вухом звуку.

Доведено, що чутливість людини до звукового сигналу залежить від його частоти: чим нижче частота, тим чутливість вище. У 1937 році була виведена формула, за допомогою якої можна перевести частоту в герцах в частоту в Гаммелах[2].

Мел - спеціальна одиниця вимірювання частоти звуку, заснована на ста-статистично обробці великого числа даних про суб'єктивне сприйняття висоти звукових тонів.

Формули переводу частоти в герцах (f) і в Гаммелах (m) представлені нижче:

$$m = 1127.01048 \ln(1 + f/700) \ ,$$

$$f = 700(e^{m/1127.01048}) \ ,$$

Сигнал подається як згортка двох функцій: вихідного сигналу і фільтра, параметри якого ми хочемо оцінити. Ми намагаємось виділити ці окремі компоненти. Для цього нам необхідно перетворення, яке б перетворювало згортку в суму:

$$x * h \rightarrow \hat{x} + \hat{h} \ ,$$

Для цього вводиться перетворення кепстра[2].

$C[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \ln|X(e^{iw})| e^{iwn} dw$ - дійсний вектор.

$C[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \ln(X(e^{iw})) e^{iwn} dw$ - комплексний вектор.

Схема перетворень сигналу буде виглядати наступним чином:

1. Перетворення Фур'є: $x * h \rightarrow XH$
2. Перетворення кепстра: $XH \rightarrow \hat{X} + \hat{H}$
3. Зворотне перетворення Фур'є: $\hat{X} + \hat{H} \rightarrow \hat{x} + \hat{h}$

Таким чином, розділення на джерело і фільтр відбувається само собою.

Переваги використання MFCC полягають у наступному:

- використовується знання про хвильовий характер сигналу (за рахунок неявного використання спектрової інформації);

- за рахунок проектування спектру на мел-шкалу виділяються найбільш важливі для людського сприйняття частоти.
- кількість мел-кепстральних характеристик сигналу може бути вибрана довільно. Це дозволяє масштабувати обсяг оброблюваної інформації за рахунок “стиснення” звукового фрейму.

1.2 Алгоритми машинного навчання для класифікації звуку

Класифікація - один з розділів машинного навчання, присвячений розв'язанню наступної задачі: в наявності множина об'єктів (ситуацій), розділених деяким чином на класи[3]. Задана певна обмежена множина об'єктів, для яких відомо, до якого класу вони відносяться. Ця множина називається навчальною вибіркою. Класова приналежність інших об'єктів не відома. Вимагається побудувати алгоритм, здатний класифікувати будь-який об'єкт початкової множини.

Класифікувати об'єкт — значить віднести об'єкт до певного класу, встановити між ними відповідність.

В математичній статистиці задачі класифікації також називаються задачами дискримінантного аналізу.

В машинному навчанні задача класифікації відноситься до розділу навчання з учителем. Існує також навчання без вчителя, коли розподіл об'єктів навчальної вибірки на класи не заданий, і потрібно класифікувати об'єкти лише на основі їх схожості між собою. У цьому випадку прийнято говорити про завдання кластеризації або таксономії, і класи називають, відповідно, кластерами або таксонами. Проте ці задачі в деяких випадках досить схожі між собою.

Класифікацію класифікують в залежності від особливостей даних або класів, з якими вона працює.

По даним класифікація буває наступна:

1. По ознакам - найпоширеніший випадок. Кожен об'єкт має сукупність своїх характеристик, що називаються ознаками. Ознаки можуть бути числовими або нечисловими.

2. За матрицями відстані між об'єктами. Кожен об'єкт описується відстанями до інших об'єктів навчальної вибірки. З цим типом вхідних даних працюють небагато методів, зокрема, метод найближчих сусідів, метод парензівного вікна, метод потенційних функцій.

3. Часовий ряд чи сигнал представляє собою послідовність вимірів у часі. Кожне вимірювання може бути представленим цифрою, вектором, а в загальному випадку - признаковим описом досліджуваного об'єкта в даний момент часу.

4. Зображення або відеоряд.

Зустрічаються і більш складні випадки, коли вхідні дані представляються у вигляді іншої інформації з більш складною структурою — в вигляді графів, текстів, тощо. Як правило, такі випадки зводяться до першого або другого випадку за допомогою різних методів попередньої обробки даних та виявлення ознак.

За класами задачу класифікації визначають як:

1. Класифікація з двома класами. Найбільш простий у технічному плані випадок, який служить основою для вирішення більш складних завдань.

2. Класифікація з багатьма класами. Коли кількість класів досягає великого числа (прикладом може слугувати задача розпізнавання ієрогліфів), задача класифікації стає значно складнішою.

Іншим варіантом визначення класифікації за класами буде:

1. Класифікація з класами, що не перетинаються.

2. Класифікація з класами, що перетинаться. Об'єкт може відноситися одночасно до декількох класів.

3. Класифікація з нечіткими класами. Вимагається визначити ступінь приналежності об'єкта кожному з класів, зазвичай це дійсне число від 0 до 1.

Задача класифікації звуку є класифікацією по ознакам (глобально кажучи, навіть класифікацію зображень коректніше відносити до класифікації по ознакам, адже значення кожного пікселя можливо розглядати як ознаки), багатокласовою (у більшості випадків) та класифікацію з неперетинаючимися класами (коректніше було б працювати з задачею перетинаючихся класів, але вона для даного випадку є значно більш складною, і що важливіше, вимагає відповідних наборів даних).

Для класифікації звуку може бути використана більшість алгоритмів машинного навчання, які здатні розв’язувати задачу класифікації.

Деякі з них будуть розглянуті далі:

1.2.1 Метод опорних векторів.

Метод опорних векторів (англ. support vector machine, SVM), є машинним алгоритмом, котрий навчається на прикладах та використовується для класифікації об’єктів[4].

Наприклад, SVM може розрізнити аварійний режим роботи електромеханічної системи та класифікувати його за наявності попередніх досліджень, можливих за технологічними вимогами режимів роботи. Такий підхід розкриває значні можливості для побудування адаптивних систем автоматичного керування. В основі SVM лежить деяка математична сутність – алгоритм максимізації деякої математичної функції відносно наявного набору даних.

Для розуміння того, як працює SVM, потрібно мати уявлення про чотири ключові поняття: – розділяюча гіперплощина (the separating hyperplane); – гіперплощина максимальної межі (the maximum-margin hyperplane); – м’яка межа (the soft margin); – функція ядра (the kernel function). Розділяюча

гіперплощина є математичною сутністю, що відділяє між собою класи об’єктів з однаковими ознаками.

Можна екстраполювати цю процедуру математично до багатовимірних просторів, розмірностей значно вищих за третю. Загальний термін для лінії, котра відділяє елементи різних класів, – багатовимірна гіперплощина.

SVM відрізняється від інших гіперплощинних методів класифікації тим, що він дозволяє обирати оптимальне розташування гіперплощини. Гіперплощина обирається таким чином, щоб бути розташованою на максимальній відстані від елементів кожного з класів, тобто посередині деякої зони, що відділяє між собою ці елементи.

В цьому полягає сутність другого ключового поняття – гіперплощина максимальної межі.

Об’єкти, що класифікуються, не завжди можуть бути розділені гіперплощиною. У реальних системах будуть наявними похибки в даних, внаслідок яких гіперплощина не виконає розподіл абсолютно точно. Тому для роботи методу SVM вводять допустиму похибку класифікації, що називається м’якою межею.

Зрозуміло, що метод опорних векторів не має робити багато помилок класифікації спостережень. Тому необхідне введення додаткового параметру для визначення кількості об’єктів, які можуть бути класифіковані неправильно (по інший бік гіперплощини максимальної межі) і як далеко вони можуть розташовуватись відносно неї. Іншими словами, вводиться м’яка межа похибки з обох боків розділяючої площини. Це пов'язане з тим, що об’єкти, які підлягають класифікації, не завжди можуть бути поділені лінійно.Часто вони не є такими, що допускають лінійне розподілення. Для вирішення проблеми лінійного розподілення вико- ристовують функції ядра, що проектують дані з низько- вимірного простору у багатовимірний. При вірному виборі функції

ядра об’єкти можуть бути розділені лінійно гіперплощиною у багатовимірному просторі. Таким чином, функції ядра виконують роль спрямляючого простору.

1.2.2 Логістична регресія

Логістична регресія або логіт-регресія (англ. Logit model) - це статистична модель, що використовується для передбачення ймовірності виникнення деякої події шляхом підгонки даних до логістичної кривої. Також вона вона може бути визначена як регресійна модель, що передбачає категоріальну змінну[3]. Незважаючи на назву, цей вид регресії використовується переважно для класифікаційних, а не регресійних задач.

Логістична регресія застосовується для передбачення ймовірності виникнення деякої події за значеннями безлічі ознак. Для цього вводиться так звана залежна змінна y, що приймає лише одне з двох значень - як правило, це числа 0 (подія не відбулася) і 1 (подія відбулася), і безліч незалежних змінних (також званих ознаками,або предикторами) — реальних значень $x_1, x_2, x_3, x_{4,\ldots}, x_n$, на основі значень яких потрібно обчислити ймовірність прийняття того чи іншого значення залежної змінної.

Робиться припущення про те, що ймовірність настання події y = 1 дорівнює:

$$P\{y=1|x\}=f(z) \tag{1}$$

де $z=\theta^T x=\theta_0+\theta_1 x_1+\ldots+\theta_n x_n$., x та θ — вектори-стовбці значень незалежних змінних $1, x_1, x_2, \ldots x_n$ і параметрів (коефіцієнтів регресії) — чисел θ_1, θ_2,...θ_n відповідно, а f(z) — логістична функція (іноді відома як сигмоїд, або логіт-функція):

$$f(z)=\frac{1}{1+e^{-z}} \tag{2}$$

З урахуванням того, що y приймає значення лише 0 та 1, ймовірність першого значення дорівнює:

$$P\{y=0|x\}=1-f(z)=1-f(\theta^T x) \tag{3}$$

Функцію розподілу y при даному x можливо записати в вигляді

$$P\{y|x\}=f(\theta^T x)^y(1-f(\theta^T x))^{1-y},\ y\in\{0,1\} \quad (4)$$

Фактично, це розподіл Бернуллі, з параметром, що дорівнює $f(\theta^T x)$

Підбір параметрів:

Підбір параметрів $\theta_1, \theta_2, \ldots, \theta_n$ виконується на основі деякої вибірки даних $(x^{(1)}, Y^{(1)}), \ldots, (x^{(m)}, Y^{(m)})$,

де $x^i \in R^n$ є вектором, що містить в собі значення незалежних змінних, а $Y^{(i)} \in \{0,1\}$ — відповідне до цього вектору значення Y , зазвичай виконується за допомогою методу максимальної правдоподібності, згідно до якого обираються такі значення θ , при яких значення функції правдоподібності максимальне на вибірці:

$$\hat{\theta}=\underset{\theta}{argmax}\, L(\theta)=argmax\prod_{i=1}^{m} Pr\{Y=Y^{(i)}|x=x^{(i)}\} \quad (5)$$

Максимізація функції правдоподібності означає і максимізацію її логарифма (що є більш зручним для обчислення):

$$\log L(\theta)=\sum_{i=1}^{m}\log Pr\{Y=Y^{(i)}|x=x^{(i)}\}=\sum_{i=1}^{m} Y^{(i)}\log\Lambda(\theta^T x^{(i)})+(1-Y^{(i)})\log(1-\Lambda(\theta^T x^{(i)})) \quad (6)$$

Для максимізації цієї функції може бути застосований будь-який відомий метод оптимізації, наприклад, метод градієнтного спуску, метод Нестерова (який є модифікацією методу градієнтного спуску), метод Ньютона, метод оптимізації рою частинок, тощо.

Ця модель знайшла широке застосування в багатьох сферах, особливо в областях науки, що відносяться до медицини та соціології через легку інтерпретацію отриманих результатів. Також вона часто використовується в банках для створення кредитних скорингів (через цю ж причину), економіці, та часто є базовою моделлю для подальшого вдосконалення при рішенні багатьох задач.

1.2.3 Метод k-найближчих сусідів

Метод k-найближчих сусідів[3] (англ. K-nearest neighbors algorithm, k-NN) - метричний алгоритм для автоматичної класифікації об'єктів або регресії.

У разі використання методу для класифікації об'єкту присвоюється той клас, який є найбільш поширеним серед k сусідів даного елемента, класи яких вже відомі. Можлива модифікація, яка полягає в присвоєнні більшої ваги до класу найближчих сусідів та меншої — до більш віддалених.

У разі використання методу для регресії, об'єкту присвоюється середнє значення по k найближчим до нього об'єктам, значення яких вже відомі.

Існує імовірнісний варіант методу k-NN. Можна оцінити імовірність того, що ознака належить до класу P, як частку k найближчих сусідів у класі P.

Параметр k у методі k-NN часто вибирається на підставі досвіду чи знань про розв'язувану задачу класифікації. Бажано, щоб параметр k був непарним, щоб зменшити імовірність «нічиєї».

Алгоритм може бути застосований до вибірок з великою кількістю атрибутів (багатовимірних). Для цього перед застосуванням потрібно визначити функцію дистанції. Класичний варіант визначення дистанції - дистанція в евклідовому просторі, манхетенська відстань, тощо.

Головним недоліком алгоритму є необхідність зберігати в пам'яті всі велику кількість даних. В деяких випадках це може бути критичним фактором проти використання методу найближчих сусідів. Одною з головних переваг алгоритму виступає швидкість процесу його тренування, в порівнянні з іншими методами машинного навчання, такими як штучні нейронні мережі, машина опорних векторів чи дерева рішень.

1.2.4 GMM класифікатор

Модель Гаусової суміші (англ. Gaussian Mixture Models) є імовірнісною моделлю, яка базується на припущенні, що всі спостереження, що належать до певних класів, утворюються з суміші скінченної кількості гаусових розподілів з невідомими параметрами [3]. Модель сумішей можна розглядати як узагальнення кластеризації k-means з включенням інформації про коваріаційну структуру даних і центри розподілів.

В цілому GMM модель цікава тим, що розглядає дані як результат лінійної комбінації певної кількості гаусових розподілів, в той час, як більшість інших методів статистичної класифікації використовують чи лише один певний розподіл, чи взагалі розглядають лише взаємовідносини між даними.

З урахуванням того, що саме нормальний розподіл є дуже розповсюдженим у реальному світі, вибір саме цього розподілу є досить очевидним при побудові моделі. До того ж, як витікає з центральної граничної теореми, у випадку, коли робота виконується з достатньо складними даними та кількість невідомих факторів достатньо велика, використання суміші нормальних розподілів є досить дескриптивним, тобто таким, що в багатьох випадках надійно описує дані.

Перед використанням цього класу моделей необхідно ввести термін «схована змінна», яка приймає наступні значення:

$$z = \begin{Bmatrix} 1, \textit{тоді } i-\textit{тий нормальний розподіл генерує } x_i \\ 0, \textit{в інакшому випадку} \end{Bmatrix}$$

Для кожного спостереження є своя схована змінна, яка описує, до якого саме нормального розподілу належить це спостереження.

Для знаходження центрів розподілів використовується EM-алгоритм (Expectation-maximization алгоритм). Цей алгоритм є ітераційним та

використовується для оцінки максимальної правдоподібності при обчисленні параметрів статистичної моделі з схованими змінними.

При використанні GMM моделі для класифікації часто використовується припущення, що спостереження різних класів належать до різних розподілів. Після отримання характеристик розподілів за допомогою ЕМ-алгоритму, для кожного спостереження можливо провести класифікацію, класифікувавши його до того класу, в розподілі якого в нього найбільша ймовірність.

1.2.5 Наївний баєсів класифікатор

Наївний баєсів класифікатор — ймовірнісний класифікатор, що використовує теорему Баєса для визначення ймовірності приналежності спостереження (елемента вибірки) до одного з класів при припущенні (наївному) незалежності змінних. [5]

В основі узагальненого байесовського класифікатора, який працює лежить наступна правило. Класифікатор вираховує апостеріорну ймовірність $P(k|x)$ кожного класу k, до якого може належати досліджуваний об'єкт, і відносить цей об'єкт до апостеріорно найбільш ймовірного класу $\hat{k}$:

$$\hat{k} = argmax \ln P(k|x_1, \dots x_m)$$

Апостеріорна ймовірність обчислюється за формулою Байеса:

$$P(k|x_1, \dots x_m) = P(k)\, p(x_1, \dots x_m|k) / p(k)$$

де $P(k)$ - апріорна ймовірність того, що об'єкт відноситься до k -того класу, $p(k)\, i\, p(x_1, \dots x_m|k)$ - безумовна та умовна багатовимірні щільності розподілу вектору ознак, компоненти якого звичайно статистично залежні.

Таким чином, байесовський класифікатор припускає, що багатовимірна спільна щільність розподілу ознак відома для всіх класів.

Аналітичне подання багатовимірної щільності ймовірностей відомо лише для нормального розподілу. Разом с тем багатовимірна нормальна щільність розподілу дає зручну модель для одного важливого випадку, а саме коли

значення вектора ознак x для даного класу k представляються неперервнозначними, трохи спотвореними версіями єдиного типового вектора, або вектора-прототипа, μ_k .

Саме цього очікують, коли класифікатор вибирається таким чином, щоб виділяти ті ознаки, які, різні для образів, що належать до різних класів, були б, можливо, більш схожі для образів з одного і того ж класу.

Багатовимірна нормальна щільність розподілу в загальному вигляді описується виразом:

$$P(x)=\frac{1}{(2\pi)^{\frac{m}{2}} det\, R^{\frac{1}{2}}} * e^{\frac{-1}{2}} * (x-\mu)^T R^{-1} (x-\mu) \quad ,$$

де μ - m -компонентний вектор середнього значення, R — коваріаційна матриця розміру $m \times m$, T - знак транспонування.

Відзначимо, що якщо все недіагональні елементи дорівнюють нулю, то $P(x)$ - зводиться до добутку одновимірних нормальних щільності компонент вектора x .

Тому для багатовимірного нормального розподілу можливо висловити в аналітично замкнутій формі (з точністю до несуттєвих доданків) алгоритм байєсівської класифікації:

$$\hat{k} = argmax\left(\ln P(k) - \frac{1}{2} \ln det\, R_k - \frac{1}{2} (x_k - \mu_k) R_k^{-1} (x_k - \mu_k)^T\right) \quad ,$$

де μ_k - m -вектор-рядок математичних очікувань значень ознак спостережень класу k , R_k — $m \times m$ -матриця ковариацій ознак класу k.

Діагональні елементи матриці утворюють m -вектор D_k дисперсій ознак об'єктів класу k .

Узагальнений байєсів класифікатор може бути трактований як оптимальний класифікатор. Аргументом для цього може бути твердження, що допоки можлива однозначна відповідь, він її дасть. Якщо ж однозначна

відповідь неможлива, результат узагальненого байєсова класифікатора буде кількісно характеризувати міру неоднозначності відповіді.

З іншого боку, для побудови такого класифікатора потрібна вибірка з усіма можливими комбінаціями змінних. Зрозуміло, що зі збільшенням кількості змінних розмір вибірки буде зростати експоненціально.

Для того, щоб подолати цю проблему, на практиці використовують певні припущення, які зводять залежність між кількістю змінних та необхідним розміром вибірки до лінійної.

У наївному байєсовому класифікаторі робиться припущення про незалежність ознак об'єкта. Якщо знехтувати статистичними зв'язками між компонентами вектора ознак, тоді матриця R_k буде діагональної з вектором D_k на головній діагоналі і узагальнений класифікатор стане наївним Байєсовим класифікатором.

Також робиться припущення, що маргінальна щільність розподілу $P(x_j|k)$ будь-якої ознаки є нормальною для будь-якого класу.

Але на практиці так буває далеко не завжди, дані, які спостерігаються, можуть не підкорюватись нормальному закону розподілу (в загальному випадку закон розподілу взагалі невідомий) і має місце статистична залежність, тому область застосування класифікатора звужується.

Додатковим аргументом на використання цього класифікатора може бути те, що на практиці нестроге виконання припущення про незалежність змінних (у випадку, якщо відхилення незначне) призводить лише до незначних втрат точності і як результат — переваги класифікатора (висока швидкість роботи, простота і масштабованість, помірні вимоги до пам'яті) часто переважують недоліки.)

1.2.6 Дерева рішень

Дерева рішень належать до самих популярних і потужних інструментів , що дозволяють вирішувати задачі класифікації. В основі роботи дерев рішень лежить процес рекурсивної розбивки вхідної множини спостережень або об'єктів на підмножини, асоційовані із класами.

Процес конструювання дерева рішень.

Побудова дерева рішень виконується в два етапи. Першим етапом є "створення дерева" (tree building), другим - "скорочення" дерева (tree prunning).

Вибір критерію розщеплення і зупинки навчання виконується під час першого етапу. У ході другого етапу виконується відсікання певних гілок дерева.

Критерій розщеплення.

Створення дерева виконується згори донизу, цей процес є ниспадаючим. Під час цього процесу алгоритм має знайти критерій розщеплення (відомий також як критерій розбивки), за котрим тренувальна множина розбивається на такі підмножини, які асоціюються з поточним вузлом. Кожен вузол має працювати з певним атрибутом (характеристикою) спостережень.

Головним правилом вибору атрибута є те, що він має розбивати дані на цьому вузлі таким чином, щоб об'єкти підмножин, отриманих після розбиття, відносились до одного класу, чи хоча б приблизно наближалися до такої розбивки. Тобто кількість кількість об'єктів з інших класів у кожному отриманому класі має бути мінімальною.

Серед усіх відомих критеріїв розщеплення найбільш відомими є міра ентропії та індекс Джіні (англ. Gini). Міра ентропії в даному контексті — це те, наскільки класи в вузлі є різноманітними. Після розбивки мають створюватись вузли з меншою різноманітністю станів вихідної змінної. Іншими словами, ентропія має спадати, а кількість інформації в вузлі має зростати.

Найкращим атрибутом для використання в дереві є такий, який максимально збільшує приріст інформації.

Розмір дерева не робить його більш ефективним. При збільшенні розміру дерева збільшується кількість описаних випадків. Отже, в середньому, менша кількість спостережень буде віднесена до кожного з класів. Такі великі дерева називають "гіллястими". Вони невиправдано великі, вхідна множина розбивається на дуже малі підмножини, яких дуже багато. Здатність до узагальнення в таких деревах спадає. Таке явище може бути розглянуте, як приклад перенавчання, побудовані моделі не завжди правильно працюють на нових даних, стабільно показуючи правильні результати на тренувальній вибірці.

Під час побудови дерева необхідно приймати певні міри, щоб дерево не стало завеликим. Для цього використовують спеціальні підходи для створення так званих "оптимальних" дерев, дерев оптимального розміру.

Для розуміння цих підходів необхідно дати визначення "оптимальності" дерева. Таке дерево має бути достатньо складним, щоб враховувати інформацію, яка наявна в тренувальній вибірці даних, проте в той же час таке дерево не має бути занадто складним.

Іншими словами, дерево має використовувати ту і тільки ту інформацію, яка покращує правильність роботи моделі і ігнорувати інформацію, яка не покращує правильність роботи.

Для цього використовують два основних підходи. Перший з них полягає до збільшення дерева до певного розміру, заданого користувачем алгоритму.

Другий підхід використовує певні методи для визначення "оптимального" розміру дерева. Серед таких методів — скорочення дерева через відсікання гілок (англ. pruning) та використання певних правил для визначення найкращого моменту зупинки алгоритму.

Варто зазначити, що не всі алгоритми конструювання дерев рішень працюють за однією схемою. Одні з них завжди послідовно використовують тільки два етапи, які вже згадувались вище ("побудова" дерева та його

"скорочення"), інші — чергують їх, "скорочуючи" дерево після "побудови", а потім знову "будуючи". Другий варіант сприяє запобіганню нарощуванню дерева, в той час як перший — нарощує дерево, а потім скорочує його.

Зупинка побудови дерева.

Отже, для визначення правила зупинки побудови дерева, необхідно сформулювати умови його виконання. Логічно, що таке правило має визначити, чи є поточний вузол кінцевим, чи є внутрішнім.

Зупинка — це момент припинення створення подальших розгалужень.

Існує кілька варіантів правил зупинки.

Перший з них — "рання зупинка". При цьому варіанті розглядається лише доцільність розбивки поточного вузла. Перевагою такого методу є значне прискорення процесу тренування моделі. Відповідним недоліком — є те, що дерево з таким критерієм зупинки може стати занадто "гіллястим" і втратити в точності класифікації.

Іншим варіантом є обмеження глибини дерева. При такому варіанті побудова дерева закінчується при досягненні певної, заданої користувачем, глибини.

Третім варіантом правила зупинки є завдання мінімальної кількості прикладів, що можуть знаходитись в кінцевих вузлах дерева. При досягненні ситуації, коли всі вузли дерева містять не менш ніж певне число об'єктів, тренування дерева закінчується і розгалуження припиняються.

Скорочення дерева або відсікання гілок.

Способом вирішити проблему з завеликою кількістю гіллок дерева є етап часткового їх відсікання.

Якість класифікатору, який представляє собою навчене дерево рішень, можно характеризувати точністю розпізнавання та помилкою розпізнавання. Відсікання гілок (або заміну якихось гілок піддеревом) логічно проводити тільки там, де після цієї процедури помилка дерева не збільшиться.

Цей процес іде від кінцевих вузлів дерева до його вершини. Скорочення дерева є більш популярним, ніж використання правил зупинки. Після процедури відсічення гілок дерево називається "усіченим".

1.2.7 Штучні нейронні мережі

Штучні нейронні мережі (ШНМ, англ. artificial neural networks, ANN), або конективістські системи (англ. connectionist systems) — це обчислювальні системи, що базуються на ідеях, що взяті з біологічних нейронних мереж, що складають мозок тварин.

Головною особливістю таких систем є те, що вони збільшують свою продуктивність на задачах (навчаються на них), під час розглядання прикладів (тренувальної вибірки), без спеціального програмування для рішення певної конкретної задачі.

Наприклад, для розпізнавання зображень чи відеоряду вони можуть навчитися класифікувати зображення, які містять певних тварин, наприклад собак, навчаючись на прикладах зображень, розмічених як “собака” та “не собака”. Отримані результати можуть бути використані для класифікації собак на зображеннях, які нейронна мережа не бачила. Це виконується без жодного попередньо відомого (апріорного) знання про собак, наприклад, що вони мають гострі (або не гострі) вуха, чотири лапи, хутро, хвіст, очі, і т.д. Замість використання таких характеристик штучні нейронні мережі створюють свій власний набір ознак, спираючись на навчальний матеріал, який вони оброблюють.

Структура ШНМ створюється на базі з'єднаних вузлів, які називають штучними нейронами (назва взята по аналогії до біологічних нейронів у мозку тварин). З'єднання між штучними нейронами здатні до передачі сигналу. Кожен нейрон, який отримує сигнал, може обробляти його певним чином, і передавати до інших нейронів, які мають зв'язки з ним.

Найчастіший вид сигналу в реалізаціях ШНМ є дійсним числом. Найчастіше вихід кожного з нейронів обчислюється нелінійною функцією суми входів цього нейрона.

З'єднання між нейронами мають певну вагу, яка змінюється під час навчання. Ця вага впливає на силу сигналу, який передається. Деякі штучні нейрони можуть мати поріг, сигнал нижче якого не сприймається.

Найчастіше штучні нейрони організовуються в шари. Різні шари можуть по різному перетворювати сигнали на своїх входах. Сигнали проходять від першого шару до останнього, інколи по кілька разів проходячи повторно через шари, через які вони уже проходили.

Базовою ідеєю підходу ШНМ було створення системи, яка б розв'язувала задачі таким самим чином, як це робить мозок живих істот. Поступово розвиток ШНМ відхилився від біологічних ідей і зосередився на розвитку певних здібностей ШНМ. Штучні нейронні мережі використовувалися в різних задачах, таких як комп'ютерний зір, розпізнавання мови, машинний переклад тексту, гра в настільні та відеоігри, медичне діагностування, соціально-мережеве фільтрування.

Найбільш поширеним видом архітектури ШНМ вважаються багатошарові мережі [6]. При використанні цієї архітектури нейрони об’єднуються у прошарки з єдиним вектором сигналів входів. Вхідний вектор подається на вхідний шар ШНМ (також відомий як рецептори). Виходами нейромережі є вихідні сигнали з останнього шару (також відомі як ефектори).
Ця архітектура базується на кількох основних принципах:

1. При збільшенні кількості нейронів, збільшення кількості зв’язків та кількості шарів зростають можливості мережі.
2. При додаванні зворотних зв’язків потужність мережі також збільшується, проте виникає динамічна нестійкість мережі.

3. При ускладненні алгоритмів, за якими функціонує мережа (наприклад, при додаванні кількох типів синапсів до одної мережі) можливості мережі теж зростають.

Питання про необхідні і достатні властивості мережі для розв’язування того чи іншого роду задач є цілим напрямом нейрокомп’ютерної науки.

Оскільки проблема розробки ШНС в значній мірі спирається на задачу, котра розв’язується, проблемою є утворення певних загальних рекомендацій. Найчастіше більш-менш придатний для використання варіант виходить на основі емпіричного підбору або на основі інших вдалих рішень, які показали високу продуктивність на схожих задачах.

Деякі типові архітектури нейронних мереж.

Мережі прямого розповсюдження:

- перцептрони;
- мережа зворотного (Back Propagation) розповсюдження;
- мережа зустрічного (Counter Propagation) розповсюдження;
- карта Кохонена;
- згорткові архітектури;

Рекурентні мережі:

- мережа Хопфілда;
- мережа адаптивної резонансної теорії;
- двоспрямована асоціативна пам’ять;
- LSTM мережа;

Функціонування нейромережі, робота з певними задачами, які вона здатна розв'язувати, базується на величинах зв’язків між нейронами, тому, при певній структурі НМ, яка відповідає потрібній задачі, розробник нейромережі повинен знайти певні значення всіх змінних коефіцієнтів мережі, при яких вона працює оптимально.

Етап підбору значень коефіцієнтів мережі називається навчанням НМ. Від якості його виконання залежить те, наскільки якісно ШНМ здатна до вирішення поставлених перед нею задач на етапі експлуатації.

На цьому етапі окрім якості добору ваг важливим фактором є час навчання нейромережі. Як правило, ці два параметри пов’язані зворотною залежністю і їх доводиться вибирати на основі компромісу.

Навчання НМ може відбуватися з вчителем або без нього. У першому випадку мережі висуваються значення як вхідних, так і бажаних вихідних сигналів, і вона за деяким внутрішнім алгоритмом підлаштовує ваги своїх синаптичних зв’язків. У другому випадку виходи НМ формуються самостійно, а ваги змінюються за алгоритмом, що враховує тільки вхідні і похідні від них сигнали.

Існує безліч різних алгоритмів навчання, які, однак, діляться на два великі класи: детерміновані і стохастичні. У першому з них підстроювання ваг являє собою жорстку послідовність дій, у другому - вона виробляється на основі дій, що підлягають деякому випадковому процесу.

Розвиваючи далі питання щодо можливої класифікації НМ, важливо відзначити існування бінарних і аналогових мереж. Перші з них оперують з двійковими сигналами, і вихід кожного нейрона може приймати тільки два значення: логічний нуль ("загальмований" стан) і логічна одиниця ("збуджений" стан). До цього класу мереж належить і розглянутий вище перцептрон, так як виходи його нейронів, що формуються функцією одиничного стрибка, дорівнюють або 0, або 1. В аналогових мережах вихідні значення нейронів здатні приймати безперервні значення, що могло б мати місце після заміни активаційної функції нейронів перцептрону на сигмоїду.

Серед інших класифікацій НМ є класифікація, що ділить нейромережі на синхронні та асинхронні. В синхронних мережах стани нейронів змінюються

по одному нейрону за раз, в другому ж — стан змінюється відразу в усього шару нейронів, або іншої, меншої групи нейронів.

Серед різних архітектур ШНМ одною з найбільш широко використовуваних є багатошарова архітектура, в якій кожен окремо взятий нейрон будь-якого шару з'єднаний з усіма нейронами наступного шару. Такі нейромережі називаються повнозв'язними [6] (англ. Dense).

Для найпростішого випадку, коли мережа є одношаровою, алгоритм навчання є нескладним — він базується на ідеї, що правильні вихідні значення відомі зразу ж і модифікація зв'язків виконується таким чином, щоб помилка на виході мережі зменшувалась. На цьому підході, наприклад, базується алгоритм навчання найпростішого одношарового перцептрону.

Якщо ж в мережі більше одного шару, оптимальні значення нейронів проміжних шарів (і першого), зазвичай, невідомі. Перцептрон з двома або більше шарами вже неможливо навчити по такому найпростішому алгоритму. Одним із варіантів рішення подібної проблеми — це створення наборів сигналів, які відповідні вхідним, для всіх шарів нейромережі. Це є досить трудомісткою операцією і на практиці неможливе.

Також цю проблему можливо вирішити шляхом динамічного підлаштовування коефіцієнтів синапсів, під час якого найбільш слабкі зв'язки змінюються на невелику величину в будь-яку сторону, а зберігаються лише ті зміни, після яких помилка на виході нейромережі зменшилась.

Цей метод, хоч і здається простим, вимагає великої кількості обчислень, а для великої нейромережі — кількість таких обчислень є надзвичайно великою.

На практиці використовується третій варіант — коли сигнали помилки від виходів ШНМ поширюються до її входів (на відміну від сигналів під час звичайного процесу роботи мережі). Цей алгоритм навчання нейромережі відомий як процедура зворотного поширення. Він буде розглянутий далі.

У відповідності до методу визначення помилки, відомого як метод найменших квадратів, що мінімізується цільовою функцією, функція помилки ШНМ може бути визначена як:

$$E(w)=\frac{1}{2}\sum_{j,p}(y_{j,p}^{(N)}-d_{j,p})^2 \tag{1}$$

де $y_{j,p}^{(N)}$ – реальний вихідний стан нейрону j виходного шару N нейронної мережі при подачі на її входи p -го образу; d_{jp} – ідеальний (бажаний) вихідний стан цього нейрону.

Сумування ведеться по всім нейронам вихідного шару і по всім образам, що опрацьовуються мережею. Оптимізація виконується за допомогою методу градієнтного спуску. Іншими словами, це підлаштовування коефіцієнтів наступним чином:

$$\Delta w_{ij}^{(n)}=-\eta\cdot\frac{\partial E}{\partial w_{ij}} \tag{2}$$

В цій формулі w_{ij} позначає ваговий коефіцієнт зв'язку між i -м нейроном шару $n-1$ та j -м нейроном шару n, η - певний коефіцієнт, який регулює швидкість навчання, $0<\eta<1$.

З (2) видно, що:

$$\frac{\partial E}{\partial w_{ij}}=\frac{\partial E}{\partial y_j}\cdot\frac{dy_j}{ds_j}\cdot\frac{\partial s_j}{\partial w_{ij}} \tag{3}$$

В цій формулі y_j , позначає вихід нейрону j , а під s_j – розуміється зважена сума його вхідних сигналів, тобто аргумент активаційної функції. Оскільки множник dy_j/ds_j є похідною цієї функції за її аргументом, з цього випливає, що похідна активаційної функції має бути визначена на всій оси абсцис. Як результат, функція одиничного стрибка (як і будь-яка інша активаційна функція з неоднорідностями) не може використовуватись в таких нейромережах, через що доводиться використовувати гладкі функції —

наприклад гіперболічний тангенс або класичну сигмоїду з експонентою. У випадку гіперболічного тангенса

$$\frac{dy}{ds}=1-s^2 \tag{4}$$

Третя складова $\delta s_j/\delta w_{ij}$, очевидно, дорівнює результату нейрону, що знаходиться в шарі $y_i^{(n-1)}$.

Стосовно першого множника в (3):

$$\frac{\partial E}{\partial y_j}=\sum_k \frac{\partial E}{\partial y_k}\cdot\frac{dy_k}{ds_k}\cdot\frac{\partial s_k}{\partial y_j}=\sum_k \frac{\partial E}{\partial y_k}\cdot\frac{dy_k}{ds_k}\cdot w_{jk}^{(n+1)} \tag{5}$$

Введемо нову змінну:

$$\delta_j^{(n)}=\frac{\partial E}{\partial y_j}\cdot\frac{dy_j}{ds_j} \tag{6}$$

тепер ми маємо рекурсивну формулу, яку можемо використати для отримання значень $\delta_j^{(n)}$ шару n з величин $\delta_k^{(n+1)}$ попереднього $n+1$ шару.

$$\delta_j^{(n)}=\left[\sum_k \delta_k^{(n+1)}\cdot w_{jk}^{(n+1)}\right]\cdot\frac{dy_j}{ds_j} \tag{7}$$

В застосуванні до останнього шару формула наступним чином:

$$\delta_l^{(N)}=(y_l^{(N)}-d_l)\cdot\frac{dy_l}{ds_l} \tag{8}$$

Це дає нам можливість переписати рівняння (2) в іншому форматі:

$$\Delta w_{ij}^{(n)}=-\eta\cdot\delta_j^{(n)}\cdot y_i^{(n-1)} \tag{9}$$

Інколи використовується модифікація цієї функції з урахуванням інерційності для зглаження стрибків на поверхні функції помилки. Для цього в формулу (9) додається значення змін ваги з $t-1$ ітерації:

$$\Delta w_{ij}^{(n)}(t)=-\eta\cdot(\mu\cdot\Delta w_{ij}^{(n)}(t-1)+(1-\mu)\cdot\delta_j^{(n)}\cdot y_i^{(n-1)}) \tag{10}$$

μ – так званий коефіцієнт інерційності, t – номер поточної ітерації.

Виходячи з усього вищесказаного, ми маємо наступну логіку побудови алгоритму навчання нейромережі:

1. Спершу ми маємо подати на вхідний шар ЗНМ один з зразків з тренувальної вибірки, прогнати його через нейромережу в звичайному режимі і отримати значення сигналів. Згадаємо, що:

$$s_j^{(n)} = \sum_{i=0}^{M} y_i^{(n-1)} \cdot w_{ij}^{(n)} \qquad (11)$$

де M – число нейронів у шарі $n-1$ з врахуванням нейрону з постійним вихідним станом $+1$, що задає зміщення; $y_i^{(n-1)} = x_{ij}^{(n)}$ – i-й вхід нейрону j шару n.

$$y_i^{(n)} = f\left(s_j^{(n)}\right), \qquad (12)$$

де f – сигмоїдальна функція

$$y_q^{(0)} = I_q, \qquad (13)$$

де I_q – q-а компонента вхідного спостереження.

2. Далі необхідно отримати $\delta^{(N)}$ для останнього шару за формулою (8). Отримати за одною з формул (9) або (10) зміну ваги $\Delta w^{(N)}$ для шару N.

3. За формулами (7) і (9) отримати $\delta^{(N)}$ і $\Delta w^{(N)}$ для решти шарів, $n = N-1,\ldots 1$.

4. Внести правки до ваги в НМ:

$$w_{ij}^{(n)}(t) = w_{ij}^{(n)}(t-1) + \Delta w_{ij}^{(n)}(t) \qquad (14)$$

5. Якщо помилка ЗНМ більше певного порогу, знову почати з кроку 1. Інакше - завершити виконання алгоритму.

Параліч мережі

Під час тренування ШНМ значення ваги зв'язків можуть досягати досить великих значень. Як наслідок, більшість нейронів буде видавати на виході мережі теж великі значення, і похідна функції активації від них буде дуже незначною. Помилка, яка розповсюджується по нейронній мережі, прямо пропорційна до похідної функції активації, тому процес навчання може майже зупинитися.

З теоретичної точки зору ця проблема особливо не досліджена. Звичайно проблему паралічу мережі уникають за допомоги зменшення розміру кроку, проте цей метод має недоліки, головним з яких є збільшення часу навчання. Також для вирішення цієї проблеми або хоча б відновлення після того, як вона настала, використовують різні евристики, але більшість з них може розглядатися як експериментальні. Фактично, найкращим способом для подолання цієї проблеми є регуляризація, яка буде розглянута пізніше.

Локальні мінімуми та сідлові точки

Як говорилося раніше, алгоритм, який поширює помилку в зворотному напрямку використовує оптимізаційний алгоритм, а саме — варіант градієнтного спуску, тобто здійснює спуск по поверхні помилки в напрямку, протилежному до градієнту функції, модифікуючи ваги ШНМ в напрямку до мінімуму функції. Поверхня функції помилки для глибокої мережі дуже нерівномірна і схожа на гірський ландшафт у високовимірному просторі — вона складається з безлічі складок, долин та пагорбів.

Теоретично ШНМ може опинитись в локальному мінімумі, коли неподалік є більш глибокий мінімум, проте на сьогоднішній день прийнята точка зору, що в просторах високої розмірності набагато більша ймовірність опинитись в сідловій точці [8], вихід з якої за допомогою градієнтного спуску може бути можливим, проте ускладненим. Проте проблема локальних мінімумів теж залишаються відкритою.

Розмір кроку

Для алгоритму зворотного поширення помилки існує доказ збіжності. Головною ідеєю цього доказу є те, що розмір корекції ваг може зменшуватись до нескінченно малих величин. На практиці це є неможливим, як через нескінченність часу навчання в такому випадку, так і через апаратні можливості сучасної обчислювальної техніки. Тому розмір кроку береться кінцевим, і дослідник при виборі значення розміру кроку може спиратись лише

на власний досвід. При замалому розмірі кроку — мережа буде дуже довго сходитись, якщо розмір кроку буде завеликим — існує ризик виникнення паралічу або нестійкості.

Часова нестійкість

Процес навчання мережі має виконуватись без пропусків тої інформації, яка вже вивчена. У доказі збіжності вважається, що ця умова виконується, але на практиці для гарантованого виконання цієї умови потрібно, щоб мережа тренувалася на всіх даних перед корекцією ваг. Зміни ваг мають бути обчисленими на всій тренувальній вибірці, що вимагає додаткових обсягів пам'яті. Цей метод може не працювати в непостійних умовах, якщо ШНМ працює в середовищі, яке постійно змінюється, так що вдруге один і той же вектор може вже не повторитись. В такому випадку процес навчання може і не зійтись.

У цьому сенсі алгоритм зворотного поширення помилки не схожий на біологічні системи.

1.Метод не завжди сходиться. Для поліпшення збіжності доводиться застосовувати велику кількість різних евристичних хитрощів.

2. Процес градієнтного спуску схильний застрявати в численних локальних мінімумах функціоналу Q .

3. Доводиться заздалегідь фіксувати число нейронів прихованого шару H. У той же час, це критичний параметр складності мережі, від якого може істотно залежати якість навчання і швидкість збіжності.

4. При надмірному збільшенні складності мережа схильна до перенавчання.

Перенавчання, або надмірно близька підгонка – занадто точна відповідність нейронної мережі конкретному набору навчальних прикладів, при якому мережа втрачає здатність до узагальнення.

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

Причиною перенавчання є те, що вибір тренувальних даних є випадковим. Поступово з процесом навчання функція помилки зменшується на тренувальних даних, проте у випадку, якщо генералізуюча здатність ШНМ є занадто великою, з деякого порогу нейромережа починає зменшувати помилку тільки на тренувальних даних, через підлаштовування під особливості тренувальної вибірки. Фактично ШНМ просто запам'ятовує тренувальні дані разом з відповідними класами.

Однак при цьому відбувається "підстроювання" не під загальні закономірності ряду, а під особливості його частини. При цьому точність прогнозу зменшується.

Один з варіантів боротьби з перенавчанням мережі – поділ навчальної вибірки на дві множини (навчальна і тестова).

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 ми можемо спостерігати зміну помилки прогнозу на тестовій множині паралельно зі спостереженнями над навчальною множиною.

Якусь кількість кроків помилки прогнозу зменшується на обох множинах. Однак на певному етапі помилка на тренувальній вибірці починає збільшуватись, при цьому помилка на навчальній множині продовжує зменшуватися. Цей момент можна вважати закінченням реального або справжнього навчання. Після нього і починається перенавчання.

Прогнозування на тестовій множині є перевіркою працездатності побудованої моделі. Помилка на тестовій множині може бути помилкою прогнозу, якщо тестова множина максимально наближена до поточного моменту.

Якщо застосовуються функції активації з горизонтальними асимптотами, типу сігмоїдної або гіперболічного тангенсу, то мережа може потрапляти в стан паралічу. Якщо нейрон один раз потрапляє в таку мертву зону, то у нього практично не залишається шансів з неї вибратися. Паралізуватися можуть окремі зв'язки, нейрони або вся мережа в цілому.

Умови зупинки

Є певні метрики, які зазвичай використовуються для визначення того, чи слід зупиняти навчання:

1. Кількість епох, впродовж яких відбувалось навчання (Maximum Epochs Reached). Нейромережа буде зупинена після досягнення певної граничної кількості.

2. Точність на тренувальній вибірці даних (Training Set Accuracy). Це відношення кількості правильно розпізнаних даних тренувальної вибірки до їх загальної кількості. Важливо зазначити, що це точність на даних, які мережа "бачила", тому в більшості випадків ця метрика не може бути надійною сама по собі, проте вона може виступати індикатором явища перенавчання в сукупності з іншими метриками.

3. Точність на даних, на яких мережа не навчалась.(Generalization Set Accuracy) Це відношення кількості правильно розпізнаних даних валідаційної вибірки до їх загальної кількості. Вона розраховується наприкінці кожної епохи.

4. Середньоквадратична помилка на тренувальній вибірці даних (Training Set Mean Squared Error (MSE)). Це сума квадратів помилок (бажаний результат мінус реальний) для кожного набору даних тренувальної вибірки.

5. Середньоквадратична помилка на валідаційній вибірці даних (Generalization Set Mean Squared Error (MSE)). Це сума квадратів помилок (бажаний результат мінус реальний) для кожного набору даних валідаційної вибірки. Найбільш актуальна ця метрика для регресійних задач.

$$MSE = \frac{\sum_{i=0}^{n} (desired\ value\ i - actual\ value\ i)^2}{n}, \qquad (15)$$

де n – кількість прикладів в наборі даних.

1.3 Метрики оцінювання якості класифікації

При оцінюванні моделі машинного навчання дуже важливо визначити, як саме оцінювати результат роботи алгоритму. Існує велика кількість різних метрик і вибір правильної метрики є дуже важливим. В середньому, якщо одна модель виконує задачу значно краща за іншу, то вона буде переважати над іншою по більшості метрик. З іншого боку, якраз переважання по більшості метрик і може бути критерієм того, що модель працює значно краще.

Проте можливі випадки, коли модель показує кращі результати за одною метрикою, та гірші за іншою. Тому при вирішенні будь-якої задачі машинного навчання необхідно визначити, яка метрика є визначальною та яку ціль ми переслідуємо при побудові алгоритму.

Наприклад, при проведенні наукової роботи часто є доречним використовувати таку саму метрику, яку використовували на таких же даних для іншого алгоритму, для того, щоб була можливість порівняти отримані наукові результати “на одному полі”. А при проведенні комерційної розробки алгоритму машинного навчання, необхідно провести глибоке дослідження предметної області, і можливо, навіть у випадку необхідності, розробити свою метрику, яка найкращим чином допомагає формалізувати поставлену проблему. Неправильний вибір метрики може коштувати підприємству

значних ресурсів та часу, витрачених на збір даних, навчання алгоритму, та використання не самого ефективного алгоритму.

Отже, найчастіше використовуються наступні метрики:

1.3.1 Точність

У найпростішому випадку метрикою може бути частка документів за якими класифікатор прийняв правильне рішення.

$$Accuracy = \frac{P}{N} \tag{16}$$

де P - кількість документів за якими класифікатор прийняв правильне рішення, а N - розмір тестової вибірки.

Проте, у цієї метрики є одна особливість яку необхідно враховувати.

Вона надає всім документам однакову вагу, що може бути некоректно в разі якщо розподіл документів в навчальній вибірці сильно зміщений в бік одного або декількох класів. В цьому випадку у класифікатора є більше інформації по цих класах і відповідно в рамках цих класів він буде приймати більш адекватні рішення.

На практиці це призводить до того, що алгоритм машинного навчання має загальну точність, скажімо, 80%, але при цьому в рамках якогось конкретного класу класифікатор працює з рук геть погано, не визначаючи правильно навіть третину документів. Іншим прикладом може слугувати абстрактна ситуація, коли наявні два класи. Перший з них становить 95% від тренувальної вибірки, другий — 5% відповідно. Отже, алгоритм машинного навчання може мати точність 0.95, лише завжди прогнозуючи перший клас. Зрозуміло, що не завжди те, чого хоче від алгоритму його розробник. У випадку ж, коли класів більше двох, і вони розподілені вкрай нерівномірно, майже неможливо по цій метриці оцінити якість роботи алгоритму.

Один вихід з цієї ситуації полягає в тому щоб навчати класифікатор на спеціально підготовленому, збалансованому корпусі документів. Мінус цього

рішення в тому що ви відбираєте у класифікатора інформацію про остостельно частоті документів. Ця інформація при інших рівних може виявитися важливою для прийняття правильного рішення.

Для балансування класів використовується як зменшення кількості прикладів класу, який є переважаючим в даних, так і генерація нових прикладів для класів, кількість прикладів яких недостатня.

1.3.2 Прецизійність та повнота.

Прецизійність (precision) і повнота (recall) є метриками які використовуються при оцінці здебільшого алгоритмів отримання інформації. Іноді вони використовуються самі по собі, іноді в якості базису для похідних метрик, таких як F-міра. Суть точності і повноти дуже проста.

Точність системи в межах класу - це частка документів, які дійсно належать даному класу щодо всіх документів які система віднесла до цього класу.

Повнота системи - це частка знайдених класифікатором документів, які належать класу щодо всіх документів цього класу в тестовій вибірці.

Ці значення легко розрахувати на підставі таблиці контингентності яка складається для кожного класу окремо, або на підставі матриці заплутувань, яка буде розглянута пізніше.

Приклад загального випадку цієї таблиці наведений в таблиці 1.

Таблиця 1 - Приклад таблиці контингентності.

Категорія I		Експертна оцінка	
Оцінка системи		Позитивна оцінка	Негативна оцінка
	Позитивна оцінка	TP	FP
	Негативна оцінка	FN	TN

У таблиці міститься основна інформація, необхідна для оцінки прецензійності та повноти - інформація, скільки разів система прийняла вірне і скільки разів невірне рішення за документами заданого класу. Вона складається з чотирьох головних комірок, кожна з яких містить один можливий варіант класифікування одного спостереження.

А саме:

$TP(true\ positive)$ - істинно-позитивне рішення, класифікатор прийняв рішення про те, що спостереження відноситься до класу, і його точка зору така ж, як і експертна;

$TN(true\ negative)$ - істинно-негативне рішення, класифікатор прийняв рішення про те, що спостереження не відноситься до класу, і його точка зору така ж, як і експертна;

$FP(false\ positive)$ - хибно-позитивне рішення, класифікатор прийняв рішення про те, що спостереження відноситься до класу, проте воно не відноситься;

$FN(false\ negative)$ - хибно-негативне рішення, класифікатор прийняв рішення про те, що спостереження не відноситься до класу, проте воно до нього відноситься;

Тоді, прецизійність і повнота визначаються наступним чином:

Прецизійність - $$Precision = \frac{TP}{TP + FP} \quad (17)$$

Повнота - $$Recall = \frac{TP}{TP + FN} \quad (18)$$

1.3.3 Матриця заплутувань.

На практиці значення точності і повноти набагато зручніше розраховувати з використанням матриці неточностей (confusion matrix). У разі якщо кількість класів відносно невелика (не більше 100-150 класів), цей підхід дозволяє досить наочно представити результати роботи класифікатора.

Матриця неточностей - це матриця розміру $N \times N$, де N - це кількість класів. Стовпці цієї матриці резервуються за експертними рішеннями, а рядки за рішеннями класифікатора. Коли ми класифікуємо документ з тестової вибірки, ми збільшуємо число, що стоїть на перетині рядка класу який повернув класифікатор і стовпця класу до якого дійсно відноситься документ (рис. 1)

	0.91	0.96	0.94	0.75	1.00	0.83	0.85	0.97	1.00	0.86	1.00	0.79	1.00	0.75	1.00	1.00	0.96	0.90	0.81	0.89	0.94	0.98	0.86	0.89	0.94	0.92	0.96
0.80		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
0.95	1	94	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
1.00	2	0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.29	3	0	0	6	0	0	3	2	0	1	0	0	0	0	0	0	1	1	0	0	1	0	1	3	0	2	0
1.00	4	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.50	5	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	1	1
0.92	6	1	0	0	0	0	152	0	0	1	0	0	0	0	0	0	0	1	4	2	3	0	0	0	0	2	0
0.97	7	1	0	1	0	0	0	256	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	2	0
0.33	8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
0.97	9	0	0	0	0	0	0	0	0	69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
0.82	10	0	0	0	0	0	2	0	0	0	18	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0.87	11	0	0	0	0	0	0	0	0	0	0	34	0	4	0	0	0	0	0	0	0	0	0	1	0	0	0
1.00	12	0	0	0	0	0	0	0	0	0	0	0	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.57	13	0	0	0	0	0	0	0	0	0	0	9	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0
0.63	14	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	3	0	0	0	0	0	0	0	0	0
0.50	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	1	0	0	0	0	0	0
0.77	16	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	47	0	1	3	4	0	0	2	0	1	0
0.87	17	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	69	1	2	5	0	0	0	0	0	0
0.97	18	0	0	0	0	1	4	0	0	1	0	0	0	0	0	0	0	0	197	1	0	0	0	0	0	0	0
0.78	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	35	183	13	0	0	2	0	1	0
0.97	20	0	0	0	0	0	10	3	0	1	0	0	0	0	0	0	0	0	0	4	702	0	0	0	0	6	0
0.93	21	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	56	0	2	0	0	0
0.29	22	0	0	1	0	0	2	0	0	6	0	0	0	0	0	0	0	0	1	1	1	0	6	2	0	1	0
0.91	23	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	3	6	0	0	115	0	0	0
1.00	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0
0.93	25	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2	4	5	0	0	0	1	196	0
0.98	26	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	78

Рисунок 1 - Приклад матриці заплутувань.

Маючи таку матрицю точність і повнота для кожного класу розраховується дуже просто. Точність дорівнює відношенню відповідного діагонального елемента матриці до суми всього рядка класу. Повнота - відношенню діагонального елемента матриці до суми всього стовпчика класу.

Прецизійність класифікатора розраховується як арифметичне середнє його точності по всіх класах. Те ж саме з повнотою. Технічно цей підхід називається macro-averaging.

1.3.4. F-метрика

Зрозуміло що чим вище точність і повнота, тим краще. Але в реальному житті максимальна точність і повнота недосяжні одночасно і доводиться шукати якийсь баланс. Тому, хотілося б мати якусь метрику яка об'єднувала б у собі інформацію про точність та повноту нашого алгоритму. Саме такою метрикою є F-метрика.

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

$$F=(\beta^2+1)\frac{Precision*Recall}{\beta^2 Precision+Recall} \quad (19)$$

, де β - приймає значення в діапазоні 0 <β <1 для підвищення ваги прецизійності. При β> 1 пріоритет віддається повноті. При β = 1 отримується збалансована F-міра (також її називають F1):

$$F=2\frac{Precision*Recall}{Precision+Recall} \quad (20)$$

1.3.5 ROC-AUC метрика

ROC-крива (англ. Receiver operating characteristic) — графік, за допомогою котрого можливо оцінити якість класифікації на двох класах (бінарна класифікація). На ньому відображається співвідношення між часткою об'єктів від загальної кількості носіїв ознаки, правильно класифікованих як таких, що мають цю ознаку, (англ. True positive rate, TPR, також відомої як чутливість алгоритму класифікації) до частки об'єктів від загальної кількості об'єктів, що не несуть ознаки, помилково класифікованих як таких, що мають ознаку (англ. false positive rate, FPR, величина 1-FPR називається специфічністю алгоритму класифікації).

Також цю криву називають кривою помилок.

Інтерпретацію ROC дає значення AUC (англ. Area under ROC curve, площа під ROC-кривою) - кількісний параметр, який позначає площу під кривою помилок, яка обмежена віссю частки неправильних позитивних класифікацій.

Зрозуміло, що цей показник прямо пропорціональний до якості класифікації. Якщо він має значення 0.5 — це означає, що класифікатор працює випадковим чином.

Значення менше 0,5 означає, що класифікатор діє з точністю до навпаки: якщо позитивні назвати негативними і навпаки, класифікатор буде працювати краще.

Розширення ROC-кривих на задачі класифікації з більш ніж двома класами завжди було пов'язане з труднощами, тому що кількість ступенів свободи зростає квадратично від кількості класів, і ROC-простір має c*(c-1) вимірювань, де c - кількість класів. Об'єм під ROC-поверхнею (VUS - Volume Under Surface) розглядається як метрика якості класифікаторів для небінарних завдань класифікації, проте ця метрика вважається досить складною для аналізу.

1.3.6 Logloss метрика

Для задачі бінарної класифікації logloss метрика визначається як:

$$logloss = \frac{-1}{l} * \sum_{i=1}^{l} (y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i)) \quad (21)$$

тут $\hat{y}_i$ - це результат алгоритму на i-му об'єкті, y - справжня мітка класу на i-му об'єкті, а l - розмір вибірки.

Ця функція часто використовується і як метрика оцінювання роботи алгоритму, так і як функція втрат алгоритму класифікації.

Інтуїтивно можна трактувати мінімізацію logloss як завдання максимізації точності шляхом штрафу за невірні прогнози. Однак необхідно відзначити, що logloss вкрай сильно штрафує за впевненість класифікатора в невірній відповіді.

В загальному вигляді (для багатокласової класифікації) ця метрика виглядає наступним чином:

$$logloss = \frac{-1}{l} * \sum_{i=1}^{q} \sum_{j=1}^{l} y_{ij} * \log(\hat{y}_{ij}) \quad (22)$$

тут q - число елементів у вибірці, l - число класів, $\widehat{y}_{ij}$ - результат (ймовірність) алгоритму на і-му об'єкті на питання приналежності його до j-го класу, $y_{ij}=1$ якщо і-й об'єкт належить j-му класу, інакше $y_{ij}=0$.

1.4 Загальні підходи до вибору гіперпараметрів алгоритмів

Проблемою багатьох алгоритмів машинного навчання є необхідність підбору гіперпараметрів алгоритму. Вибрані гіперпараметри можуть значно вплинути на фінальний результат роботи алгоритму. В загальному випадку для підбору гіперпараметрів використовуються наступні підходи:

1. Перебирання усіх можливих варіантів з деякого простору гіперпараметрів. Це один з найбільш витратних підходів, який, проте, дає достатньо надійні результати.

Для деяких алгоритмів він неможливий, або майже неможливий. Наприклад, підбір архітектури нейронної мережі таким шляхом буде дуже витратним і вимагатиме багато часу, в той час як для інших алгоритмів, таких як логістична регресія (в якій таким чином може підбиратись значення регуляризації, тощо) він є оптимальним.

2. Перебирання випадкових варіантів з деякого простору гіперпараметрів. Перевагою над першим підходом є можливість задати певну кількість спроб вибору гіперпараметрів, що дозволяє приблизно задати час, необхідний для перебирання. Недоліком — майже гарантована неоптимальність фінальної комбінації гіперпараметрів. Проте, такий підход кращий, ніж нічого.

3. Використання алгоритмів оптимізації для підбору гіперпараметрів. Вибрана метрика класифікації (як якість роботи алгоритму) виступає як значення цільової функції, яку необхідно оптимізувати. Можливе використання великої кількості алгоритмів оптимізації, в залежності від алгоритма машинного навчання, таких як градієнтний спуск та його

модифікації (такі як метод Нестерова), алгоритми рою частинок (хоча вони є сильно затратними для цієї задачі і для більшості алгоритмів машинного навчання не підійдуть), баєсівська оптимізація (яка зараз є популярним підходом для вирішення цієї задачі на практиці).

4. Емпіричний метод. Дослідник має спиратися на своє розуміння даних, алгоритму та досвід для підбору найбільш ефективних гіперпараметрів. Незважаючи на значне збільшення обчислювальних потужностей, цей метод залишається одним із основних методів підбору гіперпараметрів.

1.5 Регуляризація в методах машинного навчання

Регуляризація в машинному навчанні - метод додавання деякої додаткової інформації до умови задачі з метою розв'язати некоректно поставлену задачу або запобігти перенавчанню.

Ця інформація часто має вигляд штрафу за складність моделі. Наприклад, це можуть бути обмеження гладкості результуючої функції або обмеження по нормі векторного простору.

З байєсівської точки зору багато методи регуляризації відповідають додаванню деяких апріорних розподілів на параметри моделі.

Наприклад, універсальні і найбільш розповсюджені штрафи на складність моделі:

1. L_1 регуляризація:

$$L_1 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i |a_i|$$

2. L_2 регуляризація:

$$L_2 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i (a_i)^2$$

де a_i — вектор ваги алгоритму, а λ - додатковий гіперпараметр регуляризації, який має бути визначений дослідним шляхом.

Перша частина формули — задає звичайну квадратичну помилку, яка широко використовується в різних областях науки.

Для нейронних мереж також використовується регуляризація, вона може бути реалізована шляхом додавання до цільової функції суми модулів ваги (L_1 регуляризація) або суми квадратів ваги (L_2 регуляризація).

Також в нейромережах використовуються додаткові методи регуляризації, які використовуються тільки в них:

1. Дропаут (англ. Dropout). Метод регуляризації мережі Dropout, це метод регуляризації для зменшення перенавчання в нейронних мережах, який допомогає запобіганню складним коадаптаціям до тренувальних даних. Це дуже ефективний спосіб регуляризації моделей з нейронними мережами. Термін "dropout" означає одиниці в нейронній мережі, які випадають в процесі застосування методу регуляризації.

Його ідея надзвичайно проста. Давайте для кожного нейрона (крім самого останнього, вихідного шару) встановимо деяку ймовірність p, з якої він буде викинутий з мережі. Алгоритм навчання змінюється таким чином: на кожному новому тренувальному прикладі ми спочатку для кожного нейрона кидаємо визначаємо випадкову величину, і в залежності від результату (чи більше випадкова величина порогу p, чи менше) або використовуємо нейрон як зазвичай, або встановлюємо його вихід завжди строго рівним нулю. Далі все відбувається без змін; нуль на виході призводить до того, що нейрон фактично випадає з обчислень: і пряме обчислення, і зворотне розповсюдження градієнта зупиняються на цьому нейроні і далі не йдуть.

При подальшому використанні мережі після навчання значення виходу кожного нейрону множиться на ймовірність p .

2. Нормалізація даних батча (порції даних) BatchNormalization всередині нейронної мережі (фактично, нормалізується вихідні значення попереднього шару). Нормалізація даних (віднімання від них математичного очікування та ділення на дисперсію), призводить до того, що дані мають математичне очікування 0 та дисперсію 1.

Вважається, що такий підхід діє подібно до дропаута, проте менш ефективно. Також такий підхід сприяє значному пришвидшенню навчання нейромережі, оскільки нейрони всередині мережі, які спираються на виходи інших нейронів при такому підході завжди отримують дані, що належать одному розподілу і коли нейрони, що лежать перед ними, в процесі навчання змінюють розподіл своїх виходів, розподіл величин, які отримують нейрони наступного шару не змінюється.

1.6 Висновки до першого розділу

У даному розділі розглянуті характеристики звуку, які використовуються в алгоритмах машинного навчання для розв'язання практичних задач, пов'язаних з аудіоінформацією. Найбільш часто використовуються спектральні характеристики та мел-кепстральні характеристики звуку. Також розглянуті алгоритми машинного навчання, які можуть використовуватись для розв'язання цих задач. В якості класифікатора вибрана штучна нейронна мережа. Розглянуті метрики оцінювання точності алгоритмів, які зазвичай використовуються, наведена інформація про загальні методи підбору гіперпараметрів алгоритмів та про засоби запобігання перенавчання алгоритмів (засоби регуляризації).

2.ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ ТА ПІДХІД TRANSFER LEARNING

2.1 Згорткові нейронні мережі

Згорткові нейронні мережі (ЗНМ, англ. convolutional neural network, CNN, ConvNet) - це спеціальний вид глибинних штучних нейронних мереж прямого поширення для обробки даних з гратчастої топологією. прикладами можуть служити тимчасові ряди, які можна розглядати як одновимірну сітку прикладів, які обираються через регулярні проміжки часу, а також зображення, що розглядаються як двовимірна сітка пікселів.

Згорткові мережі досягли колосального успіху в практичних застосуваннях, особливо успішно застосовуються до аналізу візуальних зображень, а також до задач обробки природньої мови та рекомендаційних систем. Своєю назвою вони зобов'язані використанню математичної операції згортки. Згортка - це особливий вид лінійної операції. Згорткові мережі - це просто нейронні мережі, в яких замість спільної операції множення на матрицю, як мінімум в одному шарі, використовується згортка.

Також ЗНМ можуть бути визначені як різновид багатошарових перцептронів, розроблений для мінімального використання попередньої обробки.[7]

Цікавою особливістю цього виду нейронних мереж є те, що вони інваріантні відносно зсуву (англ. shift invariant), або просторово інваріантні штучні нейронні мережі (англ. space invariant artificial neural networks, SIANN). Це є наслідком специфіки їхньої архітектури спільних ваг та характеристик інваріантності відносно паралельного перенесення.[8][9]

Мотивацією для створення згорткових мереж була організація зорової кори тварин (та людини)[10].

Зорова кора - розташована позаду, в потиличній частці головного мозку.

Вона ділиться на кілька частин, які зазвичай називаються зоровою зоною V1 (visual area one), яку також називають стріарною корою або первинною

зорової корою, зоровою зоною V2 (visual area two), зоровою зоною V3 і т. д., до V6 і V7[11].

Зони відрізняються один від одного фізіологією, архітектурою та відособленою позицією в корі, і дослідники не сумніваються, що вони розрізняються і за своїми функціями. Хоча функціональна спеціалізація зон поки що до кінця не ясна, зрозуміло, що функції зон зорової кори стають поступово все більше і більше загальними при переході від початкових до останніх зон.

1. В зоні V1 виділяються локальні ознаки невеликих ділянок отриманого з сітківки зображення;

2. V2 продовжує виділяти локальні ознаки, частково узагальнюючи їх і додаючи бінокулярний зір (тобто стереоефект від двох очей);

3. В зоні V3 розпізнається колір, текстури об'єктів, з'являються перші результати їх сегментації і угруповання;

4. Зона V4 вже починає розпізнавати геометричні фігури і обриси об'єктів, поки нескладних; крім того, саме тут найбільш сильна модуляція за допомогою нашої уваги: активація нейронів в V4 не рівномірна по всьому полю зору, а сильно залежить від того, на що ми свідомо чи несвідомо звертаємо увагу;

5. Зона V5 в основному займається розпізнаванням рухів, намагаючись зрозуміти, куди і з якою швидкістю пересуваються в зоні видимості ті самі об'єкти, обриси яких виділилися в зоні V4;

6. В зоні V6 узагальнюються дані про всю зображенні, вона реагує на зміни по всьому полю зору (wide-field stimulation) і зміни в зображенні внаслідок того, що пересувається сама людина;

7. Іноді також виділяють зону V7, де відбувається розпізнавання складних об'єктів, зокрема людських облич.

Така функціональна спеціалізація — це уявлення про зорову кору, яке добре відповідає тому, що ми зазвичай бачимо в глибоких нейронних мережах: більш високі рівні потрібні для того, щоб виділяти найбільш загальні ознаки зображення, відповідні абстрактним властивостям входу, а на нижніх рівнях розпізнаються ознаки більш конкретні. У згорткових мережах, зрозуміло, ефект буде той же самий. Але є й інші цікаві властивості архітектури зорової кори, які знайшли відображення в машинному навчанні.

Крім зон V1-V7, виділяють і інші області, а серед перерахованих ієрархія не така вже однозначна: є маса прямих зв'язків, коли, наприклад, нейрони з зони V1 подають сигнал на вхід не тільки в зону V2, але і безпосередньо в зону V5; це теж знайде відображення в згорткових архітектурах.

Крім того, на відміну від більшості архітектур штучних нейронних мереж, між нейронами в мозку завжди присутній дуже сильний зворотний зв'язок від більш високих рівнів до нижчих. Наприклад, сучасні дослідження припускають, що увага, яку зароджується в зоні V4, потім переходить назад до V2 і V1.

З усієї зорової кори найбільш добре вивчена зона V1, первинна зорова кора: вона ближче всього до входів, досить просто влаштована, і там простіше зрозуміти, за що «відповідають» окремі нейрони і як вони один з одним взаємодіють. Виявилося, що зона V1 сама складається з шести рівнів нейронів: вхідні сигнали приходять з латерального колінчастого тіла в четвертий рівень, подальші сигнали виходять з рівнів 2 і 3, а зворотній зв'язок - з шостого.

Нейрони в зоні V1 розташовуються не випадковим чином: зона V1 містить повну «карту» полів зору обох очей, тобто близькі ділянки сітківки обробляються близькими нейронами в V1.

Цікаво, що при цьому локальна структура переноситься дуже точно, а от на глобальному рівні є серйозні спотворення: по-перше, центральна ділянка поля зору сильно збільшена (половина всіх нейронів відповідає за 2% поля

зору), по-друге, є геометричні спотворення, схожі на використання полярних координат: концентричні кола і радіальні лінії на зображенні перетворюються в вертикальні та горизонтальні лінії в V1.

Це дозволяє зберігати інваріантність зображення при зміні нашої позиції і точки зору. Кожен нейрон в V1 працює з дуже маленьким ділянкою зображення (він називається рецептивних полем, receptive field), і між ними зберігаються просторові взаємозв'язки.

Цікаво розібратись, на що, власне, реагують нейрони в V1. Крім розкладання, відбитого в «карті», різні нейрони в V1 розпізнають:

1. орієнтацію, тобто нейрон реагує, наприклад, на те, що освітленість вздовж діагоналі зображення висока, а по двох інших кутах низька; або на те, що освітленість на нижній частині зображення вище, ніж на верхній; таким чином нейрони розпізнають кордону зображень (edge detection);

2. просторову частоту, тобто те, наскільки часто змінюється освітленість в межах рецептивного поля нейрона;

3. напрямок: на відміну від більшості мереж, людині цікава не тільки і не стільки статична картинка, скільки рухи, що на ній відбуваються; нейрони вміють «запам'ятовувати»і порівнювати попередні входи з наступними, розпізнаючи таким чином напрямок не тільки в просторі, але і в часі;

4. відмінність між очима: в V1 у кожного нейрона два рецептивних поля, для кожного ока, і хоча більшість нейронів в основному «присвячені» одному конкретному оку (тобто не реагують на стимули з іншого), деякі розпізнають якраз відмінності між тим, що бачать ліве і праве око;

5. колір, нейрони зазвичай розпізнають один з трьох основних напрямків: червоний - зелений, синій - жовтий і чорний -білий.

Назва ЗНМ походить від терміну «згортка». Для розуміння цього терміну необхідно відступити на крок назад. Наріжним каменем усіх нейронних мереж є афінні перетворення. У кожному шарі повнозв'язної нейронної мережі в

повторюється одна і та ж операція: на вхід подається вектор, який множиться на матрицю ваги, а до результату додається вектор вільних членів; тільки після цього до результату застосовується якась нелінійна функція активації. Незалежно від типу даних ці афінні перетворення в нейромережах залишаються незмінними.

Однак багато типів даних мають свою власну внутрішню структуру, яка добре відома нам заздалегідь.

Одним з прикладів такої структури в цій можуть виступати зображення, які зазвичай представляють як масив векторів чисел: якщо зображення чорно-біле, то це просто масив інтенсивностей, а якщо кольорове, то масив векторів з трьох чисел, що позначають інтенсивності трьох або більше основних кольорів (наприклад, червоного, зеленого і синього в стандартному RGB, і т.д.). Якщо ж узагальнити таку внутрішню структуру, отримаємо наступне:

1. вихідні дані представляють собою багатовимірний масив («тензор»);
2. серед розмірностей цього масиву є одна або більше осей, порядок уздовж яких грає важливу роль; наприклад, це може бути розташування пікселів в зображенні, часова шкала для музичного твору, порядок слів або символів у тексті;
3. інші осі позначають «канали», що описують властивості кожного елемента за попередньою підмножиною осей; наприклад, три компонента для зображень, два компонента (правий і лівий) для стереозвуку і т. д.

Коли ми навчаємо повнозв'язні нейронні мережі, це додаткове знання про структуру даних ніяк не використовується.

Але це знання в задачі роботи з зображеннями може грати визначальну роль - взаємне розташування пікселів в зображенні важливо для вирішення багатьох задач.

Якщо ми не будемо робити вигляд, що нічого не знаємо про структуру входів (розділенні кольорових каналів зображення, порядку нуклеотидів в ДНК

і т. д.), а станемо її безпосередньо використовувати, це може істотно допомогти нам у навчанні мережі.

Основна ідея згорткової мережі полягає в тому, що обробка ділянки зображення дуже часто має відбуватися незалежно від конкретного розташування цієї ділянки. Це локальна задача, яку можна вирішити локальними засобами. Звичайно, взаємне розташування об'єктів грає важливу роль, але спочатку їх потрібно в будь-якому випадку розпізнати, і це распознавання - локально і незалежно від конкретного положення ділянки з об'єктом всередині великої картинки.

«Згортка» - це всього-на-всього лінійне перетворення вхідних даних особливого виду. Якщо xl - карта ознак в шарі під номером l, то результат двовимірної згортки з ядром розміру $2d+1$ і матрицею ваг W розміру $(2d+1)\times(2d+1)$ на наступному шарі буде таким:

$$y^{l}_{i,j}=\sum_{-d\le a,b<:=d} W_{a,b}\, x^{l}_{i+a,j+b} \qquad (16)$$

де y^{l}_{ij} - результат згортки на рівні l, x^{l}_{ij} - її вхід, тобто вихіл попереднього шару. Інакше кажучи, щоб отримати компоненту (i, j) наступного рівня, ми застосовуємо лінійне перетворення до квадратному вікна попереднього рівня, тобто скалярно множимо пікселі з вікна на вектор згортки.

Це перетворення має ті властивості, про які ми говорили вище:

1. згортка зберігає структуру входу (порядок в одновимірному випадку, взаємне розташування пікселів в двовимірному і т. д.), так як застосовується до кожної ділянки вхідних даних окремо;

2. операція згортки має властивість розрідженості, так як значення кожного нейрона чергового шару залежить тільки від невеликої частки вхідних нейронів (а, наприклад, в повно-нейронної мережі кожен нейрон залежав би від усіх нейронів попереднього шару);

3. згортка багаторазово перевикористовує одні і ті ж ваги, так як вони повторно застосовуються до різних ділянок входу.

Головною особливістю згорткових нейромереж є те, що вони не потребують попередньої обробки даних (особливо на фоні більш класичних методів машинного навчання)

Це означає, що мережа самостійно навчається фільтрам, які в більш традиційних, класичних алгоритмах комп'юторного зору розроблялися за допомоги експертів. Такі мережі незалежні від апріорних знань про предметну область, що є важливим плюсом.

Згорткова нейромережа має шари входу та виходу та приховані шари між ними, чим нагадує перцептрон. Різниця полягає в тому, які саме шари використовуються в нейронній мережі. Приховані шари ЗНМ зазвичай складаються зі згорткових шарів (convolutional layers), агрегувальних шарів (pooling layers), повноз'єднаних шарів (dense layers, чи full connection layers) та шарів нормалізації.

Згорткові шари за допомогою згортки імітують реакцію біологічних нейронів на візуальний сигнал. Кожен згортковий нейрон може обробляти інформацію лише в межах свого рецептивного поля.

Повноз'єднані нейронні мережі прямого поширення можуть бути використаними як для вивчення ознак, так і для розв'язання задачі класифікації, проте використання цієї архітектури до зображень є непрактичним.

При використанні багатошарового перцептрону необхідно значно більше число нейронів через дуже великі розміри входу, пов'язані з зображеннями, де кожен піксель є відповідною змінною. Операція згортки дає змогу розв'язати цю задачу, оскільки вона зменшує кількість вільних параметрів, дозволяючи мережі бути глибшою за меншої кількості параметрів. Це розв'язує проблему

зникання або вибуху градієнтів у тренуванні традиційних багатошарових нейронних мереж з багатьма шарами за допомогою зворотного поширення[12].

Архітектура ЗНМ:

ЗНМ складаються з сукупності різних шарів, які трансформують вміст входу до виходу за допомогою певної функції, що є диференційованою.

Найчастіше використовуються кілька видів шарів. Найчастіше це одні з наступних шарів:

Згортковий шар:

Згортковий шар (англ. convolutional layer) є найбільш важливим (і основним) шаром згорткової нейромережі. Його параметрами є сукупність фільтрів, кожен з яких має рецептивне поле невеликого розміру, яке простирається продовж всього обсягу вхідного вмісту. Під час етапу прямого проходу кожен фільтр обчислює результат скалярного добутку входу та значень фільтру і формує n - вимірну (де n - розмірність даних, з якими працює нейромережа, наприклад для зображень це буде 2) матрицю активації фільтру.

Внаслідок цього мережа вивчає, які фільтри реагують на окремо взяті ознаки, що розташовані певним чином на вхідних даних.

Сукупність усіх матриць активації для всіх фільтрів упродовж всього обсягу даних і є результатом виходу згорткового шару. Також є справедливою така точка зору, коли кожен блок в цій сукупності може розглядатись як результат роботи нейрону, який має невелике рецептивне поле на вході та однакові параметри з іншими нейронами тої ж карти активації.

Локальна з'єднаність:

Для роботи з багаторозмірними вхідними даними, наприклад такими як зображення, неефективно поєднувати нейрони наступного шару з усіма нейронами попередніх шарів (як це робиться в багатошаровому перцептроні),

бо така архітектура нейромережі не зможе використовувати факт просторової організації даних.

Саме тому ЗНМ використовують схему локальної з'єднаності між нейронами шарів, які знаходяться поряд — в такій схемі кожен нейрон має доступ лише до невеликої частини вхідного вмісту. Розмір цього доступу є параметром алгоритму, який часто називають рецептивним полем нейрону.

З'єднання є локальними в просторі (бо вони розташовані відносно ширини та висоти), але завжди розповсюджуються впродовж усієї глибини вхідного вмісту. Такий підхід є найбільш оптимальним з точки зору реагування навчених фільтрів.

Просторова організація:

Розмір виходу згорткового шару залежить від наступних параметрів алгоритму — глибини вмісту виходу, розміру кроку та наявності (та розміру у випадку наявності) нульового вирівнювання.

1. Глибина вмісту виходу задає кількість тих нейронів шару, що поєднані до однієї і тої самої області вхідних даних. Ці нейрони в процесі навчання тренуються реагувати на різні ознаки входу. Іншими словами, у випадку, якщо на вхід даних поступає необроблене зображення, то різні нейрони впродовж глибини вмісту виходу будуть тренуватись реагувати на різні ознаки, такі як лінії, контури, плями кольору, тощо.

2. Розмір кроку задає те, як щільно розташовані фільтри згорткового шару нейромережі. Наприклад, для значення 5 фільтри розташовані через п'ять пікселів за раз. Коли ж використовується розмір кроку 3, то фільтри розташовані через три пікселі, що веде до більшого перекриття рецептивних полів (і більшої розмірності виходу згорткового шару). Але при цьому отримується більше інформації, ніж в випадку з 5 пікселями.[12]

3. Нульове вирівнювання задає, чи доповнюється вхід нулями по краям вмісту, і розмір цього доповнення, якщо таке вирівнювання використовується.

Це дає можливість регуляції просторової розмірності вмісту виходу. Інколи це є важливим і зручним при розробці алгоритму.

Розмірність вмісту виходу залежить від розміру вхідного вмісту, розміру фільтрів, розміру кроку, з яким вони розташовані та від величини (та наявності) нульового вирівнювання по краям даних.

Спільне використання параметрів:

Спільне використання параметрів фільтрів використовується в згорткових межах з причини необхідності зменшення кількості параметрів, які мають бути навченими. Головна ідея такого підходу полягає в припущенні, що характеристика зображення, яка є важливою для задачі нейромережі є важливою в будь-якому положенні та будь-якій позиції. Інакше кажучи, для кожної підмножини нейронів, розташованої в зрізі за глибиною, ми використовуємо для нейронів одинакові параметри (ваги та зміщення).

Через те, що всі нейрони в такій підмножині використовують спільні параметри, то результат стандартного режиму роботи (в якому дані просуваються від входу до виходу) в цій підмножині згорткового шару може бути отримане як згортка ваг (та зміщень) нейронів та вхідних даних.

Саме тому така підмножина нейронів часто називається фільтром (точніше, не стільки підмножина нейронів, скільки набір їх ваг), або ядром згортки. Після застосування згортки отримується те, що називається матрицею активації. Сукупність таких матриць активації, поєднана в матрицю більшої розмірності і отримується на вмісті виходу.

Саме через це нейромережі архітектури ЗНМ називаються інваріантними відносно зсуву — бо ознака розпізнається (якщо ЗНМ її вміє розпізнавати), незалежно від її положення.

Незважаючи на те, що зазвичай спільне використання параметрів є виправданим, інколи воно не має практичної користі. Головною причиною такої проблеми може бути випадок, коли зображення вже мають уніфіковані

особливості структури, для якої ознаки в різних просторових положеннях можуть бути тільки різними. Зрозуміло, що в такому випадку, можливість розпізнавати ознаки в будь-яких положеннях є неактуальною та особливої вигоди не несе.

Прикладом такої ситуації може бути, наприклад, велика кількість фотографій автомобілів з відцентрованими номерними знаками — ми можемо очікувати, що розпізнавання номерних знаків буде виконуватись тільки в одній області зображення.

У випадку подібних даних зазвичай використовується спрощена вхема спільного використання параметрів. Такий шар звичайно називають локально поєднаним.

Агрегувальний шар:

Іншим важливим шаром, характерним для ЗНМ є агрегувальний шар. Агрегування (англ. pooling) — є видом нелінійного пониження дискретизації. Існує певна кількість варіантів реалізації агрегування. Часто використовується максимізаційний пулінг та усереднений пулінг. В першому випадку вхідні дані розбиваються на набір прямокутників, які не перекриваються та далі передається лише максимальне значення регіону. В другому випадку далі передається лише усереднене значення. Головним аргументом такого підходу є те, що точне положення певної характеристики зображення менш важливе, ніж приблизне позіціонування цієї характеристики відносно інших розпізнаних характеристик зображення. За допомогою агрегувального шару виконується поступове зниження розмірності даних для скорочення кількості вільних параметрів і зменшення об'єму обчислень у мережі.

В архітектурі ЗНМ агрегувальний шар зазвичай вставляється між певною послідовною кількістю згорткових шарів. Це додає додатковий варіант просторової інваріантності відносно паралельного перенесення.

Агрегувальний шар працює незалежно для кожного фільтру (кожної сукупності нейронів, які мають спільні ваги) і знижує розмірність для такого фільтру. Найчастіше використовують агрегувальний шар для зниження дискретизації в два рази, за шириною і за висотою. Для цього застосовують фільтри розміру 2×2 з кроком 2. Це відкидає три чверті активацій, лишаючи лише чверть найбільш сильних у випадку максимізаційного пулінгу. У випадку ж усереднюючого, фактично створюються нові матриці активацій замість оригінальних з усередненими значеннями.

Окрім перечислених двох найбільш поширених функцій усереднення, в агрегувальних узлах можуть використовуватись і інші варіанти. Прикладом такого варіанту може бути L2-нормове агрегування(англ. L2-norm pooling), та інші. Впродовж розвитку згорткових нейромереж, усереднювальний пулінг використовувався досить часто, проте останнім часом максимізаційний пулінг почав використовуватись значно частіше.[14] З іншого боку, усереднений пулінг, точніше його модифікація — усереднений глобальний пулінг — використовується для дослідження карт активації класів ЗНМ.

Наразі, через занадто агресивне зниження розмірності даних в агрегувальному шарі, починають частіше використовуватись менші фільтри[15]. Також інколи відмовляються від агрегувального шару повністю. [16]

Іншим варіантом пулінгу є агрегування регіонів інтересу, (англ. Region of Interest pooling, або англ. RoI pooling). Це модифікація максимізаційного пулінгу, яка відрізняється тим, що розмірність виходу в ній зафіксовано, а прямокутник входу — налаштовується гіперпараметром. [17]

В згорткових нейромережах, які спеціалізуються на детектуванні зображень, агрегування є одною з основних складових.(англ. Fast R-CNN).[18]

Шар зрізаних лінійних вузлів (ReLU):

ReLU (англ. Rectified Linear Units) — тип шару з нейронами з функцією “випрямовувач”(англ. “rectifier”). Цей шар використовує наступну ненасичену функцію:

$$f(x)=max(0,x) \tag{16}$$

Ця функція є більш популярною, ніж інші, в першу чергу, через те, що нейромережа з цією функцією активації тренується значно швидше[19], без великих втрат точності розпізнавання.

Повноз'єднаний шар:

В кінці згорткової нейромережі, після згорткових шарів та шарів агрегації, фінальне рішення з класифікації приймається за допомогою повноз'єднаних шарів (англ. fully connected layers, або dense layers). В такому шарі кожен нейрон поєднаний до всіх активацій минулих шарів, подібно до того, як це робиться в перцептроні. І як в перцептроні, результат роботи цього шару обчислюється за допомогою матричного множення, після якого виконується зсув.

Шар втрат:

Зазвичай це фінальний шар класифікації. Він визначає, як саме має штрафуватись помилка нейромережі (відхилення між розміченими класами та класами, спрогнозованими нейромережею). В ньому можуть використовуватись різні функції втрат.

Найчастіше використовуються наступні функції:

1. Softmax - функція нормованих експоненційних втрат. Вона використовується для прогнозування одного класу з певної кількості взаємно виключених класів.
2. Сигмоїдні перехрестно-ентропійні втрати використовуються для прогнозування певної кількості незалежних значень ймовірності в проміжку.
3. Звичайні евклідові втрати використовуються для регресійних задач.

2.2 Підхід transfer learning

Термін transfer learning («перенесення навчання») відноситься до ситуації, коли щось навчене в одній ситуації (наприклад, розподіл P_1) використовується для поліпшення узагальненість в іншій ситуації (наприклад, при розподілі P_2).[21]

При перенесенні навчання передбачається, що багато факторів, що пояснюють варіативність P_1 , відносяться і до змін, які належить вловити для навчання P_2 . Зазвичай це інтерпретується в контексті навчання з учителем, коли вхід один і той же, а природа міток може бути різною.

Наприклад, ми можемо навчитися чомусь, що належить до одному набору зорових категорій, скажімо собакам і кішкам, а потім перейти до другим категоріям, скажімо осам і мурашкам. Якщо в першому випадку даних (обраних з розподілу P_1) набагато більше, то, можливо, має сенс навчити представлення, корисні для швидкого узагальнення при наявності лише невеликої вибірки з P_2 . У багатьох зорових категорій є загальні особливості: низькорівневі характеристики - межі і форми, ефекти від застосування геометричних перетворень, змін освітлення і т. д.

У загальному випадку перенесення навчання, багатозадачне навчання і адаптацію домену можна реалізувати шляхом навчання представлення, якщо існують ознаки, корисні в різних ситуаціях або задачах, які відповідають пояснює факторам, яке трапляється в кількох ситуаціях.

Іноді загальної для різних завдань є семантика виходу, а не входу. Наприклад, система розпізнавання мови повинна породжувати правильні речення в вихідному шарі, але попередні шари можуть розпізнавати дуже різні варіанти одних і тих же фонем або субфонемних огласовок, що залежать від мовця.

У таких випадках розумніше розділяти верхні шари нейронної мережі і виконувати залежну від завдання попередню обробку.

Іншим спорідненим терміном є задача адаптації домену. В ній задача (оптимальне відображення входу на вихід) залишається тією ж самою у всіх ситуаціях, але розподілу входу розрізняються.

Наприклад, розглянемо задачу аналізу емоційного забарвлення, коли потрібно визначити, висловлює чи коментар позитивні або негативні емоції. Ця задача відома як аналіз тональності (англ. sentiment analysis) та має досить широке практичне застосування. Коментарі в інтернеті можуть мати різне походження. Проблема адаптації домену виникає, коли предиктор емоційного забарвлення, навчений на відгуках користувачів книг, відео і музики, потім використовується для аналізу коментарів, що відносяться, скажімо, до споживчій електроніці: телевізорам і смартфонам.

Можна уявити собі, що існує базова функція, яка розпізнає емоційне забарвлення - позитивне, нейтральне або негативне. Але, звичайно, словниковий склад і стиль можуть залежати від предметної області - домена, що ускладнює узагальнення з одного домену на інший.

Споріднена задача до transfer learning задача - дрейф концепцій (concept drift), також відома, як (dataset shift), її можна розглядати як форму перенесення навчання в силу поступових змін розподілу даних з часом. Наприклад, вона виникає, коли в користувача рекомендаційної системи змінюються вподобання. І дрейф концепцій, і перенесення навчання можна вважати особливими видами багатозадачного навчання. Хоча термін «багатозадачне навчання» частіше застосовується до завдань навчання з учителем, більш загальне поняття перенесення навчання може бути застосовано також до навчання без вчителя і до навчання з підкріпленням.

У всіх цих випадках мета - скористатися знаннями, набутими в першій конфігурації, для добування інформації, яка може стати в нагоді для навчання або навіть прямого використання в другій конфігурації. Ключова ідея навчанню представлень полягає в тому, що одне і те саме представлення може

бути корисним в обох конфігураціях. А це дозволяє збагатити представлення, користуючись навчальними даними, доступними для обох задач.

Два крайніх випадки перенесення навчання - навчання на одному прикладі (one-shot learning) і навчання без прикладів (zero-shot learning). У першому випадку для перенесення задачі є тільки один розмічений приклад, а в другому - ні одного.

Найчастіші застосування підходу перенесення навчання, це:

1. Обробка природної мови. В першу чергу це стосується представлень слів (word embeddings). Спеціалізовані представлення слів — набагато більш інформативній шлях представлення слів, ніж "мішок слів" (bag-of-words). Вони широко використовуються та існує багато їх варіантів (word2vec, glove, fasttext, тощо). Головним недоліком таких представлень є існування слів, представлень для яких навчений корпус не має, оскільки ці слова були відсутні в тренувальному наборі даних.

2. Алгоритми комп'ютерного зору. Останні архітектури нейронних мереж, достатньо потужні для роботи з наборами даних великого розміру (наприклад, такими, як ImageNet) погано масштабуються до невеликих наборів даних через проблему перенавчання. Іншими словами, вони просто запам'ятовують всі дані тренувального набору, погано працюючи на тестовому наборі даних.

При використанні перенесення навчання в нейромережах, є багато факторів, які впливають на вибір правильної стратегії, проте найбільш важливими є два фактори:

1. Розмір набору даних, з яким має навчитися працювати нейромережа. Він може бути як відносно невеликим, так і велетенським.

2. Схожість цього набору даних до початкового набору даних. Вони можуть бути як схожими, так і зовсім несхожими.

З урахуванням того факта, що ЗНМ виокремлює більш загальні характеристики на ранніх шарах і більш важливі для конкретного набору даних на наступних шарах, можливі чотири основних ситуації:

1. Новий набір даних менше за розміром і аналогічний за змістом початкового набору даних. Якщо обсяг даних невеликий, то немає сенсу проводити тонке налаштування ЗНМ через донавчання. Оскільки дані схожі з початковими, можна припускати, що характеристики зображення в ЗНМ будуть доречні і для цього набору даних. Тому рішенням, яке часто застосовується, є навчання лінійного класифікатора на значеннях останніх шарів нейромережі. Традиційно використовується SVM (Support Vector Machine, машина опорних векторів).

2. Новий набір даних великий і аналогічний за змістом до початкового набору даних. Фактично сценарій схожий до першого, проте можливо як використовувати рішення проблеми з використанням лінійного класифікатору, так і донавчати мережу на нових даних.

3. Новий набір даних менший за кількістю даних і значно несхожий за змістом до початкового набору даних. Тому що кількість даних невелика, має вистачити тільки нескладного класифікатора, найчастіше лінійного. З урахуванням того, що дані помітно відрізняються, логічніше навчати класифікатор на даних, взятих з низу мережі, де містяться більш універсальні дані, замість використання більш специфічних шарів ЗНМ.

4. Новий набір даних великий і дані, що в ньому містяться, значно відрізняються від того набору даних, з яким мережа працювала раніше. Тому що набір даних дуже великого розміру, теоретично можливо навчити ЗНМ аналогічної архітектури з нуля. Але на практичних задачах часто навіть в такому випадку вигідніше ініціалізувати нейромережу параметрами вже тренованої моделі. В цьому випадку даних має вистачити для піднастройки всієї нейромережі шляхом перенавчання.

В цій дипломній роботі розглядається використання останнього сценарію, оскільки кількість даних в наборі даних достатня (в UrbanSound[21] Dataset 8732 аудіосигналів, що відносяться до 10 різних класів) та вони суттєво відрізняються від набору даних, на яких попередньо навчалася мережа (ImageNet[23] містить фотографії більш ніж 2000 різних класів, хоча на щорічних змаганнях з розпізнавання використовується лише 1000 класів).

2.3 Висновки до другого розділу

У даному розділі детально розглянута структура згорткових нейронних мереж та наведені їх основні особливості. Також в розділі коротко викладені біологічні особливості кори головного мозку тварин, які підштовхнули до розвитку цей тип нейромереж. Також в розділі наведена інформація про підхід перенесення навчання в цілому. Окрім цього у цьому розділі викладена логіка вибору методики перенесення навчання в залежності від даних, яка найчастіше використовується. Окреслено основні особливості набору даних, який досліджується в даній дипломній роботі та наведена загальна інформація про вибрану методику перенесення навчання.

3. РОЗРОБКА АЛГОРИТМУ КЛАСИФІКАЦІЇ ЗВУКУ

3.1 Розробка алгоритму класифікації звуку

Запропонований підхід полягає в використанні попередньо навченої на класифікації зображень мережі для класифікації звуку шляхом перенавчання останніх (і лише останніх шарів мережі) на наборі спектрограм, згенерованих для аудіосигналів (приклад спектрограми на рис. 1).

Цей підхід не є новаторським [24], і вперше був запропонований в 2014 році для класифікації музики за жанром. Проте був використаний інший набір даних (що не дає можливості провести повноцінне порівняння отриманих результатів) та інша, набагато простіша, структура нейронної мережі (а саме ЗНМ архітектури AlexNet). Також використовувався інший процес попередньої обробки даних — автори статті використовували гармонічно-ударне розділення, після якого для кожної з компонент генерувалась своя спектрограма.

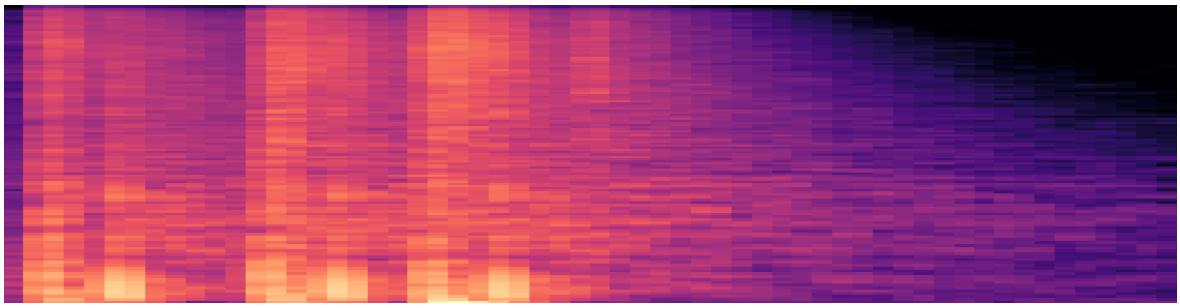


Рисунок 2 - Приклад згенерованої спектрограми.

Нижні шари будь-якої згорткової нейронної мережі, навченої на розв’язання задачі класифікації (або будь-якої іншої задачі роботи з зображеннями, наприклад, локалізації об’єктів), зображень розпізнають низькорівневі характеристики зображення, такі як кути або лінії. Ці характеристики в цілому є притаманними всім зображенням.

Допоки ми не маємо дуже специфічну проблему або набір даних, наша нейронна мережа має починати з розпізнавання таких самих низькорівневих характеристик зображення. Замість навчання всієї нейронної мережі з нуля, ми можемо використати ваги з вже навченої іншої моделі для нижніх шарів, та зосередитися на навчання більш важливих шарів (тих, які знаходяться вище).

Для комп'ютерних експериментів використана нейромережа архітектури VGG16 [25] (рис. 2), навчена на наборі даних ImageNet[23] без останнього шару. Архітектура цієї мережі була вперше запропонована в 2014 році.

ImageNet — проект зі створення і супроводу масивної бази даних анотованих зображень, призначений для відпрацювання і тестування методів розпізнавання образів і машинного зору. Станом на 2016 рік в базу даних було записано близько десяти мільйонів URL з зображеннями, які пройшли ручну анотацію для ImageNet, в анотаціях перелічувались об'єкти, що потрапили на зображення, і прямокутники з їх координатами.

З 2010 року ведеться проект ILSVRC (ImageNet Large Scale Visual Recognition Challenge - змагання по широкомасштабному розпізнаванню образів в ImageNet), в рамках якого різні програмні продукти щорічно змагаються в класифікації і розпізнаванні об'єктів і сцен в базі даних ImageNet.

Варто зазначити, що через особливість просторової інваріантності ЗНМ та спільного використання параметрів, можливо донавчати згорткову нейронну мережу на зображеннях іншого формату, ніж зображення початкового набору даних. Іншими словами, без зайвих ускладнень можливий перенос навчання на зображення іншого формату. Наприклад формат зображення набору даних,на яких попередньо була навчена мережа, зображеннях ImageNet — дорівнює 224x224 пікселя. Така особливість нейронних мереж є дуже важливою, що значною мірою впливає на можливість та зручність використання такого підходу на практиці (на реальних задачах).

Ця попередньо навчена модель дає лише 7.5 відсотки топ-5 похибки на валідаційному наборі даних ILSVRC-2012-val та 7.4 відсотки похибки на тестовому наборі даних ILSVRC-2012-test. Це є достатньо якісним результатом, особливо, якщо взяти до уваги складність задачі, що розв'язується.

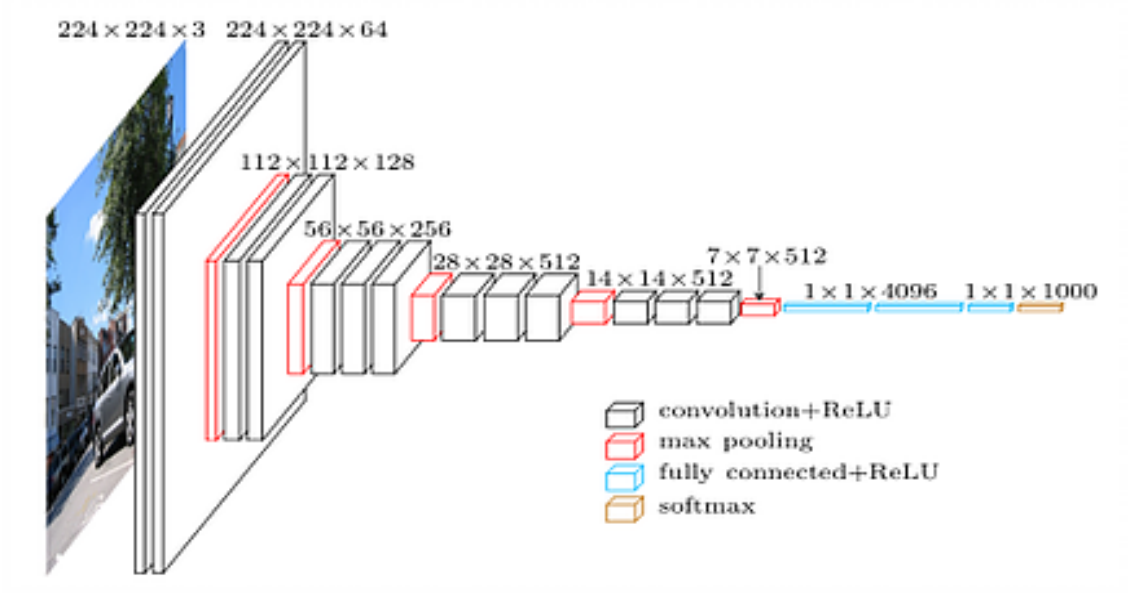


Рисунок 3 - Структура оригінальної мережі VGG16.

Топ-5 похибка — міра точності, яка означає, що при тестуванні вважається, що нейронна мережа правильно класифікувала зображення, якщо зображення відноситься до одного з 5 класів, які отримали найбільшу вагу в ЗНМ.

Такий тип похибки використовується у випадках, коли є певна ймовірність того, що окрім предмету, який належить до певного класу (до якого зображення належить згідно аннотації) на зображенні може бути ще деяка кількість предметів, які відносяться до інших класів, з якими вміє працювати нейромережа. Це є актуальним лише для дуже великих наборів даних і дуже великої кількості класів, зокрема, в змаганні ImageNet вона використовується через ймовірність помилок в розмітці класів.

Вага перших 14 шарів нейромережі була “заморожена”, що означає, що вона в процесі навчання не змінювалась. Останній шар нейромережі, який початково займався класифікацією зображень, було прибрано.

Замість нього зверху (в кінці) згорткової нейронної мережі було додано три шари:

1. Шар на 512 нейрони, до якого застосовувалась регуляризація мережі типу Dropout [26] з активаційною функцією типу Relu[27].

2. Шар на 100 нейронів без регуляризації з активаційною функцією типу Relu.

3. Шар-класифікатор на 10 нейронів (по числу класів) з активаційною функцією типу Softmax. Ця функція широко використовується для шарів-класифікаторів, останніх шарів нейромережі.

Детальніше архітектура отриманої мережі мережі наведена в таблиці 2.

В стовпці “Тип та назва шару” вказано назву шару та тип шару (в дужках). В “Розмірність шару” вказана розмірність шару. Значення None позначає не задану явно кількість зображень, для яких одночасно можуть проводитись обчислення в процесі навчання та ніякого особливого змісту не несе. В стовпці “Кількість параметрів шару” вказана кількість параметрів шару. Типи шарів:

1. InputLayer — умовний тип шару, що означає вхідні дані.

2. Conv2D — двовимірний (працюючий з двовимірними даними) згортковий шар.

3. MaxPooling2D — шар максимального сабсемплінгу.

4. Flatten — умовний шар, який “розплющує” багатовимірний вектор до одновимірного.

5. Dense — звичайний повнозв'язний шар.

6. Dropout — умовний шар, який позначає використання техніки регуляризації Dropout для того шару, який вказаний поряд з ним.

Таблиця 2 - Модифікована архітектура нейронної мережі VGG16.

Тип та назва шару	Розмірність шару	Кількість параметрів шару
input1 (InputLayer)	(None, 144, 432, 3)	0
block1conv1 (Conv2D)	(None, 144, 432, 64)	1792
block1conv2 (Conv2D)	(None, 144, 432, 64)	36928
Block1pool(MaxPooling2D)	(None, 72, 216, 64)	0
block2conv1 (Conv2D)	(None, 72, 216, 128)	73856
block2conv2 (Conv2D)	(None, 72, 216, 128)	147584
block2pool (MaxPooling2D)	(None, 36, 108, 128)	0
block3conv1 (Conv2D)	(None, 36, 108, 256)	295168
block3conv2 (Conv2D)	(None, 36, 108, 256)	590080
block3conv3 (Conv2D)	(None, 36, 108, 256)	590080
block3pool (MaxPooling2D)	(None, 18, 54, 256)	0
block4conv1 (Conv2D)	(None, 18, 54, 512)	1180160
block4conv2 (Conv2D)	(None, 18, 54, 512)	2359808
block4conv3 (Conv2D)	(None, 18, 54, 512)	2359808
block4pool (MaxPooling2D)	(None, 9, 27, 512)	0
block5conv1 (Conv2D)	(None, 9, 27, 512)	2359808
block5conv2 (Conv2D)	(None, 9, 27, 512)	2359808
block5conv3 (Conv2D)	(None, 9, 27, 512)	2359808
block5pool (MaxPooling2D)	(None, 4, 13, 512)	0
flatten1 (Flatten)	(None, 26624)	0
dense1 (Dense) + dropout1 (Dropout)	(None, 512)	13632000
dense2 (Dense)	(None, 100)	51300
dense3 (Dense)	(None, 10)	1010

Використовувався набір даних UrbanSound[21] , що містить 8732 розмічених коротких (4 секунди або менше) аудіосигнали, що розбиті за 10 класами (схематично зображеними на рисунку 3), які належать до 5 глобальних категорій.

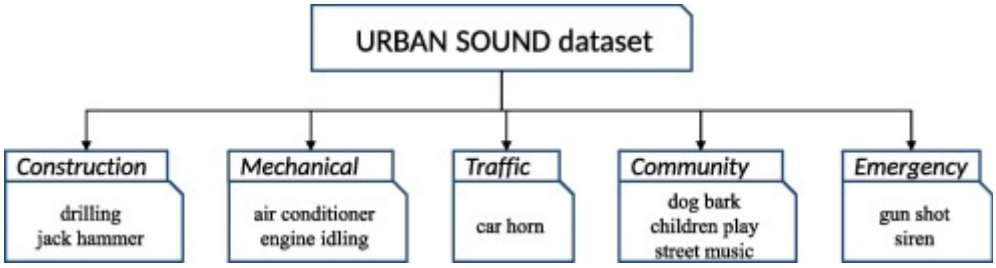


Рисунок 4 - Набір даних Urban Sound за категоріями.

Класи, що наявні в цьому наборі даних:

1. Звук роботи кондіціонера.
2. Звук граючихся дітей.
3. Звук сигналу автомобіля.
4. Звук гавкоту собаки.
5. Звук роботи дрелі
6. Звук пострілу
7. Звук відбійного молотка
8. Звук сирени
9. Звук роботи двигуна
10. Звук вуличної музики

Проте нумерація класів не співпадає з кодуванням класів, які використовувались при навчанні нейромережі. Кодування класів починалось з нуля, та виглядало наступним чином:

0. Звук класу 'air_conditioner'
1. Звук класу 'car_horn'
2. Звук класу 'children_playing'
3. Звук класу 'dog_bark'
4. Звук класу 'drilling'
5. Звук класу 'engine_idling'
6. Звук класу 'gun_shot'
7. Звук класу 'jackhammer'

8.Звук класу 'siren'

9.Звук класу 'street_music'

Для всіх аудиосигналів генерувались спектрограми, які масштабувались до розміру 432 на 144 пікселя. Масштабування виконувалось для зменшення розмірності даних та пришвидшення процесу навчання (щоправда, воно мало негативно вплинути на точність класифікації, знаходження оптимального з точки зору швидкості навчання мережі та метрик роботи алгоритму масштабування є можливим напрямком подальших досліджень).

Для навчання мережі для кожної епохи випадковим чином вибирались 5000 спектрограм аудиосигналів з тренувальної вибірки (5889 аудиосигналів). Нейронна мережа навчалась протягом 20 епох на обчислювальному процесорі Tesla K80. Для імплементації нейромережі була використана мова програмування Python з використанням бібліотеки Keras (з бекендом TensorFlow) у якості високорівневої бібліотеки нейромереж).

Keras — це відкрита нейромережева бібліотека, написана мовою Python, яка є своєрідною надстройкою над фреймворками TensorFlow, Theano та DeepLearning4j. Ця бібліотека містить чисельні реалізації розповсюджених будівельних блоків нейромереж, таких як шари, цільові та передаточні функції, оптимізатори, та багато інструментів для полегшення роботи з зображеннями та текстом.

Валідація проводилась на іншій підвибірці набору даних (2866 аудиосигналів).

Також проводилось дослідження карт активації мережі (class activation map) [27] за допомогою засобів бібліотеки Keras. Для такого дослідження модифікована мережа була навчена повторно на тих же даних в таких же умовах.

Карти активації мережі — це спеціальна техніка, яка використовується для отримання регіонів зображення, які ЗНМ використала для класифікації цього зображення.

Іншими словами, карта активації мережі дозволяє побачити, які регіони зображення релевантні до певного класу, які регіони зображення були найбільш важливими з точки зору ЗНМ під час класифікації.

На рисунку 5 зображений приклад карти активації мережі з оригінальної статті, що гарно ілюструють потенціал цього підходу.

На рисунку 6 зображений приклад карти активації мережі, отриманий в процесі підготовки дипломної роботи, разом з спектрограмою, до якої належить ця карта активації (варто зазначити, що карту активації можливо отримати для одного зображення та всіх класів, куди мережа теоретично могла б це зображення класифікувати, тобто всіх класів взагалі. На цьому рисунку видно, як нейромережа зреагувала на характерні особливості звукового сигналу, характерні для даного класу.

Щоправда, знаходження карт активації класів для інших класів (класів, до яких не відноситься зображення) не несе особливої цінності, тому далі під терміном “карта активації спектрограми” буде розумітись карта активації спектрограми відносно того класу, до якого її класифікувала нейронна мережа, у випадку, якщо класифікація була виконана правильно, в інших випадках така карта розглядатися не буде.

Для використання карти активації мережі в архітектурі ЗНМ має бути шар глобальної усередненої підвибірки, також відомий як шар глобального усередненого пулінгу (global average pooling layer) після останнього згорткового шару, а потім — звичайний повнозв'язний шар.

Це означає, що для дослідження карт активації мережі необхідно внести деякі модифікації до неї, а саме — додати цей шар, що означає необхідність повторного навчання.

Ці обмеження стосуються лише механізму, описаного в оригінальній статті, вже існує модифікація цього підходу, яка дозволяє проводити дослідження карт активації мережі без обмежень на архітектуру [28].

Такі карти використовуються під час процесу навчання ЗНМ для кращого моніторингу процесу. Вони допомогають детектувати помилки навчання ЗНМ, коли вона “звертає увагу” не на характеристики певного об’єкту, а на щось, що цей об’єкт зазвичай супроводжує. Така проблема може бути викликана неправильним вибором даних, помилками в зборі даних, помилками в попередній обробці даних, тощо. Наприклад,прикладом такої помилкової роботи може бути реагування на ландшафт гірської місцевості, де зазвичай пасуться вівці, замість реагування на овець при спробі класифікувати тварин.

Таку помилку важко відслідкувати при звичайними засобами, адже усі метрики будуть показувати високу точність класифікації (за умови, що в наборі даних мало зображень гірської місцевості з іншими тваринами).

Проте така неправильно навчена нейромережа може погано себе показати при спробі використати її для вирішення реальної задачі.

В цілому, можливість такої проблеми існує для більшості методів машинного навчання, і саме тому правильна підготовка даних при застосуванні алгоритмів машинного навчання до реальних задач є дуже важливою.

Для більшості інших алгоритмів відслідкувати таку проблему важко, для цього необхідне мануальне тестування, з залученням експертів в предметній області, адже наприклад в медицині помилки такого плану можуть бути фатальними.

Тому карти активації класів наразі є одним із самих зручних і розповсюджених засобів, які застосовуються з метою відлагодження ЗНМ та уникнення проблем навчання мережі.

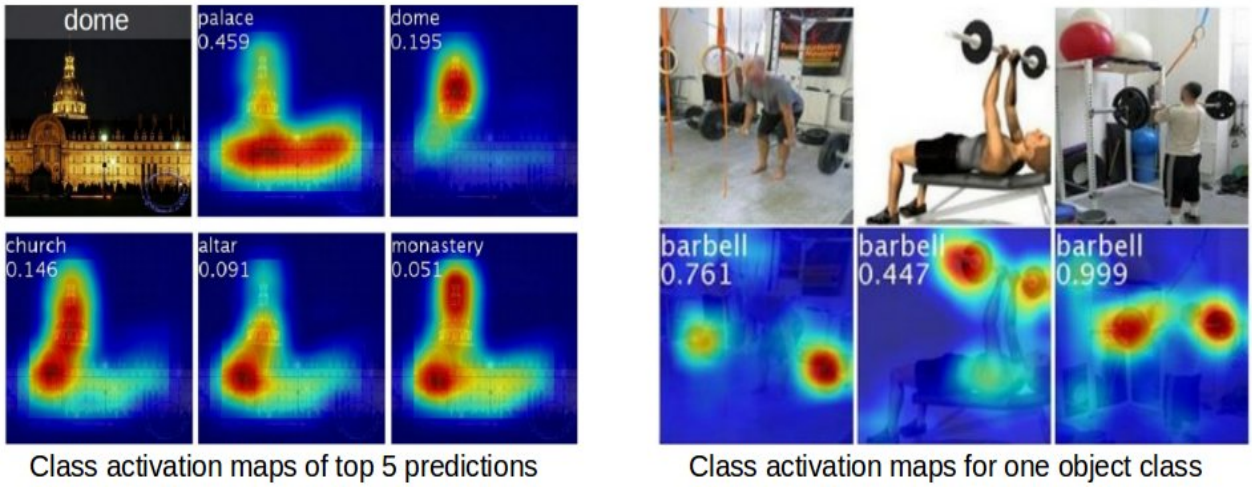


Рисунок 5 - Приклад карт активації класів з оригінальної статті [27].

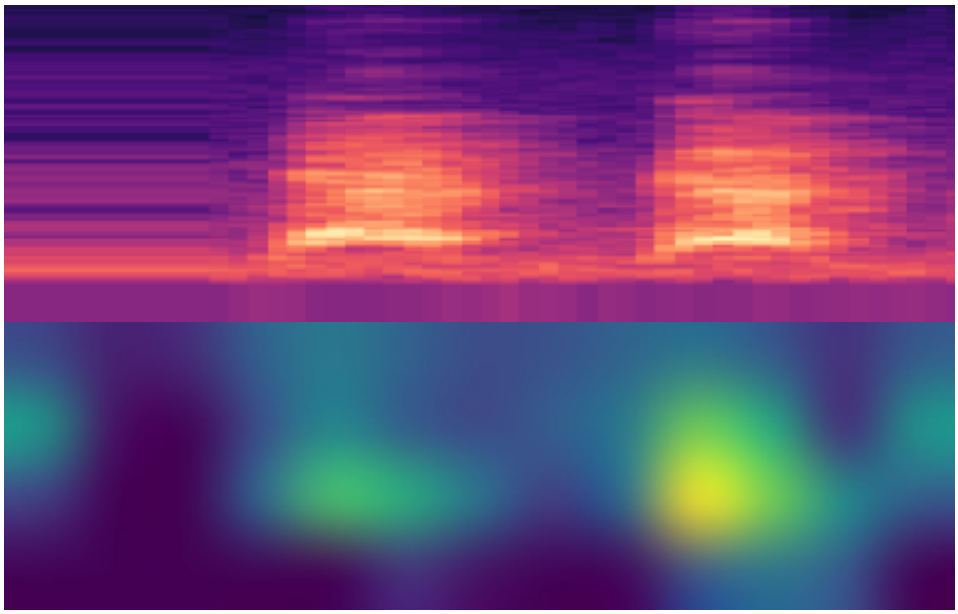


Рисунок 6 - Приклад карти активації мережі на спектрограмі звуку гавкоту собаки.

Також карти активації класів можуть бути використані для частково контрольованої (semi-supervised) розмітки даних на зображеннях, коли спершу нейронна мережа вчиться на задачі класифікації зображень (наприклад, класифікуючи фотографії за наявністю/відсутністю певного класу в найпростішому випадку, або просто навчається на задачі багатокласової

класифікації), а потім за допомогою отриманих карт активації виконується подальша розмітка даних з отриманими координатами об’єкта. Таке неочевидне застосування цієї техніки дозволяє значно зменшити витрати ресурсів на розмітку зображень для задач знаходження координат певного об'єкту.

Застосування карт активації до задачі класифікації звуку здається нелогічним лише на певний погляд. Окрім полегшення відлагодження нейромережі, ця техніка має ще додаткову цінність.

В застосуванні карт активації до задачі класифікації звуку є неочевидна на перший погляд актуальність — адже за допомогою цього методу можливо отримати точну часову інформацію про період певного звуку, яка може бути використана в подальшому. Без використання цього підходу, відомий тільки приблизний час певної аудіоподії. Іншими словами, ми знаємо, що десь на певній спектрограмі, на проміжку часу в декілька секунд присутня інформація про звук пострілу, проте ми не знаємо точно часові координати цієї події. В деяких випадках відсутність такої точної інформації може бути критичною, тому можливість отримати її в автоматичному режимі є перевагою.

Іншими словами, використання карт активації класів для роботи зі звуковими даними може використовуватись для точної часової розмітки даних, що може стати в нагоді при створенні нових наборів даних.

До того ж, такий підхід дає певну теоретичну можливість розрізняти кілька подій, що сталися протягом часового відрізку однієї спектрограми, проте потребує подальших досліджень стосовно практичної доцільності та точності такого підходу.

Іншим застосуванням карт активації до задачі класифікації звуку може бути визначення певних частотних характеристик звуку, характерних для певного класу, хоч актуальність вирішення цієї задачі наразі під питанням.

3.2 Висновки до третього розділу

У даному розділі наведено детальну інформацію про вибрану структуру нейронної мережі, структуру модифікованої мережі та наведено загальну інформацію про набір даних, на якому мережа попередньо навчалась. Також у даному розділі проаналізовано набір даних звуку, на якому проводилось подальше дослідження. Окрім цього, в розділі наведено інформацію про карти активації класів нейромереж.

4. АНАЛІЗ РЕЗУЛЬТАТІВ АЛГОРИТМУ

4.1 Аналіз результатів роботи алгоритму

Комп’ютерні симуляції націлені на те, щоб показати, що розроблений підхід може використовуватись на практичних задачах.

Значення точності (accuracy) та значення крос-ентропії (cross-enrtopy, фактично — цільової функції) для тренувальних та валідаційних (тестових) даних відповідно до епох для модифікації без карт активації класів наведені в таблиці 3.

Враховуючи те, що в даному випадку задача класифікації є мультикласовою, під точністю розуміється усереднена точність по всім класам. Такий підхід можливий за рахунок збалансованого набору даних. У випадку незбалансованих даних більш доречне використання інших метрик точності, наприклад таких як F-метрика, які більш адаптовані для такого випадку.

Аналогічні значення для модифікації з картами активації класів наведені в таблиці 4.

Розшифровка назв стовпців:

1. *Epoch* — номер епохи навчання. Починається з нуля.
2. *Acc* — точність впродовж навчання під час конкретної епохи.
3. *Loss* — функція втрат впродовж епохи.
4. *Val_acc* — точність на валідаційному наборі даних після закінчення епохи.
5. *Val_loss* — функція втрат на валідаційному наборі даних після закінчення епохи.

Для другої таблиці назви стовпців розшифровуються аналогічно.

Таблиця 3 - Точність та функція втрат на тренувальній та тестовій вибірці за епохами на модифікації алгоритму без карт активації класів.

Epoch	acc	loss	val_acc	val_loss
0	0,205882353	2,16289255	0,421391184	1,662312597
1	0,4734	1,51111987	0,549000441	1,246892586
2	0,543234588	1,263427854	0,615942486	1,096455688
3	0,6044	1,131108612	0,652456328	1,002234306
4	0,631652661	1,050745508	0,680645772	0,9164938
5	0,658463385	0,976316643	0,686542221	0,902681704
6	0,675270108	0,925453983	0,703443274	0,881083137
7	0,697879152	0,871120023	0,724191209	0,814838785
8	0,704481793	0,849763241	0,741795001	0,761464201
9	0,7192	0,803677324	0,751434698	0,737887267
10	0,745698279	0,750886343	0,753925711	0,734470606
11	0,740896359	0,759304282	0,769092514	0,673076217
12	0,763105242	0,693873884	0,74992117	0,727731789
13	0,756302521	0,703188447	0,756227534	0,714986764
14	0,7748	0,677046065	0,740430094	0,751529944
15	0,772818255	0,664032378	0,789966576	0,635545341
16	0,7752	0,664175679	0,791480103	0,619718483
17	0,795718287	0,6138943	0,785376168	0,632269362
18	0,798119248	0,580020757	0,7763133	0,67570387
19	0,793517407	0,610786085	0,777385382	0,637479964

Таблиця 4 - Точність та функція втрат на тренувальній та тестовій вибірці за епохами на модифікації алгоритму з картами активації класів.

	acc	loss	val_acc	val_loss
0	0,43797519	1,670177656	0,590717033	1,219106092
1	0,582032813	1,247118539	0,652645519	1,045419377
2	0,6368	1,107066704	0,681591726	0,939647256
3	0,668134508	1,004886177	0,710348742	0,878205808
4	0,681	0,966297323	0,658068992	0,99953433
5	0,702881152	0,89057143	0,701866683	0,870811555
6	0,721288515	0,839353147	0,725042568	0,832328057
7	0,730892357	0,831249726	0,721006496	0,846461158
8	0,74729892	0,762165215	0,775997982	0,689499949
9	0,753701481	0,76900176	0,765308696	0,713266485
10	0,755	0,749770365	0,742921107	0,741735161
11	0,774509804	0,688414648	0,708078451	0,902286281
12	0,772308924	0,686064895	0,791101722	0,634761451
13	0,785914366	0,653409695	0,801917134	0,595732123
14	0,7882	0,638339452	0,681213344	0,989750544
15	0,797838271	0,611894314	0,797218894	0,622141439
16	0,792116847	0,606033793	0,777479977	0,714103602
17	0,798719488	0,611092427	0,767074478	0,731623224
18	0,805	0,598038784	0,801254966	0,610581519
19	0,81452581	0,575626985	0,726556095	0,879533745

Значення точності тренувальної та тестової вибірки відповідно до епох для алгоритму без карт активації класів наведені на рисунку 7. Значення крос-ентропії для тестової та тренувальної вибірки для цього ж алгоритму наведені на рисунку 8.

Для алгоритму з модифікацією для карт активації класів аналогічні значення наведені на рисунках 9 (значення точності відповідно до епох) та 10 (значення крос-ентропії відповідно до епох) відповідно.

Помітно, що процес навчання більш ефективно почався для другої модифікації алгоритму, проте більш стабільно проходив для першої. Також можливо дійти до висновку, що друга модифікація алгоритму почала перенавчатися приблизно на 18-19 епохах. Це видно з того, що точність на тренувальній вибірці почала зростати, а точність на тестовій спадати в той же час.

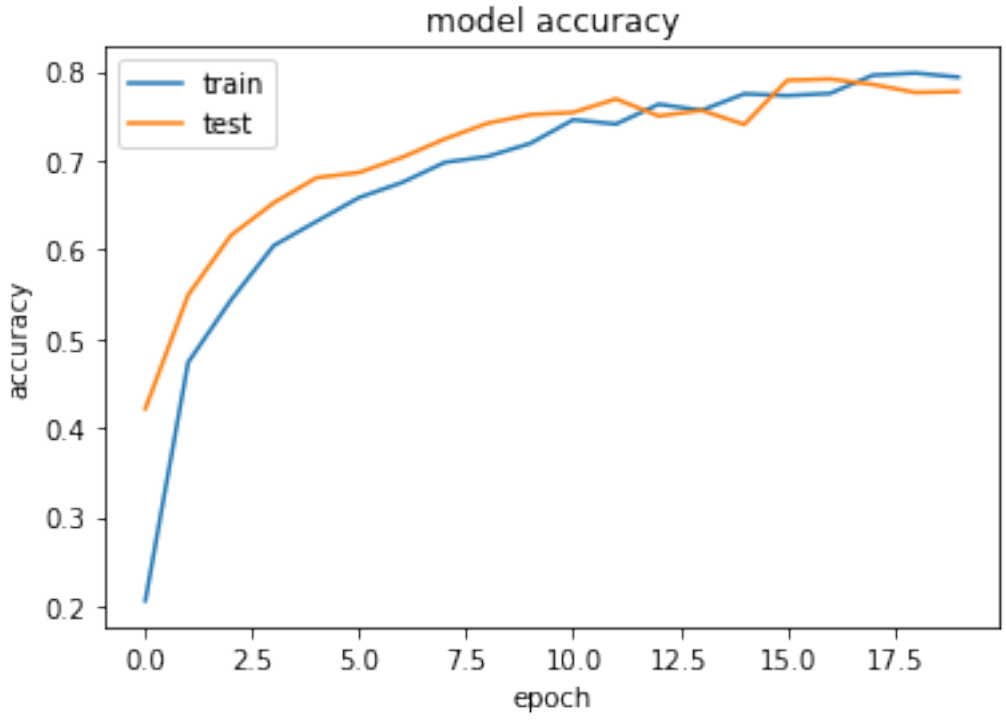


Рисунок 7 - Точність моделі без карт активації класів відповідно до епох.

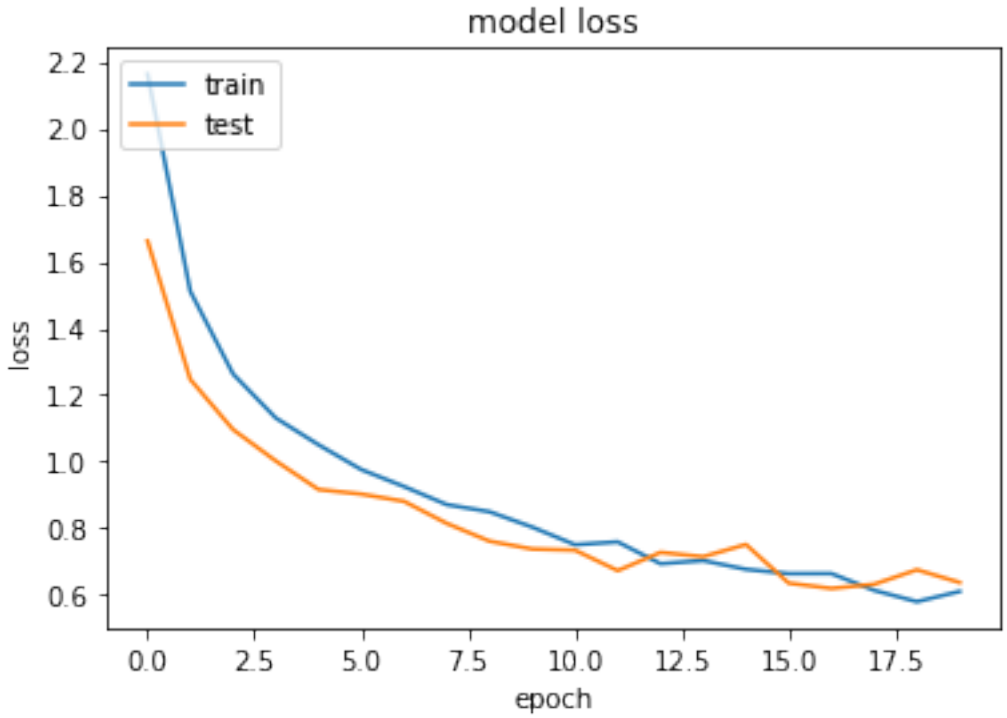


Рисунок 8 - Функція втрат моделі без карт активації класів відповідно до епох.

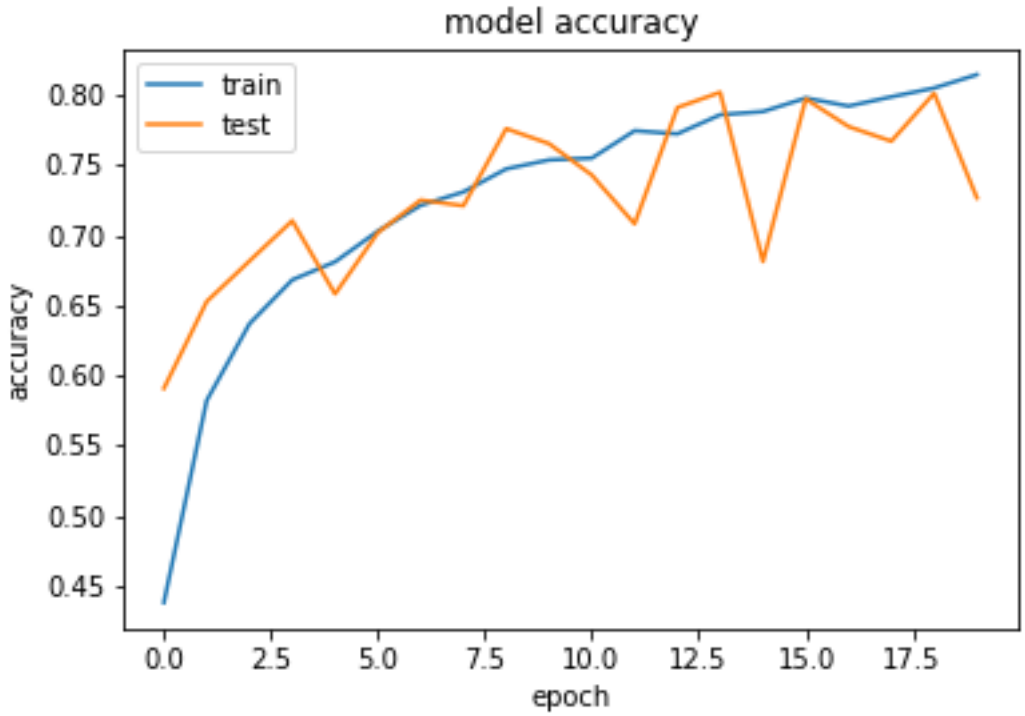


Рисунок 9 - Точність моделі з картами активації класів відповідно до епох.

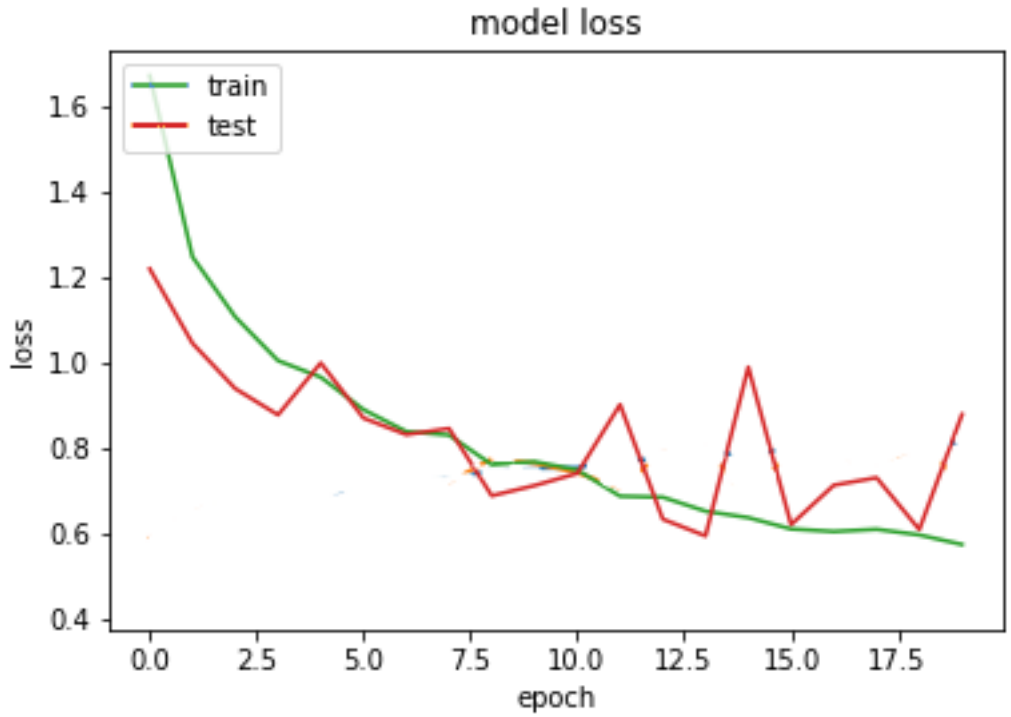


Рисунок 10 - Функція втрат моделі з картами активації класів відповідно до епох.

Приклади отриманих карт активації класів для другої модифікації алгоритму разом з відповідними спектрограмами наведені на рисунках 11,12,13 та 14.На верхній частині кожного рисунку зображена спектрограма, на нижній — карта активації класу відповідно. Видно, що для різних класів ЗНМ по різному створювала карти активації класів.

Яскравість пікселів означає рівень уваги, який нейромережа приділила певній області. Чим світліша область, тим більший вклад регіон вніс до процесу класифікації. Темні регіони, в свою чергу, не дуже вплинули на результат роботи ЗНМ.

Треба зазначити, що не завжди нейронна мережа корректно знаходить регіони, які мають вплинути на класифікацію, що добре видно на рисунку 12. Незважаючи на монотонність і періодичність спектрограми, карта активації класу не є монотонною та періодичною (що не означає, що приклад в результаті був класифікований невірно, проте створює перешкоди в

використанні карт активації для точної часової локалізації події і вимагає подальшого вивчення)

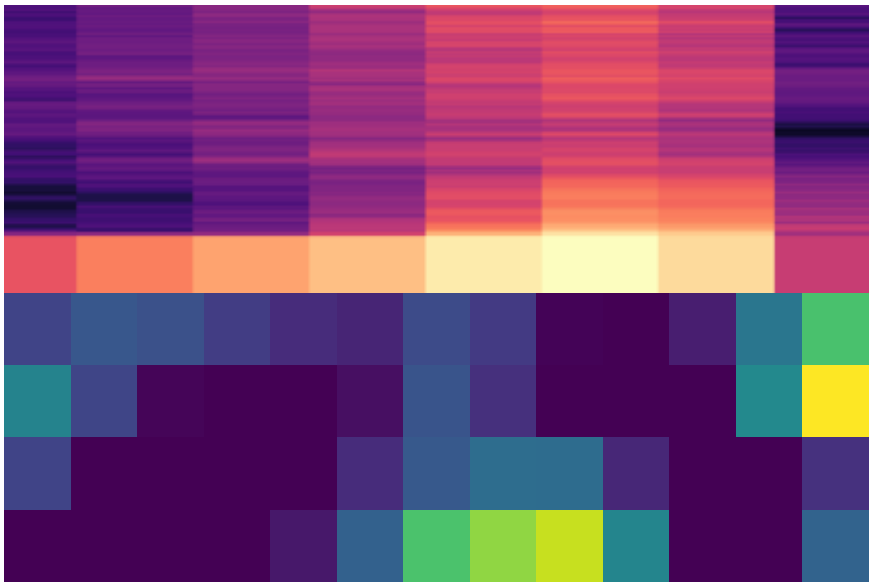


Рисунок 11. Спектрограма звуку відбійного молотка та відповідна карта активації.

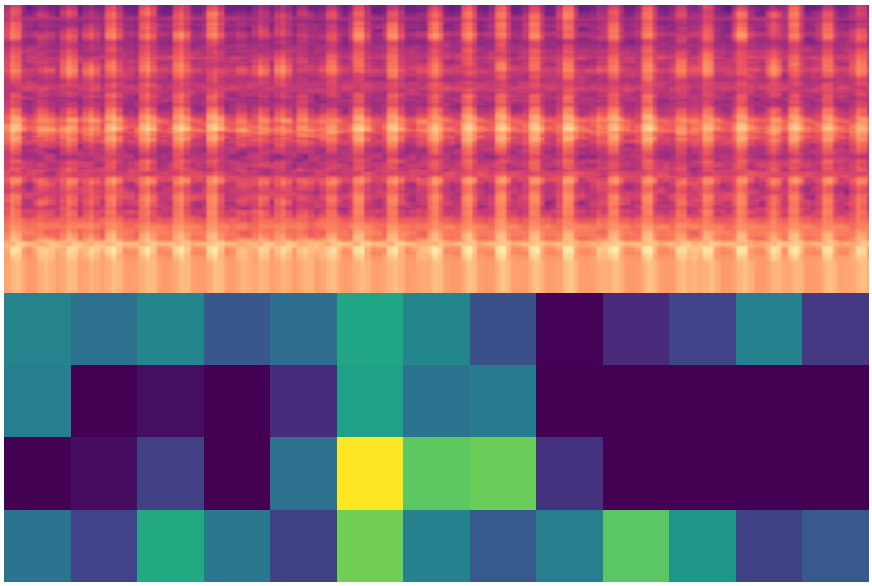


Рисунок 12. Спектрограма звуку роботи двигуна та відповідна карта активації.

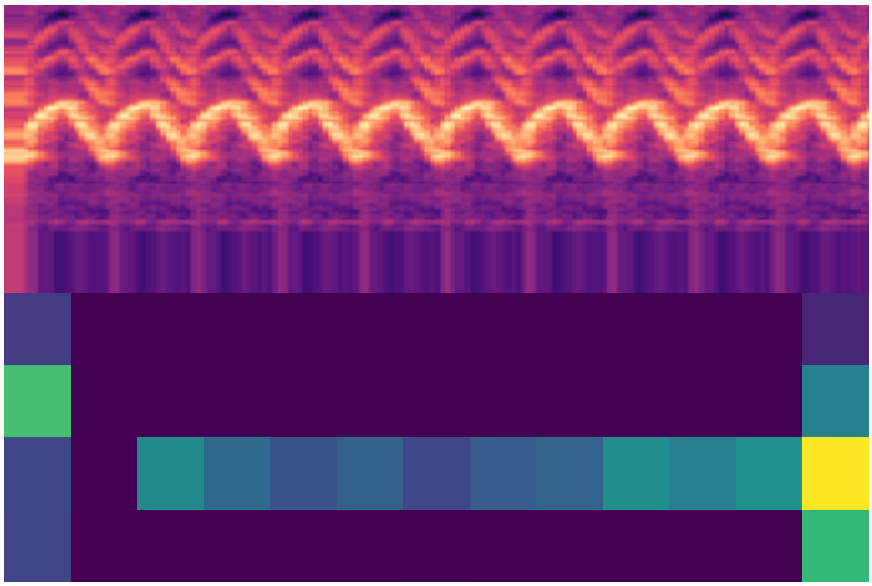


Рисунок 13. Спектрограма звуку гудка автомобіля та відповідна карта активації

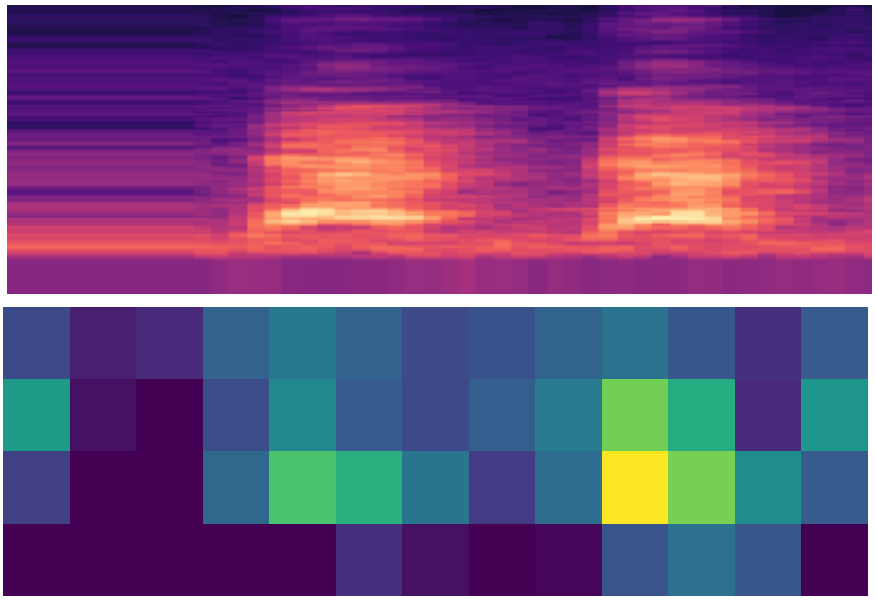


Рисунок 14. Спектрограма звуку гавкоту собаки та відповідна карта активації.

4.2 Висновки до четвертого розділу

У даному розділі виконане тестування розробленої системи на точність роботи на даних та наведені результати роботи системи по класифікації аудіосигналів. В даному розділі було проведено експерименти з двома модифікаціями алгоритму. Також в розділі наведена інформація про процес навчання системи по епохам. Результати роботи системи є прийнятними для використання на практичних задачах. Отримані результати проаналізовано та були зроблені висновки.

ВИСНОВКИ

Для навчання алгоритму без карт активацій класів, як видно з таблиці 3, тестування алгоритму показує достатньо високу результуючу точність підходу (точність класифікації на валідаційній вибірці досягає 77% після 20 епох навчання), що вже є достатнім для практичного використання розробки. Також, як видно з рисунку 7, нейронна мережа не досягла рівня, після якого починається перенавчання (в такому випадку точність на тренувальному наборі даних продовжила б рости, а точність на валідаційному наборі даних почала б спадати), що означає, що у випадку більш довгого навчання нейромережі можливо досягти ще більшої точності.

При порівнянні запропонованого підходу з підходами навчання звичайної рекурентної мережі або згорткової мережі з нуля (які зазвичай використовуються для роботи зі звуком) запропонований підхід має наступні переваги.

1. Час навчання. Цей підхід дозволяє значно скоротити час навчання нейромережі, отже, зменшити використання ресурсів.

2. Підхід демонструє можливість використання нейромережі, навченої на наборі зображень, після подальшого донавчання, розпізнавати аудіосигнали з високою точністю.

3. Підхід дозволяє зменшити кількість даних, необхідних для навчання нейромережі, що є актуальним при невеликому наборі даних.

У подальшому було б доцільно провести більш глибокий порівняльний аналіз різних архітектур попередньо навчених нейромереж на задачі класифікації звуку. Також доцільне дослідження архітектури останніх шарів ЗНМ для підвищення результуючої точності.

Для модифікації алгоритму з використанням карт активації класів ЗНМ справедливе все вищесказане. Як видно з таблиці 4, точність алгоритма на валідаційній вибірці досягала 80%.

Також цікавим є той факт, що ця модифікація алгоритму значно швидше навчалась впродовж перших епох, але процес навчання був більш нестабільним, і йшов стрибками (як видно на рисунку 7). Причини цього є доцільним дослідити впродовж подальшої наукової роботи..

Різниця в графіках навчання пояснюється трохи іншою архітектурою ЗНМ, а саме використанням шару глобального усередненого пулінгу.

ВИКОРИСТАНІ ЛІТЕРАТУРНІ ДЖЕРЕЛА

1. Сидоров К. В. Анализ признаков эмоционально окрашенной речи / К. В. Сидоров, Н. Н. Филатова. // Вестник ТвГТУ. – 2012. – №20. – С. 26–32.
2. Николенко С. И. Конспект лекцій з предмету Automated Speech Recognition [Електронний ресурс] / С. И. Николенко. – 2008. – Режим доступу до ресурсу: https://logic.pdmi.ras.ru/~sergey/teaching/asr/notes-06-features.pdf (дата звернення 08.05.2018)
3. Trevor H. The elements of statistical learning. Second edition / H. Trevor, R. Tibshirani., 2008.
4. Шеремет О. Метод опорних векторів / О. Шеремет, В. Садовой. // Мат. Мод. № 1. – 2013. – №28. – С. 13.
5. Domingos P. On the optimality of the simple Bayesian classifier under zero-one loss / P. Domingos, P. Pazzani. // Machine Learning. – 1997. – №29. – С. 103–137.
6. Зорін Ю. М. Конспект лекцій з предмету «Комп`ютерні системи штучного інтелекту» [Електронний ресурс] / Ю. М. Зорін – Режим доступу до ресурсу: http://m.scs.kpi.ua/course/view.php?id=107(дата звернення 08.05.2018)
7. Біла Н. І. Інформаційні системи та технології в управлінні. Теоретичні відомості і завдання до лабораторних робіт / Н. І. Біла. – С. 16–19.
8. Goodfellow I. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville., 2016.
9. LeCun Y. LeNet-5, convolutional neural networks [Електронний ресурс] / Yann LeCun – Режим доступу до ресурсу: http://yann.lecun.com/exdb/lenet/ (дата звернення 08.05.2018)
10. Zhang W. Shift-invariant pattern recognition neural network and its optical architecture / Wei Zhang. // Proceedings of annual conference of the Japan Society of Applied Physics. – 1988.

11. Zhang W. Parallel distributed processing model with local space-invariant interconnections and its optical architecture / Wei Zhang. // Applied Optics. – 1990. – №29. – С. 4790–4797.
12. Subject independent facial expression recognition with robust face detection using a convolutional neural network / M.Matusugu, M. Katsuhiko, Y. Mitari, Y. Kaneda. // Neural Networks. – 2003. – №16. – С. 555–559.
13. Николенко С. И. Глубокое обучение. Погружение в мир нейронных сетей. / С. И. Николенко, А. А. Кадурин, Е. О. Архангельская., 2018. – 480 с.
14. Glorot X. Understanding the difficulty of training deep feedforward neural networks / X. Glorot, Y. Bengio. // Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. – 2010.
15. Матеріали курсу CS231n: Convolutional Neural Networks for Visual Recognition [Електронний ресурс] – Режим доступу до ресурсу: https://cs231n.github.io/convolutional-networks/ (дата звернення 08.05.2018)
16. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. / D. Scherer, A. Müller, C. Andreas, S. Behnke // Springer. – 2010. – С. 92–101
17. Graham B. Fractional Max Pooling [Електронний ресурс] / B. Graham. – 2014. – Режим доступу до ресурсу: https://arxiv.org/abs/1412.6071 (дата звернення 08.05.2018)
18. Striving for Simplicity: The All Convolutional Net [Електронний ресурс] / J.Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller. – 2014. – Режим доступу до ресурсу: https://arxiv.org/abs/1412.6806 (дата звернення 08.05.2018)
19. Grel T. Region of interest pooling explained. deepsense.io [Електронний ресурс] / Grel. – 2017. – Режим доступу до ресурсу:

https://blog.deepsense.ai/region-of-interest-pooling-explained/(дата звернення 08.05.2018)

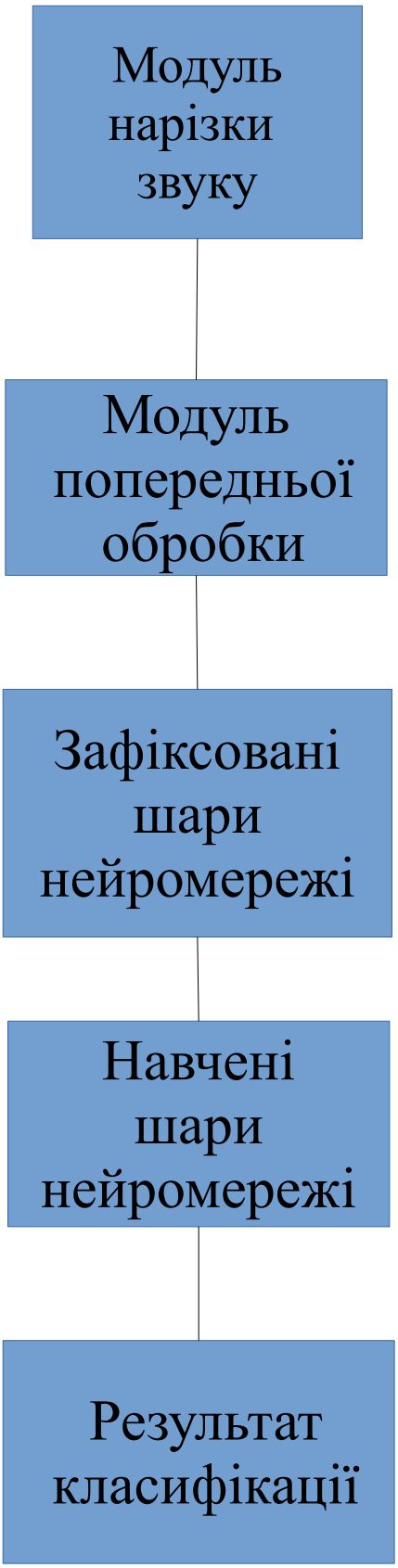
20. Girshick R. Fast R-CNN [Електронний ресурс] / Girshick – Режим доступу до ресурсу: https://arxiv.org/abs/1504.08083 (дата звернення 08.05.2018)
21. Krizhevsky A. Imagenet classification with deep convolutional neural networks / A. Krizhevsky, I. Sutskever, G. Hinton. // Advances in Neural Information Processing Systems. – 2012. – №1. – С. 1097–1105.
22. Dropout: A Simple Way to Prevent Neural Networks from overfitting / [S. Nitish, G. Hinton, A. Krizhevsky та ін.]. // Journal of Machine Learning Research. – 2014. – №15. – С. 1929–1958.
23. Salamon J. A Dataset and Taxonomy for Urban Sound Research / J. Salamon, C. Jacoby, J. Bello. // 22nd ACM International Conference on Multimedia, Orlando USA. – 2014.
24. ImageNet: A Large-Scale Hierarchical Image Database / [J. Deng,W. Dong, R. Socher, L.-J. Li та ін.]. // CVPR09 – 2009
25. Gwardys G. Deep image features in music information retrieval / G. Gwardys, D. Grzywczak. // International Journal of Electronics and Telecommunications. – 2014. – №60. – С. 321–326.
26. Simonyan K. Very Deep Convolutional Networks for Large-Scal Image Recognition [Електронний ресурс] / K. Simonyan. – 2014. – Режим доступу до ресурсу: http://arxiv.org/abs/1409.1556 (дата звернення 08.05.2018)
27. Improving neural networks by preventing co-adaptation of feature detectors [Електронний ресурс] / [G. Hinton, S. Nitish, A. Krizhevsky та ін.]. – 2012. – Режим доступу до ресурсу: https://arxiv.org/abs/1207.0580 (дата звернення 08.05.2018)
28. Learning Deep Features for Discriminative Localization [Електронний ресурс] / [B. Zhou, A. Khosla, A. Lapedriza та ін.]. – 2015. – Режим доступу до ресурсу: https://arxiv.org/abs/1512.04150 (дата звернення 08.05.2018)

29. Visual Explanations from Deep Networks via Gradient-based Localization [Електронний ресурс] / [R. Selvaraju, M. Cogswell, A. Das та ін.]. – 2017. – Режим доступу до ресурсу: https://arxiv.org/abs/1610.02391 (дата звернення 08.05.2018)

Структура отриманої нейромережі

Тип та назва шару	Розмірність шару	Кількість параметрів шару
input1 (InputLayer)	(144, 432, 3)	0
block1conv1 (Conv2D)	(144, 432, 64)	1792
block1conv2 (Conv2D)	(144, 432, 64)	36928
Block1pool(MaxPooling2D)	(72, 216, 64)	0
block2conv1 (Conv2D)	(72, 216, 128)	73856
block2conv2 (Conv2D)	(72, 216, 128)	147584
block2pool (MaxPooling2D)	(36, 108, 128)	0
block3conv1 (Conv2D)	(36, 108, 256)	295168
block3conv2 (Conv2D)	(36, 108, 256)	590080
block3conv3 (Conv2D)	(36, 108, 256)	590080
block3pool (MaxPooling2D)	(18, 54, 256)	0
block4conv1 (Conv2D)	(18, 54, 512)	1180160
block4conv2 (Conv2D)	(18, 54, 512)	2359808
block4conv3 (Conv2D)	(18, 54, 512)	2359808
block4pool (MaxPooling2D)	(9, 27, 512)	0
block5conv1 (Conv2D)	(9, 27, 512)	2359808
block5conv2 (Conv2D)	(9, 27, 512)	2359808
block5conv3 (Conv2D)	(9, 27, 512)	2359808
block5pool (MaxPooling2D)	(4, 13, 512)	0
Межа “заморозки” ваги		
flatten1 (Flatten)	(26624)	0
dense1 (Dense) + dropout1 (Dropout)	(512)	13632000
dense2 (Dense)	(100)	51300
dense3 (Dense)	(10)	1010

Структурна схема програми

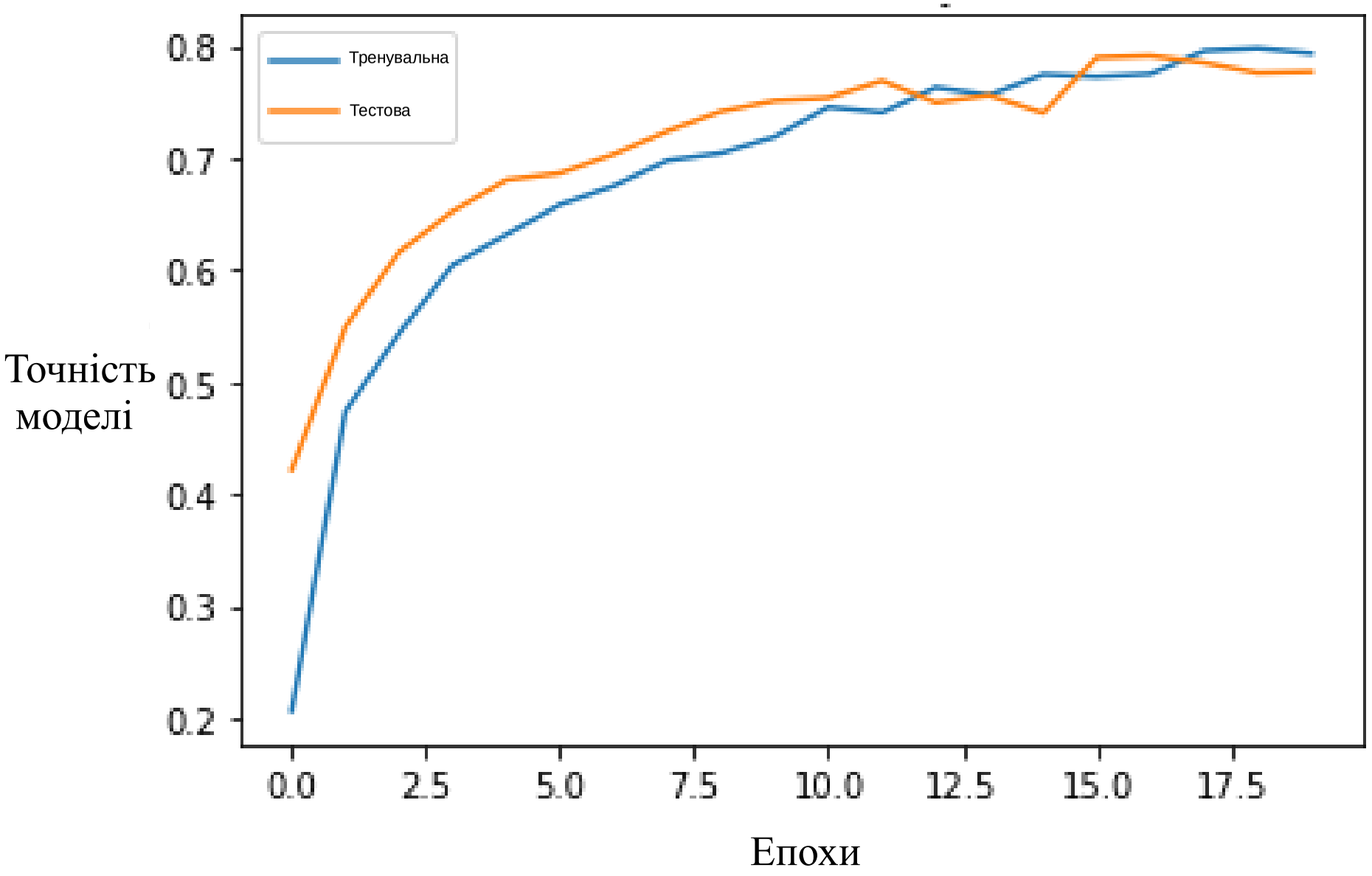


Гончаров Д.А.

Точність та функція втрат на тренувальній та тестовій вибірці за епохами для модифікації алгоритму без карт активації класів.

Epoch	acc	loss	val_acc	val_loss
0	0,205882353	2,16289255	0,421391184	1,662312597
1	0,4734	1,51111987	0,549000441	1,246892586
2	0,543234588	1,263427854	0,615942486	1,096455688
3	0,6044	1,131108612	0,652456328	1,002234306
4	0,631652661	1,050745508	0,680645772	0,9164938
5	0,658463385	0,976316643	0,686542221	0,902681704
6	0,675270108	0,925453983	0,703443274	0,881083137
7	0,697879152	0,871120023	0,724191209	0,814838785
8	0,704481793	0,849763241	0,741795001	0,761464201
9	0,7192	0,803677324	0,751434698	0,737887267
10	0,745698279	0,750886343	0,753925711	0,734470606
11	0,740896359	0,759304282	0,769092514	0,673076217
12	0,763105242	0,693873884	0,74992117	0,727731789
13	0,756302521	0,703188447	0,756227534	0,714986764
14	0,7748	0,677046065	0,740430094	0,751529944
15	0,772818255	0,664032378	0,789966576	0,635545341
16	0,7752	0,664175679	0,791480103	0,619718483
17	0,795718287	0,6138943	0,785376168	0,632269362
18	0,798119248	0,580020757	0,7763133	0,67570387
19	**0,793517407**	**0,610786085**	**0,777385382**	**0,637479964**

Точність моделі без карт активації класів відповідно до епох.



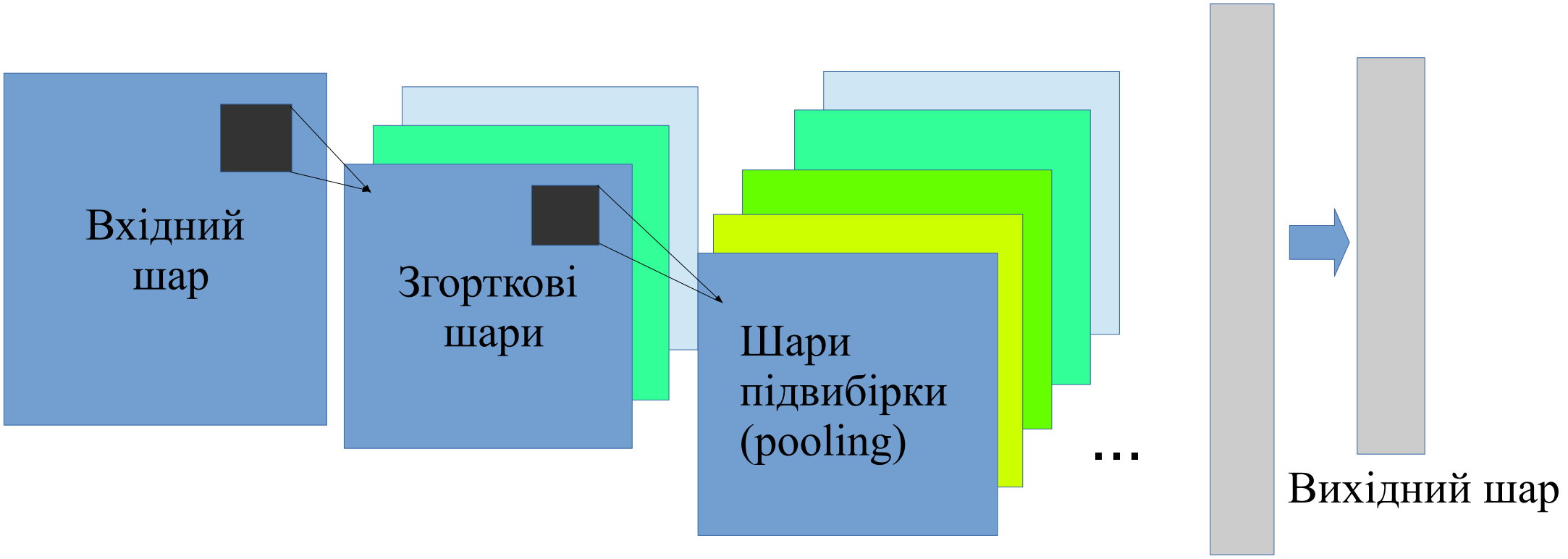
Гончаров Д.А.

Точність та функція втрат на тренувальній та тестовій вибірці за епохами на модифікації алгоритму з картами активації класів

	acc	loss	val_acc	val_loss
0	0,43797519	1,670177656	0,590717033	1,219106092
1	0,582032813	1,247118539	0,652645519	1,045419377
2	0,6368	1,107066704	0,681591726	0,939647256
3	0,668134508	1,004886177	0,710348742	0,878205808
4	0,681	0,966297323	0,658068992	0,99953433
5	0,702881152	0,89057143	0,701866683	0,870811555
6	0,721288515	0,839353147	0,725042568	0,832328057
7	0,730892357	0,831249726	0,721006496	0,846461158
8	0,74729892	0,762165215	0,775997982	0,689499949
9	0,753701481	0,76900176	0,765308696	0,713266485
10	0,755	0,749770365	0,742921107	0,741735161
11	0,774509804	0,688414648	0,708078451	0,902286281
12	0,772308924	0,686064895	0,791101722	0,634761451
13	0,785914366	0,653409695	0,801917134	0,595732123
14	0,7882	0,638339452	0,681213344	0,989750544
15	0,797838271	0,611894314	0,797218894	0,622141439
16	0,792116847	0,606033793	0,777479977	0,714103602
17	0,798719488	0,611092427	0,767074478	0,731623224
18	0,805	0,598038784	0,801254966	0,610581519
19	**0,81452581**	**0,575626985**	**0,726556095**	**0,879533745**

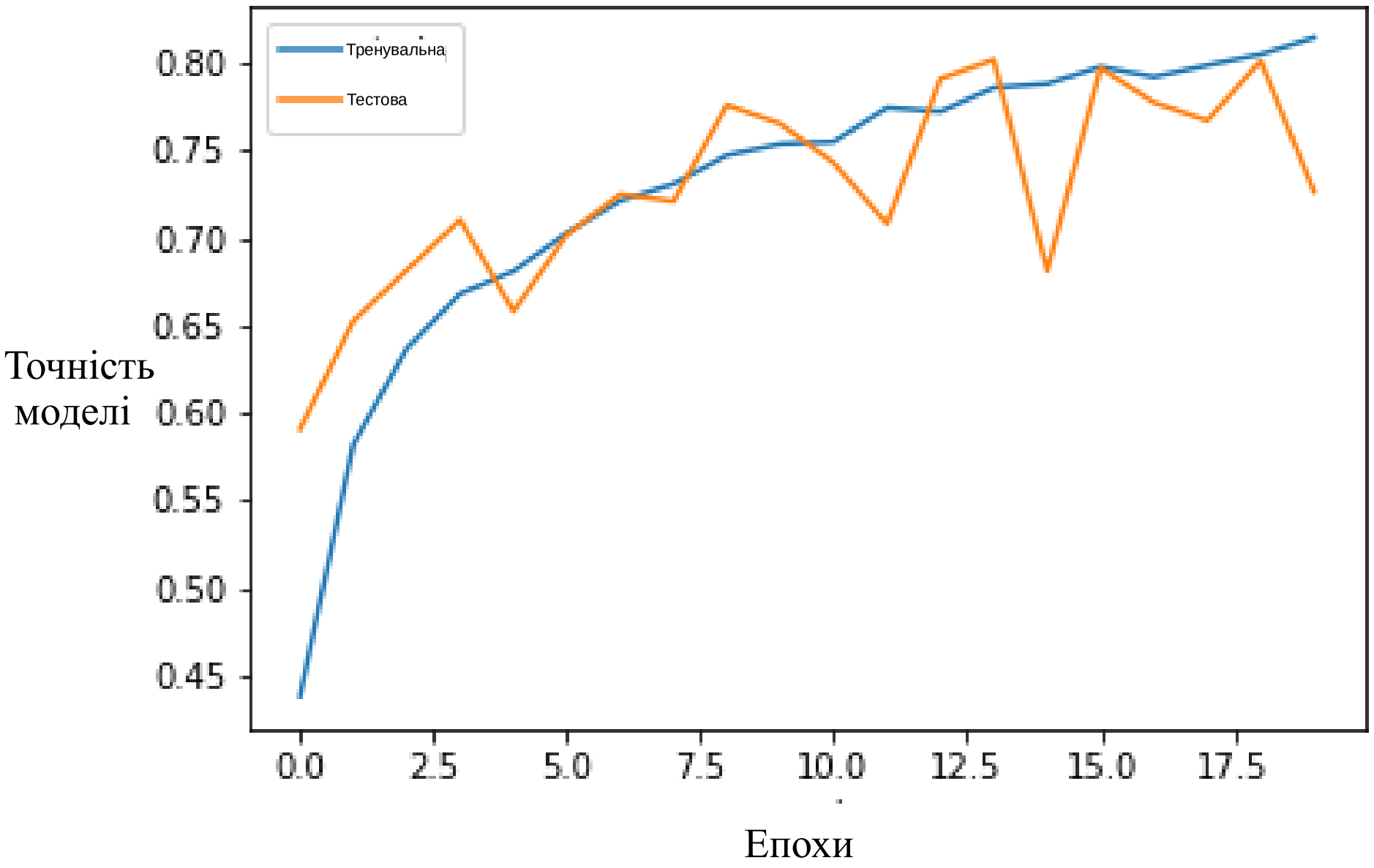
Гончаров Д.А.

Структурна схема роботи згорткової нейромережі



Гончаров Д.А.

Точність моделі з картами активації класів відповідно до епох.



Гончаров Д.А.

Додаток до диплому. Гончаров Данило. Файл Jupyter notebook.

Імпорт бібліотек:

In [8]:

```
import glob
import os
import librosa
import numpy as np
import matplotlib.pyplot as plt
#import tensorflow as tf
from matplotlib.pyplot import specgram
from tqdm import tqdm
```

Визначення функцій:

In [9]:

```
def load_sound_files(file_paths):
    raw_sounds = []
    for fp in tqdm(file_paths):
        X,sr = librosa.load(fp)
        raw_sounds.append(X)
    return raw_sounds

def plot_waves(sound_names,raw_sounds):
    i = 1
    fig = plt.figure(figsize=(25,60), dpi = 900)
    for n,f in zip(sound_names,raw_sounds):
        plt.subplot(10,1,i)
        librosa.display.waveplot(np.array(f),sr=22050)
        plt.title(n.title())
        i += 1
    plt.suptitle("Figure 1: Waveplot",x=0.5, y=0.915,fontsize=18)
    plt.show()

def plot_specgram(sound_names,raw_sounds):
    i = 1
    fig = plt.figure(figsize=(25,60), dpi = 900)
    for n,f in zip(sound_names,raw_sounds):
        plt.subplot(10,1,i)
        specgram(np.array(f), Fs=22050)
        plt.title(n.title())
        i += 1
    plt.suptitle("Figure 2: Spectrogram",x=0.5, y=0.915,fontsize=18)
    plt.show()

def plot_log_power_specgram(sound_names,raw_sounds):
    i = 1
    fig = plt.figure(figsize=(25,60), dpi = 900)
    for n,f in zip(sound_names,raw_sounds):
        plt.subplot(10,1,i)
        D = librosa.logamplitude(np.abs(librosa.stft(f))**2, ref_power=np.max)
        librosa.display.specshow(D,x_axis='time' ,y_axis='log')
        plt.title(n.title())
        i += 1
    plt.suptitle("Figure 3: Log power spectrogram",x=0.5, y=0.915,fontsize=18)
    plt.show()
```

In [7]:

```
sound_file_paths = ["57320-0-0-7.wav","24074-1-0-3.wav","15564-2-0-1.wav","31323-3-0-1.wav",
"46669-4-0-35.wav","89948-5-0-0.wav","40722-8-0-4.wav",
"103074-7-3-2.wav","106905-8-0-0.wav","108041-9-0-4.wav"]

prefix="UrbanSound8K/audio/fold1/"
sound_file_paths=[prefix+x for x in sound_file_paths]

sound_names = ["air conditioner","car horn","children playing",
"dog bark","drilling","engine idling", "gun shot",
"jackhammer","siren","street music"]

raw_sounds = load_sound_files(sound_file_paths)
```

```
100%|██████████| 10/10 [00:03<00:00,  2.77it/s]
```

Спектрограма:

In [ ]:

```
%matplotlib inline

#plot_waves(sound_names,raw_sounds)
#plot_specgram(sound_names,raw_sounds)
plot_log_power_specgram(sound_names,raw_sounds)
```

Кодування класів:

In [10]:

```
from tqdm import tqdm
def extract_feature(file_name):
    X, sample_rate = librosa.load(file_name)
    stft = np.abs(librosa.stft(X))
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,axis=0)
    chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)
    mel = np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T,axis=0)
    contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T,axis=0)
    tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X),
    sr=sample_rate).T,axis=0)
    return mfccs,chroma,mel,contrast,tonnetz

def parse_audio_files(parent_dir,sub_dirs,file_ext="*.wav"):
    features, labels = np.empty((0,193)), np.empty(0)
    for label, sub_dir in tqdm(enumerate(sub_dirs)):
        for fn in glob.glob(os.path.join(parent_dir, sub_dir, file_ext)):
            try:
                mfccs, chroma, mel, contrast,tonnetz = extract_feature(fn)
            except Exception as e:
                print("Error encountered while parsing file: ", fn)
                continue
            ext_features = np.hstack([mfccs,chroma,mel,contrast,tonnetz])
            features = np.vstack([features,ext_features])
            #print(fn)
            labels = np.append(labels, fn.split('/')[3].split('-')[1])
    return np.array(features), np.array(labels, dtype = np.int)

def one_hot_encode(labels):
    n_labels = len(labels)
    n_unique_labels = len(np.unique(labels))
    one_hot_encode = np.zeros((n_labels,n_unique_labels))
    one_hot_encode[np.arange(n_labels), labels] = 1
    return one_hot_encode
```

In [ ]:

```
parent_dir = 'UrbanSound8K/audio'
tr_sub_dirs = ["fold1","fold2"]
ts_sub_dirs = ["fold3"]
tr_features, tr_labels = parse_audio_files(parent_dir,tr_sub_dirs)
ts_features, ts_labels = parse_audio_files(parent_dir,ts_sub_dirs)

tr_labels = one_hot_encode(tr_labels)
ts_labels = one_hot_encode(ts_labels)
```

In [3]:

```
import pickle
# with open("tr_features.pkl",'wb') as f:
#     pickle.dump(tr_features,f)
# with open("tr_labels.pkl",'wb') as f:
#     pickle.dump(tr_labels,f)
# with open("ts_features.pkl",'wb') as f:
#     pickle.dump(ts_features,f)
# with open("ts_labels.pkl",'wb') as f:
#     pickle.dump(ts_labels,f)
with open("tr_features.pkl",'rb') as f:
    tr_features=pickle.load(f)
with open("tr_labels.pkl",'rb') as f:
    tr_labels=pickle.load(f)
with open("ts_features.pkl",'rb') as f:
    ts_features=pickle.load(f)
with open("ts_labels.pkl",'rb') as f:
    ts_labels=pickle.load(f)
```

In [ ]:

```
#"UrbanSound8K/audio/fold1/180937-7-2-7.wav".split('/')[3].split('-')[1]
```

Тестування логістичної регресії для отримання базового предиктора:

In [19]:

```
# result=extract_feature("/home/leon/Научная работа/UrbanSound8K/audio/fold1/14113-4-0-1.wav")
# print(len(result[0]))
# print(len(result[1]))
# print(len(result[2]))
# print(len(result[3]))
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
lgst=MLPClassifier(hidden_layer_sizes=(1000,25))
corrector=np.array([0,1,2,3,4,5,6,7,8,9])
corrected_tr_labels=np.sum(corrector*tr_labels,axis=1)
corrected_ts_labels=np.sum(corrector*ts_labels,axis=1)
lgst.fit(tr_features, corrected_tr_labels)
prediction=lgst.predict(ts_features)
confusion_matrix(corrected_ts_labels,prediction)
```

Out[19]:

```
array([[ 5,  6, 22, 14,  8,  1,  0, 10,  2, 32],
       [ 0, 32,  0,  7,  1,  0,  2,  0,  0,  1],
       [ 0,  0, 52, 24,  4,  0,  1,  0,  3, 16],
       [ 0,  2, 14, 54,  2,  3,  4,  8,  2, 11],
       [ 5,  0,  0, 12, 28,  0, 13, 30,  6,  6],
       [29,  3,  0,  0,  0, 17,  2, 38,  0, 18],
       [ 0,  4,  0,  2,  5,  0, 22,  1,  0,  2],
       [16,  0,  0,  0, 13,  2,  0,  6, 17, 66],
       [ 2,  2,  0, 14,  0, 10,  0, 18, 62, 11],
       [ 1,  0, 16, 10, 11,  4,  8,  0,  2, 48]])
```

Diploma appendix, Honcharov Danylo

9 мая 2018 г.

Додаток до диплому. Гончаров Данило. Файл Jupyter notebook.
Імпорт бібліотек:

```
In [ ]: import shutil
        import os
        import numpy as np
```

Тестування апаратури:

```
In [ ]: from tensorflow.python.client import device_lib
        print(device_lib.list_local_devices())
```

Код для розбиття даних:

```
In [ ]: def split_dataset_into_test_and_train_sets(all_data_dir, training_data_dir, testing_data_dir, te
            # Recreate testing and training directories
            if testing_data_dir.count('/') > 1:
                shutil.rmtree(testing_data_dir, ignore_errors=False)
                os.makedirs(testing_data_dir)
                print("Successfully cleaned directory " + testing_data_dir)
            else:
                print("Refusing to delete testing data directory " + testing_data_dir + " as we prevent

            if training_data_dir.count('/') > 1:
                shutil.rmtree(training_data_dir, ignore_errors=False)
                os.makedirs(training_data_dir)
                print("Successfully cleaned directory " + training_data_dir)
            else:
                print("Refusing to delete testing data directory " + training_data_dir + " as we prevent

            num_training_files = 0
            num_testing_files = 0
            for subdir, dirs, files in os.walk(all_data_dir):
                category_name = os.path.basename(subdir)

                # Don't create a subdirectory for the root directory
                print(category_name + " vs " + os.path.basename(all_data_dir))
                if category_name == os.path.basename(all_data_dir):
                    continue

                training_data_category_dir = training_data_dir + '/' + category_name
                testing_data_category_dir = testing_data_dir + '/' + category_name

                if not os.path.exists(training_data_category_dir):
```

```
                os.mkdir(training_data_category_dir)

            if not os.path.exists(testing_data_category_dir):
                os.mkdir(testing_data_category_dir)

            for file in files:
                input_file = os.path.join(subdir, file)
                if np.random.rand(1) < testing_data_pct:
                    shutil.copy(input_file, testing_data_dir + '/' + category_name + '/' + file)
                    num_testing_files += 1
                else:
                    shutil.copy(input_file, training_data_dir + '/' + category_name + '/' + file)
                    num_training_files += 1

        print("Processed " + str(num_training_files) + " training files.")
        print("Processed " + str(num_testing_files) + " testing files.")
```

```
In [ ]: split_dataset_into_test_and_train_sets("UrbanSound8K/spectrograms/","UrbanSound8K/spectrograms_t
```

```
In [ ]: #validation
        split_dataset_into_test_and_train_sets("UrbanSound8K/spectrograms/spectrograms_test_val/","Urban
```

Створення та навчання моделі:

```
In [ ]: from keras import applications
        from keras.preprocessing.image import ImageDataGenerator
        from keras import optimizers
        from keras.models import Sequential, Model
        from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D
        from keras import backend as k
        from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping

        #img_width, img_height = 864, 288
        img_width, img_height = 432, 144

        train_data_dir = "floyd_dataset/spectrograms_train"
        validation_data_dir = "floyd_dataset/spectrograms_val"
        nb_train_samples = 8729
        nb_validation_samples = 2823
        batch_size = 8
        epochs = 50

        model = applications.VGG19(weights = "imagenet", include_top=False, input_shape = (img_height, i

        """
        Layer (type)                 Output Shape              Param #
        =================================================================
        input_1 (InputLayer)         (None, 256, 256, 3)       0
        _________________________________________________________________
        block1_conv1 (Conv2D)        (None, 256, 256, 64)      1792
        _________________________________________________________________
        block1_conv2 (Conv2D)        (None, 256, 256, 64)      36928
        _________________________________________________________________
        block1_pool (MaxPooling2D)   (None, 128, 128, 64)      0
        _________________________________________________________________
```

```
block2_conv1 (Conv2D)        (None, 128, 128, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 128, 128, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 64, 64, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 64, 64, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 64, 64, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 64, 64, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 64, 64, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 32, 32, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 32, 32, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 32, 32, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 32, 32, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 32, 32, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 16, 16, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 8, 8, 512)         0
=================================================================
Total params: 20,024,384.0
Trainable params: 20,024,384.0
Non-trainable params: 0.0
"""

# Freeze the layers which you don't want to train. Here I am freezing the first 5 layers.
for layer in model.layers[:15]:
    layer.trainable = False

#Adding custom Layers
x = model.output
x = Flatten()(x)
x = Dense(1024, activation="relu")(x)
x = Dropout(0.5)(x)
x = Dense(1024, activation="relu")(x)
predictions = Dense(10, activation="softmax")(x)

# creating the final model
```

```
model_final = Model(input = model.input, output = predictions)

# compile the model
model_final.compile(loss = "categorical_crossentropy", optimizer = optimizers.SGD(lr=0.0001, mor

# Initiate the train and test generators with data Augumentation
train_datagen = ImageDataGenerator(
rescale = 1./255,
horizontal_flip = True,
fill_mode = "nearest",
zoom_range = 0.3,
width_shift_range = 0.3,
height_shift_range=0.3)

test_datagen = ImageDataGenerator(
rescale = 1./255,
horizontal_flip = True,
fill_mode = "nearest",
zoom_range = 0.3,
width_shift_range = 0.3,
height_shift_range=0.3)

train_generator = train_datagen.flow_from_directory(
train_data_dir,
target_size = (img_height, img_width),
batch_size = batch_size,
class_mode = "categorical")

validation_generator = test_datagen.flow_from_directory(
validation_data_dir,
target_size = (img_height, img_width),
class_mode = "categorical")

# Save the model according to the conditions
checkpoint = ModelCheckpoint("vgg16_1.h5", monitor='val_acc', verbose=1, save_best_only=True, sa
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=10, verbose=1, mode='auto')


# Train the model
model_final.fit_generator(
train_generator,
samples_per_epoch = nb_train_samples,
epochs = epochs,
validation_data = validation_generator,
nb_val_samples = nb_validation_samples,
callbacks = [checkpoint, early])
```

Зберігання моделі:

```
In [ ]: model_final.save("keras_model_VGG19")
```

Друк навчальної інформації:

```
In [ ]: print(history.history.keys())
```

Графіки:

```
In [ ]: import matplotlib.pyplot as plt
        plt.plot(history.history['acc'])
        plt.plot(history.history['val_acc'])
        plt.title('model accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.show()
        # summarize history for loss
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('model loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.show()
```

```
In [ ]: with open("log_f",'w') as f:
            f.write(history)
```

Перезавантаження моделі та тестування:

```
In [ ]: from keras.models import load_model
        model_final = load_model('vgg16_1.h5')
```

```
In [ ]: import numpy as np
        from keras.preprocessing.image import ImageDataGenerator
        img_width, img_height = 432, 144


        # from tqdm import tqdm
        test_data_dir = "floyd_dataset/spectrograms_test"

        t_datagen = ImageDataGenerator()


        test_generator = t_datagen.flow_from_directory(
        test_data_dir,
        target_size = (img_height, img_width),
        class_mode = "categorical")


        #probabilities = model_final.predict_generator(test_generator)
```

```
In [ ]: validation_generator = t_datagen.flow_from_directory(
        "floyd_dataset/spectrograms_test",
        target_size = (img_height, img_width),
        class_mode = "categorical")

        model_final.evaluate_generator(validation_generator)
```

```
In [ ]: model_final.metrics_names
```

```
In [ ]: predicted_classes = np.argmax(probs,axis=1)
```

Матриця заплутувань:

```
In [ ]: from sklearn.metrics import confusion_matrix
        confusion_matrix(y_true, predicted_classes)
```

Нова модель з картами активації класів:

```
In [ ]: from keras import applications
        from keras.preprocessing.image import ImageDataGenerator
        from keras import optimizers
        from keras.models import Sequential, Model
        from keras.layers import Dropout, Flatten, Input, Dense, GlobalAveragePooling2D,Conv2D,LocallyCo
        from keras import backend as k
        from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping

        #img_width, img_height = 864, 288
        img_width, img_height = 432, 144

        train_data_dir = "floyd_dataset/spectrograms_train"
        validation_data_dir = "floyd_dataset/spectrograms_val"
        nb_train_samples = 5000#8729
        nb_validation_samples = 1000#2823
        batch_size = 8
        epochs = 20

        in_lay = Input((144,432,3))


        model = applications.VGG16(weights = "imagenet", include_top=False, input_shape = (img_height, i

        pt_depth = model.get_output_shape_at(0)[-1]

        pt_features = model(in_lay)

        from keras.layers import BatchNormalization
        bn_features = BatchNormalization()(pt_features)

        bn_features = BatchNormalization()(pt_features)


        # Freeze the layers which you don't want to train. Here I am freezing the first 5 layers.
        for layer in model.layers[:14]:
            layer.trainable = False


        #attention

        attn_layer = Conv2D(64, kernel_size = (1,1), padding = 'same', activation = 'relu')(bn_features)
        attn_layer = Conv2D(16, kernel_size = (1,1), padding = 'same', activation = 'relu')(attn_layer)
        attn_layer = LocallyConnected2D(1,
                                        kernel_size = (1,1),
                                        padding = 'valid',
                                        activation = 'sigmoid')(attn_layer)
        # fan it out to all of the channels
        up_c2_w = np.ones((1, 1, 1, pt_depth))
        up_c2 = Conv2D(pt_depth, kernel_size = (1,1), padding = 'same',
                       activation = 'linear', use_bias = False, weights = [up_c2_w])
```

```
up_c2.trainable = False
attn_layer = up_c2(attn_layer)

mask_features = multiply([attn_layer, bn_features])
gap_features = GlobalAveragePooling2D()(mask_features)
gap_mask = GlobalAveragePooling2D()(attn_layer)
# to account for missing values from the attention model
gap = Lambda(lambda x: x[0]/x[1], name = 'RescaleGAP')([gap_features, gap_mask])
gap_dr = Dropout(0.5)(gap)
dr_steps = Dropout(0.25)(Dense(1024, activation = 'elu')(gap_dr))

predictions = Dense(10, activation="softmax")(dr_steps)
```

Інформація про модель:

```
In [ ]: model_final = Model(input = in_lay, output = predictions)
        model_final.summary()
```

Тренування:

```
In [ ]: #reating the final model


        #ompile the model
        model_final.compile(loss = "categorical_crossentropy", optimizer = optimizers.SGD(lr=0.0001, mor

        # from keras.models import load_model
        # model_final = load_model('vgg16_1_2.h5')


        # Initiate the train and test generators with data Augumentation
        train_datagen = ImageDataGenerator(
        rescale = 1./255,
        horizontal_flip = True,
        fill_mode = "nearest",
        zoom_range = 0.3,
        width_shift_range = 0.3,
        height_shift_range=0.3)

        test_datagen = ImageDataGenerator(
        rescale = 1./255,
        horizontal_flip = True,
        fill_mode = "nearest",
        zoom_range = 0.3,
        width_shift_range = 0.3,
        height_shift_range=0.3)

        train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size = (img_height, img_width),
        batch_size = batch_size,
        class_mode = "categorical")

        validation_generator = test_datagen.flow_from_directory(
        validation_data_dir,
```

```
        target_size = (img_height, img_width),
        class_mode = "categorical")

        # Save the model according to the conditions
        checkpoint = ModelCheckpoint("vgg16_1_2.h5", monitor='val_acc', verbose=1, save_best_only=True,
        early = EarlyStopping(monitor='val_acc', min_delta=0, patience=10, verbose=1, mode='auto')


        # Train the model
        history = model_final.fit_generator(
        train_generator,
        samples_per_epoch = nb_train_samples,
        epochs = epochs,
        validation_data = validation_generator,
        nb_val_samples = nb_validation_samples,
        callbacks = [checkpoint, early])
```

In [ ]:
```
        import pandas as pd
        pd.DataFrame(history.history)
```

In [ ]:
```
        import pandas as pd
        pd.DataFrame(history.history).to_excel("table_attention.xls")
```

In [ ]:
```
        model_final.save("keras_model_VGG16_attention")
```

In [ ]:
```
        import matplotlib.pyplot as plt
        import matplotlib
        from matplotlib.pyplot import imshow
        from PIL import Image,ImageOps
        import keras.backend as K

        count_of_image = 50
        attn_func = K.function(inputs = [model.get_input_at(0), K.learning_phase()],
                   outputs = [attn_layer.get_output_at(0)]
                  )
        fig, m_axs = plt.subplots(20, 2, figsize = (8, 4*20))
        [c_ax.axis('off') for c_ax in m_axs.flatten()]
        #for c_idx, (img_ax, attn_ax) in zip(rand_idx, m_axs):
        cntr=0
        for img,y in validation_generator:
            #cur_img = c_idx
            #attn_img = attn_func([cur_img, 0])[0]
            #img_ax.imshow(cur_img, cmap = 'bone')
            #print(img)
            #print(img.shape)
            matplotlib.image.imsave(str(cntr)+"_class_"+str(np.argmax(y[0]))+"_predicted_"+str(np.argmax
            #print(img[0].reshape(1,144,432,3).shape)
            print(attn_func([img[0:1],0])[0][0].shape)
        #     new_width  = 432
        #     new_height = 144
        #     img_for_resizing = img_for_resizing.resize((new_width, new_height))
            matplotlib.image.imsave(str(cntr)+"_class_"+str(np.argmax(y[0]))+"_predicted_"+str(np.argmax
        #     attn_ax.imshow(attn_img[0, :, :, 0], cmap = 'viridis',
        #                    vmin = 0, vmax = 1,
        #                    interpolation = 'lanczos')
```

```
#     real_age = boneage_div*test_Y[c_idx]+boneage_mean
#     img_ax.set_title('Hand Image\nAge:%2.2fY' % (real_age/12))
#     pred_age = boneage_div*bone_age_model.predict(cur_img)+boneage_mean
#     attn_ax.set_title('Attention Map\nPred:%2.2fY' % (pred_age/12))
    cntr+=1
    if(cntr==count_of_image):
        break
#fig.savefig('attention_map.png', dpi = 300)
#plt.show()
```

Watch class activation maps:

In [ ]:
```
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
img_width, img_height = 432, 144
from tqdm import tqdm

# from tqdm import tqdm
test_data_dir = "floyd_dataset/spectrograms_val"

t_datagen = ImageDataGenerator()


test_generator = t_datagen.flow_from_directory(
test_data_dir,
target_size = (img_height, img_width),
class_mode = "categorical")
true_y = []
xs = []
predictions = []
c=0
for (x,y) in tqdm(test_generator):
    true_y.extend(np.argmax(y,axis=1))
    print(len(true_y))
    #print(true_y[-1])
    xs.append(x)
    c+=len(np.argmax(y,axis=1))
    if(c>=128):
        #print(np.argmax(model_final.predict(np.vstack(xs)),axis=1))
        predictions.extend(np.argmax(model_final.predict(np.vstack(xs)),axis=1))
        print(predictions[-1])
        xs = []
        c=0
        #print(len(true_y)==len(predictions))
    if(len(true_y)>=1024):
        break

#probabilities = model_final.predict_generator(test_generator)
```

In [ ]:
```
validation_generator = t_datagen.flow_from_directory(
"floyd_dataset/spectrograms_train",
target_size = (img_height, img_width),
class_mode = "categorical")

model_final.evaluate_generator(validation_generator)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(true_y, predictions)
```

In [ ]:
```
from sklearn.metrics import classification_report
print(classification_report(true_y, predictions))
```