Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики факультету інформатики

ОЦІНКА НЕВИЗНАЧЕНОСТІ У ЗАДАЧАХ КЛАСИФІКАЦІЇ

Текстова частина до курсової роботи
за спеціальністю 113 „Прикладна математика”

Керівник курсової роботи
к.ф.-м.н., ст.викл. Швай Н.О.
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2021 р.

Виконав студент
Миколайчик Я.А.
(прізвище та ініціали)
“11” травня 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри математики,
проф., д.ф.-м.н.
Олійник Б. В.
(підпис)
„____ ”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Миколайчику Я.А. факультету інформатики 1 курсу МП
ТЕМА: Оцінка невизначеності у задачах класифікації
Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Календарний план
Вступ
1 Задача класифікації. Нейронні мережі. Основні означення
2 Оцінка невизначеності у нейронних мережах
3 Аналіз методів оцінки невизначеності у контексті класифікації зображень
Висновки
Список літератури
Додатки

Дата видачі „___” _____ 2021 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема: Оцінка невизначеності у задачах класифікації

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	19.03.2021	
2.	Збір теоретичного матеріалу за темою нейронних мереж.	02.05.2021	
3.	Ознайомлення з поняттям невизначеності у задачах класифікації.	03.05.2021	
4.	Збір теоретичного матеріалу за темою невизначеності у задачах класифікації.	03.05.2021	
5.	Аналіз алгоритмів оцінки невизначеності.	04.05.2021	
6.	Програмування системи для аналізу тестування алгоритмів.	05.05.2021	
7.	Застосування запрограмованої системи для проведення експериментів.	07.05.2021	
8.	Опис результатів експериментів.	07.05.2021	
9.	Написання пояснювальної роботи.	10.05.2021	
10.	Створення слайдів для доповіді та написання доповіді.	11.05.2021	
11.	Здача курсової роботи.	11.05.2021	

Студент: Миколайчик Я.А.

Керівник: Швай Н.О.

“_____”_____

Зміст

Вступ

Разом зі швидким розвитком галузі програмної інженерії та збільшенням обсягу обчислювальних можливостей у останньому десятилітті, глибинне навчання стрімко розвивається. Сучасні інформаційні технології дають нам можливість проектувати нейронні мережі з мільйонами параметрів, які здатні навчатись виконувати завдання практично будь-якої складності.

Не зважаючи на це, для нас важливо розуміти межі можливостей наших моделей. Результат роботи моделі, навіть з високою ймовірністю, не повинен автоматично сприйматись як абсолютно точний. У класичній постановці задачі класифікації вихід алгоритму машинного навчання обмежений лише відповідним набором класів, на яких він був натренований. У реальних задачах на вхід моделі можуть подаватися дані, що не відповідають жодному з класів, але які модель якимось чином повинна опрацювати і дати певну адекватну оцінку. Для вирішення цієї проблеми були розроблені методи оцінки невизначеності.

Метою цього дослідження є визначити суть оцінки невизначеності у машинному навчанні, з’ясувати існуючі методи оцінки невизначеності, проаналізувати та порівняти їх на основі задачі класифікації графічних зображень за допомогою згорткових нейронних мереж.

Робота складається з трьох розділів.

У першому розділі буде розкрито базові поняття, необхідні для розуміння сутностей, пов’язаних з темою дослідження.

У другому розділі розглядається теоретична база завдання оцінки невизначеності: сфери її застосування, її типи, прояв у глибинних нейронних мережах, і методи оцінки.

Третій розділ присвячено програмному проектуванню деяких з методів оцінок невизначеності та їх порівнянню у межах задачі класифікації графічних зображень за допомогою згорткових нейронних мереж у середовищі Jupyter Notebook.

1 Задача класифікації. Нейронні мережі. Основні позначення

1.1 Навчання з вчителем. Задача класифікації

Навчання з вчителем – клас задач машинного навчання, у якому модель спостерігає набір розмічених вчителем пар вхід-вихід, і вивчає певну функцію відображення між значеннями входу та вихідними значеннями. [1]

При навчанні з вчителем ми маємо набір тренувальних даних D з N тренувальними точками $(x_n, y_n), n = 1, \dots, N$, де значення x_n – входи моделі (вектори значень ознак), а значення y_n – виходи моделі, що їм відповідають.

Ми припускаємо, що тренувальний набір D згенерований точками (x_n, y_n), які належать певному невідомому розподілу, є незалежними і ідентично розподіленими (independent and identically distributed, i.i.d). Ми хочемо, щоб модель апроксимувала функцію-предиктор $\hat{y}(x, \theta)$, де θ – параметри моделі, яка якомога точно буде передбачати вихід всіх можливих точок x, які ще не були спостережені.

Класифікація – тип задач з класу навчання з вчителем, у якому виходи y є дискретними значеннями, які можуть приймати скінченну кількість значень. Значення виходу y (label) для даного входу x позначає клас, якому належить x.

1.2 Функція втрат. Оптимізація. Градієнтний спуск

Якість моделі визначається значенням функції вартості або цільової функції (target function). Зазвичай значення цільової функції предиктора $\hat{y}$ для набору входів x з відомими відповідними значеннями y знаходиться за формулою [2]:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L\left(y^i, \hat{y}\left(x^i, \theta\right)\right), \tag{1.1}$$

де N – кількість вхідних векторів;

θ – апроксимовані параметри функції-предиктора;

x^i – вектор ознак і-го входу;

y^i – значення виходу, що відповідає входу x^i;

L – функція втрат.

Функція втрат показує величину розбіжності між передбаченням моделі, яка оцінюється, і справжнім значенням y^i.

Для оцінки моделі рекомендується обирати набір x, що не входив до тренувального набору. Такий набір називається валідаційним.

Тренування моделі (оптимізація) – це процес знаходження оптимальних параметрів θ, які мінімізують (максимізують) значення цільової функції. У машинному навчанні найчастіше використовуються оптимізаційні методи, що базуються на градієнтному спуску.

Градієнтний спуск [2] – метод оптимізації, що полягає у оновленні значень параметрів моделі у протилежному (при мінімізації) до градієнту цільової функції напрямку. Оновлення відбувається ітеративно, з використанням значення темпу навчання (learning rate) η, що визначає розмір кроку на кожній ітерації і забезпечує збіжність при достатньо малих значеннях. Градієнтний спуск складається з двох кроків, які повторюються, поки не буде досягнутою певна умова збіжності:

1) взяття частинної похідної цільової функції $J(\theta)$ відносно параметру θ_j, отримавши відповідний градієнт, де θ – множина параметрів, θ_j – параметр, що відповідає ознаці j, $j = 1, \dots, D$, D – кількість ознак;
2) оновлення кожного параметра θ_j у протилежному (при мінімізації) напрямку до градієнта з використанням значення темпу навчання за формулою:

$$\theta_j' = \theta_j + \eta * \frac{1}{N}\sum_{i=1}^{N}\left(y^i - \hat{y}_\theta\left(x^i\right)\right)x_j^i. \quad (1.2)$$

1.3 Нейронні мережі

Сучасні глибокі мережі надають потужне середовище для навчання з вчителем. Додавання більшої кількості шарів і нейронів всередині них дозволяє глибоким мережам описувати функції вищої складності. Більшість практичних задач, які полягають у знаходженні відображення з вхідних даних на вихідні, можуть бути розв'язані методами глибокого навчання при наявності достатньо комплексних моделей і достатньої кількості розмічених тренувальних даних.

1.3.1 Перцептрон. Багатошаровий перцептрон

Перцептрон [3] – лінійний бінарний класифікатор, який відображає вхідний вектор ознак **x** на бінарну множину виходів через використання ваг $\boldsymbol{w}$, зміщення (bias) b, і порогової функції (step function) $f(\boldsymbol{x})$ (функції активації), що може набувати двох значень відносно певного порогу. Він працює шляхом обрахунку скалярного добутка вектора ознак і вектора ваг, від значення якого залежить вихід функції $f(x)$.

Нехай множина класів – $\{0,1\}$, поріг – 0. Тоді вихід функції активації перцептрона матиме наступний вигляд:

$$f(\boldsymbol{x}) = \begin{cases} 1, & \text{якщо } \boldsymbol{w} * \boldsymbol{x} + b > 0 \\ 0, & \text{якщо } \boldsymbol{w} * \boldsymbol{x} + b \leq 0' \end{cases} \quad (1.3)$$

де $\boldsymbol{x}$ – вектор ознак;

$\boldsymbol{w}$ – вектор ваг;

b – зміщення;

$\boldsymbol{w} * \boldsymbol{x}$ – скалярний добуток векторів.

Схема роботи перцептрона представлена на рисунку 1.1.

Перцептрон вважається одношаровою нейронною мережею.

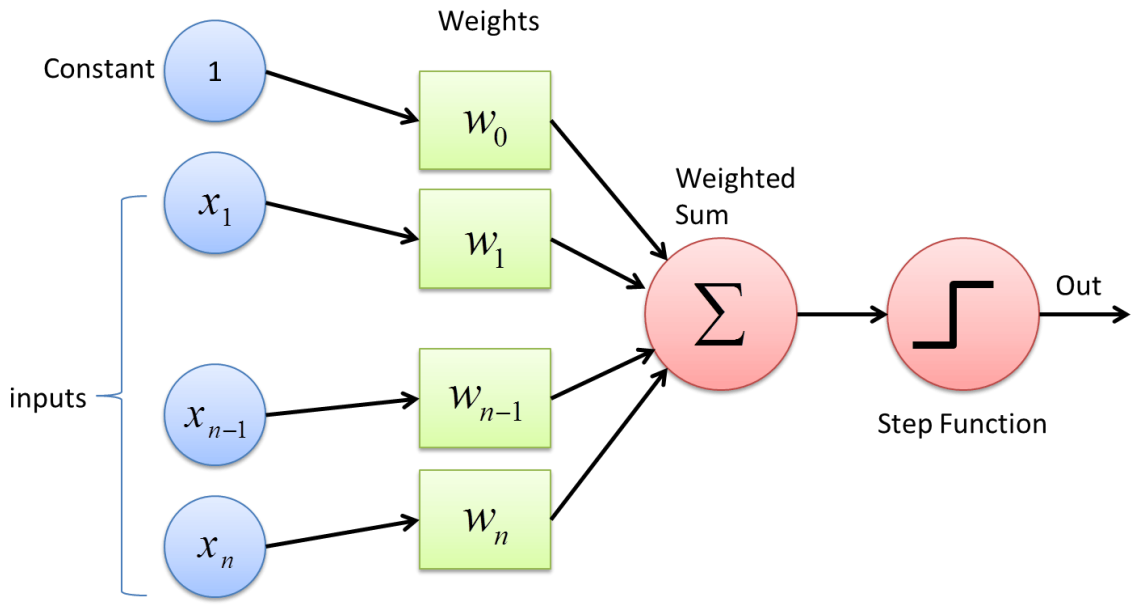

Рисунок 1.1 – схема роботи перцептрона [4]

Багатошаровим перцептроном (Multilayer Perceptron, MLP) [5] називають нейронну мережу, у якої між вхідним і вихідним шаром знаходиться принаймні ще один шар, які називають прихованими. Сьогодні під даним означенням зазвичай мають на увазі нейронні мережі загалом, тобто не обов'язково ті, які складаються з перцептронів у їх класичному розумінні з пороговою функцією активації, а такі, в яких функції активації можуть бути довільними, і в літературі ці поняття можуть використовуватись взаємозамінювано [6].

Кількість шарів, не рахуючи вхідного, називається глибиною мережі. Відповідно, мережі з великою кількістю шарів називають глибокими нейронними мережами.

Більшість прихованих шарів можуть бути описані як такі, що приймають вектор входу x, рахують афінне перетворення $z = \boldsymbol{W}^T\boldsymbol{x} + b$, де $\boldsymbol{W}$ – матриця ваг шару, а тоді поелементно застосовують функцію активації $g(z)$ і посилають результат на наступний шар (або на вихід моделі у випадку якщо шар є вихідним).

1.3.2 Функції активації

Як вже було сказано, функції активації обраховують вихідне значення нейронів мережі. Існує багато різних функцій активації і їх вибір залежить від того, в якому шарі мережі вони мають використовуватись, і від поставленої задачі, яку має вирішити мережа. Нижче розглянемо такі функції активації як ReLU і Softmax.

1.3.2.1 ReLU

Функція активації ReLU [6] має наступний вигляд:

$$\mathrm{g(z)} = \max\{0, z\}. \tag{1.4}$$

Вона є дуже схожою на лінійну, окрім того, що для від'ємних вхідних значень вона повертає нуль, тому їх легко і дуже швидко рахувати, а також значення похідних залишаються високими, коли нейрон є активним, що допомагає при визначенні напрямку градієнту під час навчання.

1.3.2.2 Softmax

Функція активації softmax [6] має наступний вигляд:

$$softmax(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}, \tag{1.5}$$

де exp – функція експоненти, а індекси i, j відповідають за елементи вектора афанних перетворень z.

Функція softmax часто використовується у вихідних шарах нейронних мереж-класифікаторів, де її результатом є нормоване представлення ймовірнісного розподілу дискретної величини по класам.

1.3.3 Узагальнення. Перенавчання. Регуляризація

Однією з найголовніших проблем машинного навчання є задача створення моделі, яка добре працюватиме не лише на навчальних даних, але і добре передбачатиме результат для нових.

Вміння моделі правильно описувати не лише на тренувальні дані, а і на тестувальні, називається узагальненням (generalization). Явище, коли модель не може адекватно описати тренувальні дані через її недостатню складність, називається недонавчаннням (underfitting). Явище, коли модель добре описує лише тестувальні дані, називається перенавчанням (overfitting).

Перенавчання виникає тоді, коли параметри моделі починають описувати занадто специфічні для тренувальних даних ознаки, замість того, щоб охоплювати загальні ознаки, притаманні всім даним, в тому числі поза тренувальним набором.

Схематичне зображення явищ недонавчання і перенавчання зображені на рисунку 1.6:

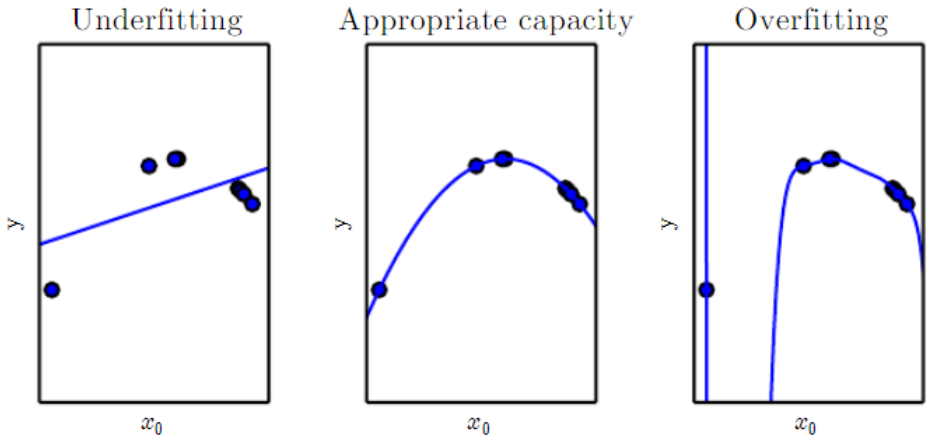

Рисунок 1.2 – Схематичне зображення недонавчання, оптимального навчання і перенавчання моделі [6]

Під час тренування, ми намагаємось зменшити тренувальну похибку (training error) за допомогою методів оптимізації. Головна ж відмінність машинного навчання від звичайної оптимізації полягає у тому, що ми хочемо також досягти якомога низької узагальнювальної похибки (generalization error), яку також називають тестувальною похибкою (test error), і яка є очікуваним значенням похибки на нових вхідних даних [6].

Розроблено багато методів, спрямованих на те, щоб зменшити тестувальну похибку. Деякі з них, такі як Dropout, Batch Normalization і ансамблі моделей, будуть розглянуті у розділі 2.6. Всі такі методи загалом називаються методами регуляризації.

1.4 Згорткові нейронні мережі

Згорткова нейронна мережа (Convolutional Neural Network, CNN) [6] – спеціалізований вид нейронних мереж для обробки даних, що мають сітчасту структуру. Прикладом можуть бути дані про часові ряди, які можна розглядати як одномірну сітку, яка бере зразки через рівні проміжки часу, та графічні зображення, які можна сприймати як двовимірну сітку пікселів. Згорткові мережі надзвичайно успішні у практичному застосуванні. Назва цих мереж походить від назви спеціалізованої лінійної операції операції, яка в них використовується, - згортки. Згорткові мережі - це нейронні мережі, які використовують згортку замість звичайного матричного множення принаймні в одному зі своїх шарів.

1.4.1 Згортка

Згорткою [6] двох функцій дійсної змінної f і g називають математичну операцію вигляду:

$$(x * w)(t) = \int x(a)w(t-a)da. \tag{1.6}$$

У термінології згорткових нейронних мереж, перший аргумент згортки часто називають входом (input), а другий аргумент – ядром (kernel). Результат згортки називають feature map.

При роботі з комп’ютером у межах згорткових нейронних мереж, дані є дискретними, і в задачах машинного навчання вхід зазвичай представлений у вигляді матриці даних, а ядро – у вигляді матриці параметрів, які вивчає алгоритм. Їх також часто називають тензорами. Це означає, що на практиці, операція згортки, враховуючи також те, що вона є комутативною, може бути представлена у вигляді операцій сум:

$$(K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{1.7}$$

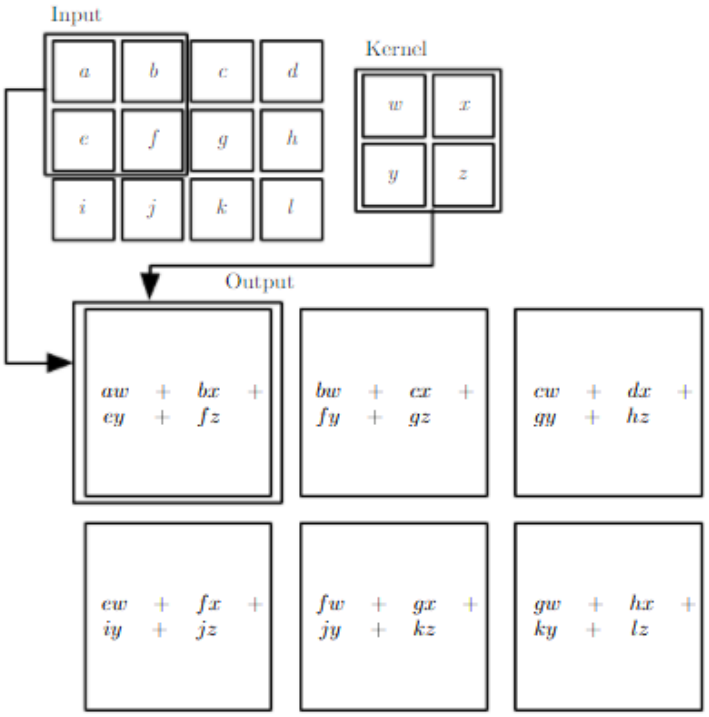

Рисунок 1.3 – Приклад згортки двох двовимірних матриць

2 Оцінка невизначеності у нейронних мережах

2.1 Невизначеність моделі

Моделі згорткових нейронних мереж особливо успішно застосовуються у рішенні різноманітних задач, пов’язаних з графічними зображеннями, при достатній кількості доступних даних і якісному тренуванні можуть на етапі тестування точно передбачати результат для нових даних. Класифікатори навчаються передбачати саме ті класи зображень, на яких вони були натреновані. Наприклад, модель була натренована на зображеннях котів та собак, і ми очікуємо, що вона буде добре розрізняти їх зображення. Але вона ніколи не бачила зображень інших тварин. Що трапиться, якщо ми захочемо, щоб модель передбачила клас зображення іншої тварини? Це є прикладом тестувальних даних поза розподілом (out of distribution test data) [7]. Як варіант, ми можемо спроектувати модель таким чином, щоб вона повертала не тільки передбачення (яке вона змушена повертати у будь-якому випадку), але також і певний показник невизначеності, який набував би високих значень для тестувальних даних поза розподілом, і показував би низьку впевненість моделі у передбаченні.

Інформація про невизначеність буває дуже важливою. Знання про недостатню або надмірну впевненість (в залежності від показника невизначеності) моделі може допомогти отримати кращий результат при роботі з нею. Знаючи, що тестувальні дані є далекими від тренувальних, ми могли б розширити нашу модель або внести виправлення до тестувальних даних. Але, найважливіший напрям, у якому моделі з невизначеністю можуть бути використані, - це задачі прийняття рішень, які певним чином впливають на життя людини. Деякі області застосування оцінки невизначеності будуть розглянуті далі.

2.2 Практичні застосування оцінки невизначеності

2.2.1 Активне навчання

У машинному навчанні, отримання розмічених даних для їх використання у тренуванні і тестуванні буває дуже трудозатратним, дорогим і довгим процесом.

Активне навчання (active learning) – це методологія систем машинного навчання, за якою модель може під час послідовних етапів тренування сама обирати дані для тренування, які вона вважає найбільш корисними, і робити запити про їх розмітку вчителю [8]. Перевагою активного навчання є те, що його використання часто призводить до значного зменшення кількості даних, що потребуються для тренування системи, і, як наслідок, зменшення витрат на тренування.

Система активного навчання робить запити до вчителя за допомогою функції здобуття (acquisition function). Такі функції можуть використовувати принципи оцінки невизначеності. Наприклад, коли модель тестує свою продуктивність на нерозмічених даних, вона може оцінити, яка частина цих даних належить тренувальному розподілу, а яка є ще повністю новою і не дослідженою. Дані, в яких модель найбільш невизначена, є найбільш корисними для тренування, тому вони обираються нею для запиту до вчителя про розмітку.

2.2.2 Автономний транспорт

Методи оцінки невизначеності також можуть застосовуватись для моделей автономного керування транспортом, наприклад, автомобілями [9]. У автономному керуванні, тренувальний набір може складатись з даних, в тому числі зображень, зафіксованих сенсорами автомобіля у різних дорожніх умовах. Однак, під час тестування у реальних умовах, може виникати багато унікальних ситуацій, з якими модель ще не стикалась і їй не відомий

найкращий план дій, що є небезпечним для пасажирів. Якщо модель матиме в собі певний метод оцінки невизначеності для дорожніх умов, це могло б допомогти їй уникнути потенційно небезпечних дій у невідомих їй ситуаціях, а замість них, наприклад, передати керування водієві.

2.3 Типи невизначеності

2.3.1 Алеаторна невизначеність

Алеаторна невизначеність [7] – це така, яка виникає через шум у даних (наприклад, неточність у вимірюваннях датчика). Певний шум притаманний більшості даних у машинному навчанні і не може бути зменшеним шляхом збільшення кількості спостережень.

2.3.2 Епістемічна невизначеність

Епістемічна невизначеність (невизначеність моделі) [7] – це така, яка виникає внаслідок нестачі знань про справжню модель досліджуваного процесу. До неї відноситься параметрична невизначеність (невизначеність у виборі параметрів моделі, що найкраще описують дані), а також структурна невизначеність (невизначеність у виборі найкращої структури моделі). На відміну від алеаторної, епістемічна невизначеність може бути зменшена шляхом проведення додаткових спостережень або підборі кращої архітектури моделі, що краще б змогла описати дані.

2.3.3 Передбачувальна невизначеність

Передбачувальна невизначеність – це загальна невизначеність передбачення моделі, що складається з алеаторної та епістемічної, і є показником впевненості у передбаченнях моделі.

2.4 Невизначеність моделі глибинного навчання

Визначивши, що мати показник невизначеності моделі буває досить важливо, варто зазначити, що більшість моделей глибинного навчання його не обраховують. Моделі класифікації з вихідним шаром softmax дають на виході вектор ймовірності належності вхідних даних до певного класу, але його не варто плутати з впевненістю чи невизначеністю моделі. Модель може бути доволі невизначеною у своєму передбаченні навіть з високим значенням softmax ймовірності [7].

Хоча сучасні моделі глибинного навчання не рахують невизначеність моделі, вони близькі до сімейства ймовірнісних моделей, що визначають розподіли ймовірностей на функціях: Гаусового процесу. Нейронні мережі, що мають в основі ймовірнісні розподіли на параметрах називаються Баєсовими нейронними мережами, і невизначеність може бути отримана з таких моделей. Але такими моделями буває досить важко працювати, тому вони є не дуже практичними і не набули широкого використання у області глибинного навчання.

У праці [7] було показано, що глибинна модель, що використовує стохастичні методи регуляризації (підрозділ 2.5) апроксимує Гаусовий процес, і з такої моделі можна отримати апроксимоване значення невизначеності. Запропонований ним підхід є дуже практичним у застосуванні і може бути використаним у складних моделях і моделях, що вимагають великої кількості даних.

2.5 Стохастичні методи регуляризації

Стохастичні методи регуляризації – це методи, що регуляризують глибинні нейронні мережі через впровадження в модель стохастичного шуму. Найбільш розповсюдженими методами регуляризації в глибинному навчанні є dropout (пункт 2.6.1) та batch normalization (пункт 2.6.2).

2.6 Методи оцінки невизначеності

2.6.1 Метод Monte Carlo Dropout (MCDO)

Dropout [10] – це одна з технік регуляризації глибоких нейронних мереж. Цей метод полягає у тимчасовому «викиданні» нейронів мережі під час проходження даних через неї. Під викиданням нейрону мається на увазі його видалення з нейронної мережі, разом з його вхідними і вихідними зв’язками. Вибір нейронів для виключення на кожному кроці проходження по мережі відбувається випадково. Приклад dropout зображений на рисунку 2.1:

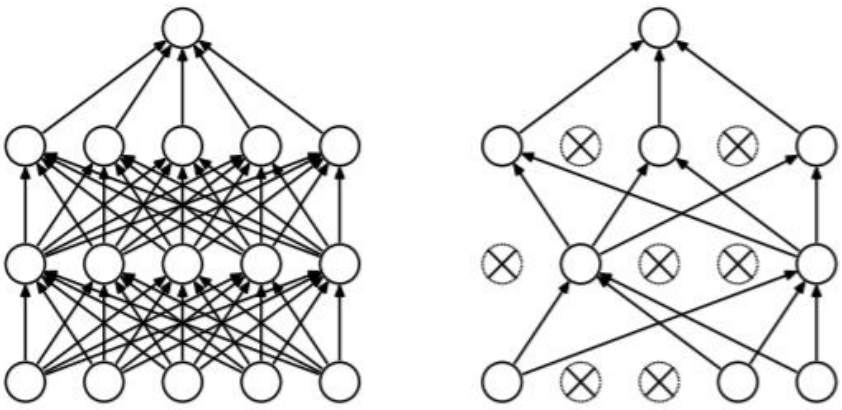

Рисунок 2.1 – Приклад dropout. Зліва зображена звичайна нейронна мережа з двома прихованими шарами. Справа зображена та сама мережа після операцій dropout (закреслені нейрони були викинуті)

В силу своєї стохастичності, dropout може бути використаний для оцінки невизначеності глибинної нейронної мережі шляхом виконання декількох стохастичних проходжень по мережі і передбачень для даних, і обрахунку середнього значення та середньоквадратичного відхилення цих передбачень [7]. Такий процес передбачення називається Monte Carlo Dropout і може бути застосований до будь-якої мережі з dropout шарами.

2.6.2 Метод Monte Carlo Batch Normalization (MCBN)

Batch normalization [11] – це операція над нейронами, призначена для нормалізації розподілу вхідних до нейронів значень, що має регуляризаційні властивості і може використовуватись замість dropout. Алгоритм трансформації кожного вхідного пакету (batch), де γ і β – параметри масштабування і зміщення, що вивчаються під час тренування, представлені на рисунку 2.2:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Рисунок 2.2 – алгоритм batch normalization [11]

Схожим до Monte Carlo Dropout чином, ми також можемо оцінити невизначеність передбачень моделі завдяки стохастичності batch normalization, використовуючи метод під назвою Monte Carlo Batch Normalizaion [12].

2.6.3 Ансамблевий метод

Ансамбль моделей [13] – метод машинного навчання, що полягає у об’єднанні роботи декількох моделей, використання яких разом дає кращий результат, ніж поодинці. Об’єднання передбачень класифікаторів у ансамблі зазвичай відбувається шляхом голосування (обрання класу, який передбачили

більшість класифікаторів, як результат) або усереднення значень передбачених ймовірностей.

Було показано, [14] що використання ансамблевих методів з рандомізованим підходом дає змогу також оцінювати невизначеність, використовуючи середнє і середньоквадратичне відхилення передбачень моделей ансамблю.

2.6.4 Обрахунок значення оцінки невизначеності

Як ми вже з'ясували, методи Monte Carlo Dropout, Monte Carlo Batch Normalization та ансамблевий метод для кожного вхідного прикладу повертають декілька передбачень, кількість яких у перших двох дорівнює кількості повторних проходжень даних по моделі, а у третього – кількості моделей у ансамблі. Передбачений клас може бути отриманий, взявши максимальне значення з усереднених ймовірностей класів. Оцінка невизначеності може бути отримана за формулою знаходження середньоквадратичного відхилення:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}, \tag{2.1}$$

де x_i – окремі softmax передбачення;

μ – середнє значення всіх передбачень;

N – кількість передбачень для поточного приладу x.

3 Аналіз методів оцінки невизначеності у контексті класифікації зображень

3.1 Набір даних

Для тестування методів оцінки невизначеності було обрано набір даних CIFAR-10 [15]. Він складається з 60000 кольорових зображень розміром 32x32 пікселів, що належать до 10 класів, з 6000 зображеннями у кожному класі. Всього у наборі 50000 тренувальних зображень і 10000 тестувальних. Класи є повністю взаємно виключеними, без перетинів. Приклади зображень та класів з CIFAR-10 наведені на рисунку 3.1:

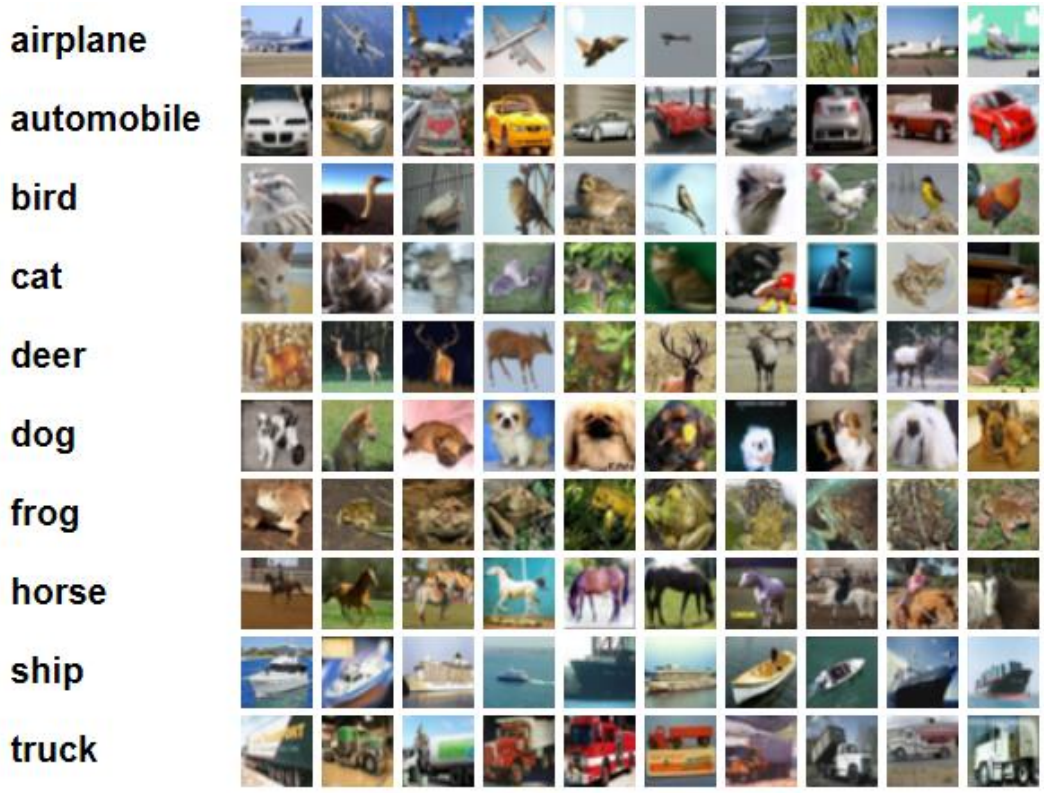

Рисунок 3.1 – Приклади зображень десяти класів набору даних CIFAR-10

Зображення класу «Truck» були відділені від набору даних для використання у якості даних поза розподілом.

3.2 Тестувальні нейронні мережі

Ми реалізуємо і порівняємо два представлених методи – Monte Carlo Dropout та ансамбль моделей. Monte Carlo Dropout буде реалізовано на основі згорткової нейронної мережі VGG16 [16], яка була обрана через її відносну простоту у архітектурі, а також відсутність попередньо вбудованих шарів регуляризації. Вона складається з тринадцяти згорткових шарів, п’яти об’єднувальних (MaxPooling) шарів, та трьох щільно з’єднаних (dense) шарів. Кожен згортковий шар, а також щільні шари, крім останнього, використовують функцію активації ReLU (підпункт 1.1.2.2). Вихідний (output) шар використовує функцію активації softmax (підпункт 1.1.2.3) для класифікації. Структура мережі VGG16 зображена на рисунку 3.2:

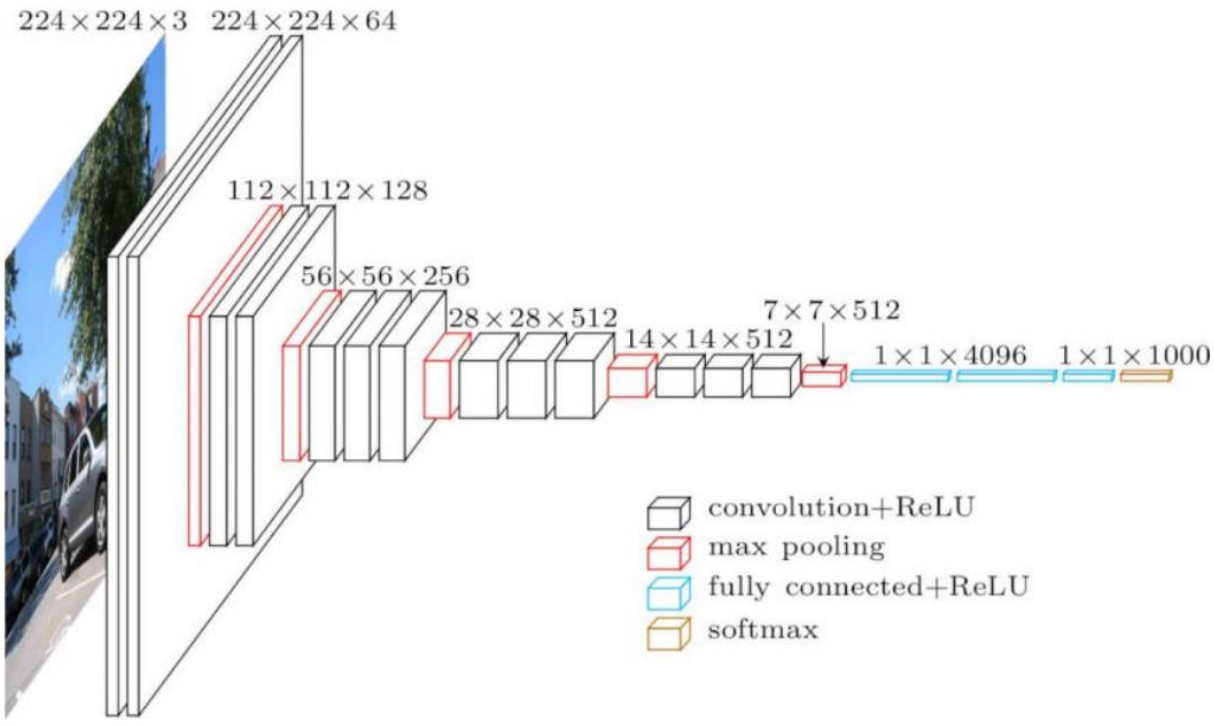

Рисунок 3.2 – Структура мережі VGG16

Для реалізації ансамблевого методу були довільним чином обрані такі десять моделей: ResNet50, ResNet101, ResNet152, ResNet50V2, ResNet101V2, ResNet152V2, DenseNet121, DenseNet169, DenseNet201, VGG16.

3.3 Реалізація і тестування

Тестувальна програма була виконана на мові Python у середовищі Jupyter Notebook за допомогою бібліотек машинного навчання Keras та TensorFlow.

Для всіх моделей під час тренування був використаний метод оптимізації Adam з параметром learning rate 0.0001.

Кількість зображень у тренувальному наборі даних становив 36000. Клас “Truck” був попередньо вилучений з тренувальних і тестувальних даних і його тестувальний набір out-of-distribution даних з тисячі зображень був використаний для оцінки невизначеності. Для тестування точності передбачення інших дев’яти класів також була використана тисяча тестувальних зображень.

3.3.1 Monte Carlo Dropout

Для реалізації методу Monte Carlo Dropout, було використано модель VGG16 з ініціалізованими вагами imagenet на всіх шарах, крім останніх трьох щільних шарів. Щоб досягти стохастичного процесу і мати змогу оцінити невизначеність, було додано шари dropout після двох передостанніх щільних шарів з drop rate 0.5, які були введені у режим роботи під час не тільки навчання, але і передбачення.

3.3.2 Ансамблевий метод

Для реалізації ансамблевого методу було використано десять моделей (див. підрозділ 3.2). Кожна модель була ініціалізована з вагами imagenet і без верхніх шарів. Верхні шари, а саме два щільні шари з 4086 нейронами, разом з шарами dropout з drop rate 0.5 для уникнення перенавчання, і softmax класифікатор, були додані вручну до всіх моделей. Разом моделі утворювали

масив, у якому послідовно проходили тренування і робили передбачення однакових вхідних даних.

3.4 Порівняння та оцінка предаставлених методів

3.4.1 Швидкість тренування

Тренування моделі Monte Carlo Dropout відбувалось з використанням механізму early stopping. Всього модель тренувалась сім епох з найкращим результатом валідаційної похибки на третій епосі, стан якої був відновлений через early stopping. Тренування зайняло 18 хвилин, 7 секунд (рисунок 3.3) і в середньому займало 2 хвилини, 35 секунд на епоху.

```
Epoch 00008: early stopping
MCDO training (36000 images + 9000 validation) time, in seconds: 1087.4231321811676
```

Рисунок 3.3 – Час тренування моделі MCDO

Тренування ансамблю відбувалось у 5 епох для кожної моделі. Всього воно зайняло 1 годину, 26 хвилин і 41 секунду (рисунок 3.4), і це приблизно 17 хвилин, 20 секунд на одну епоху всіх моделей разом.

```
Ensemble (10 models) training (36000 images + 9000 validation) time, in seconds: 5201.29
1935682297
```

Рисунок 3.4 – Час тренування ансамблю моделей

3.4.2 Точність передбачення тестових даних

Обидві моделі були протестовані на однаковому наборі з тисячі тестових зображень.

Модель MCDO показала точність у 83.4%, а ансамблева модель – у 88.5% (рисунок 3.5).

```
Dropout model accuracy: 0.834
Ensemble accuracy: 0.885
```

Рисунок 3.5 – Точність моделей на тестувальних даних

3.4.3 Якість оцінки невизначеності

Для перевірки якості оцінки невизначеності, моделі передбачали класи тисячі зображень вантажівок, які не належали тренувальному розподілу.

Модель MCDO робила передбачення сто ітерацій, а ансамбль – одну для кожної з моделей.

Результати для MCDO і для ансамблю, і порівняння невизначеності передбачень між даними розподілу і даними поза розподілом, зображені на рисунках 3.6 і 3.7 відповідно.

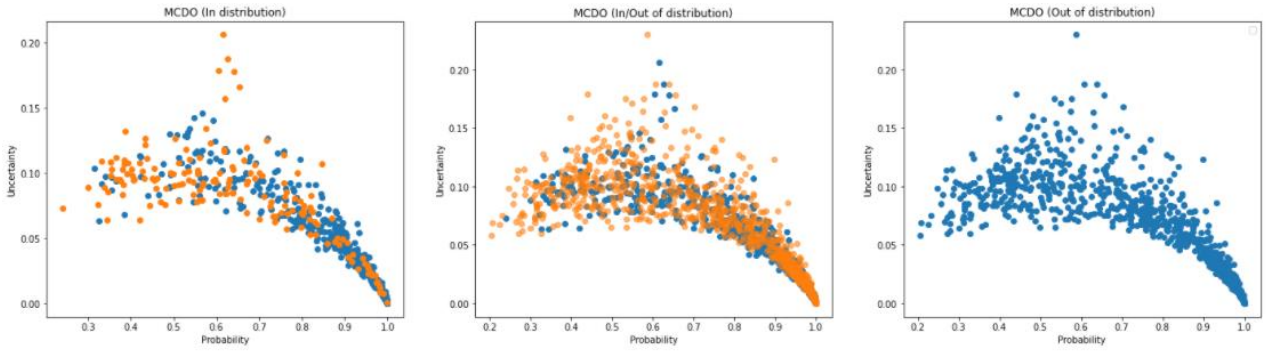

Рисунок 3.6 – Оцінки невизначеності MCBO. На зображенні ліворуч сині точки – правильно передбачені зображення, оранжеві – неправильно. На центральному зображенні сині точки – передбачення для зображень, що належать розподілу, оранжеві – для зображень, що не належать.

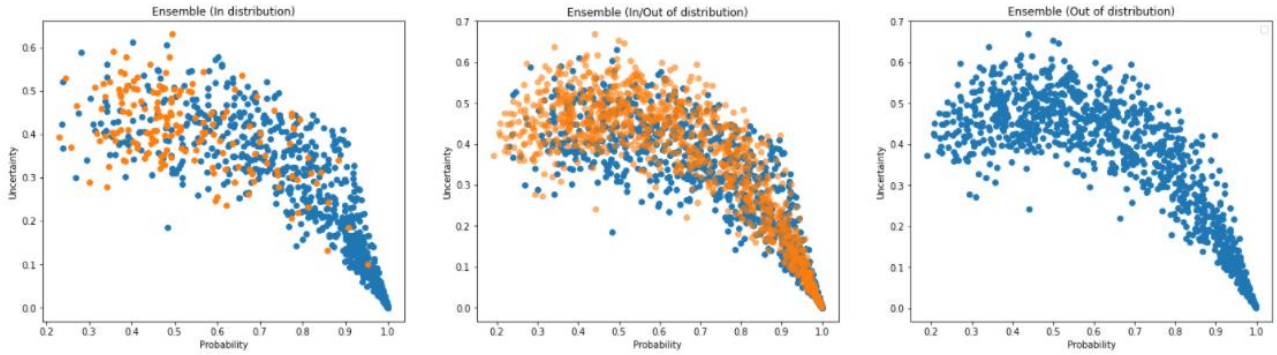

Рисунок 3.7 – Оцінки невизначеності ансамблю моделей. На зображенні ліворуч сині точки – правильно передбачені зображення, оранжеві – неправильно. На центральному зображенні сині точки – передбачення для зображень, що належать розподілу, оранжеві – для зображень, що не належать.

Середні значення невизначеностей, а також їх різниця для даних у моделі і поза моделлю наведені на рисунку 3.8:

```
MCDO mean uncertainty (In distribution): 0.036326827021021 29
MCDO mean uncertainty (Out of distribution): 0.058123181022138724
MCDO mean uncertainty difference: 0.021796354001117436
Ensemble mean uncertainty (In distribution): 0.2532975143351969
Ensemble mean uncertainty (Out of distribution): 0.35780024933111043
Ensemble mean uncertainty difference: 0.1045027349959135
```

Рисунок 3.8 – Середні показники невизначеності і їх різниці

3.4.4 Висновки про роботу методів

Ми дослідили такі показники моделей, як швидкість тренування, точність передбачення і якість оцінки невизначеностей.

Ансамблева модель значно краще себе проявила у плані якості оцінки невизначеності, а також швидкості передбачень. Дивлячись на лівий графік на рисунку 3.7, можемо бачити, що у більшості неправильних передбаченнях модель була досить невпевненою, а також можемо спостерігати певну

кореляцію між показником ймовірності передбачуваного класу та невизначеністю. Також на середньому графіку бачимо, що для даних поза розподілом модель є більш невизначеною.

Все це також виконується і для методу Monte Carlo Dropout, але набагато менш виражено і майже непомітно на графіках, проте дані про середні значення невизначеностей на різних даних (рисунок 3.8) підтверджують це. Також метод Monte Carlo Dropout у конкретно цьому варіанті постановки задачі має сильну перевагу у швидкості навчання.

Висновок

Отже, у ході даної роботи було переглянуто основні поняття машинного навчання, та глибинного навчання. Ми з’ясували області застосування оцінки невизначеності моделей машинного навчання і її важливість у деяких сферах людського життя, дослідили суть невизначеності у задачах глибинного навчання, її джерела, прояви, способи подолання. Також ми розглянули існуючі методи оцінки невизначеності, такі як Monte Carlo Dropout, Monte Carlo Batch Normalization та ансамблевий метод, а також програмно реалізували два з них. Ми дослідили ці методи у рамках задачі класифікації графічних зображень та порівняли їх за швидкістю тренування, швидкістю передбачення та якістю оцінки невизначеності на даних поза розподілом.

Список літератури

1. Osvaldo S. A Very Brief Introduction to Machine Learning With Applications to Communication Systems / Simeone Osvaldo // IEEE Transactions on Cognitive Communications and Networking / Simeone Osvaldo., 2018. – (IEEE). – С. 648–664.
2. A Survey of Optimization Methods from a Machine Learning Perspective / S.Sun, Z. Cao, H. Zhu, J. Zhao. // CoRR. – 2019.
3. What is a Perceptron? [Електронний ресурс] // DeepAI – Режим доступу до ресурсу: https://deepai.org/machine-learning-glossary-and-terms/perceptron.
4. What the Hell is Perceptron? [Електронний ресурс] // Towards Data Science. – 2017. – Режим доступу до ресурсу: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.
5. What are multilayer perceptrons? [Електронний ресурс] // DeepAI – Режим доступу до ресурсу: https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron.
6. Goodfellow I. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville., 2016. – (MIT Press).
7. Gal Y. Uncertainty in Deep Learning : дис. докт. техн. наук / Gal Yarin, 2016.
8. Gal Y. Deep Bayesian Active Learning with Image Data / Y. Gal, R. Islam, Z. Ghahramani. // CoRR. – 2017.
9. Michelmore R. Evaluating Uncertainty Quantification in End-to-End Autonomous Driving Control / R. Michelmore, M. Kwiatkowska, Y. Gal. // CoRR. – 2018.
10. Dropout: A Simple Way to Prevent Neural Networks from Overfitting / [N. Srivastava, G. Hinton, A. Krizhevsky та ін.]. // Journal of Machine Learning Research. – 2014. – №15. – С. 1929–1958.

11. Ioffe S. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift / S. Ioffe, C. Szegedy. // CoRR. – 2015.
12. Teye M. Bayesian Uncertainty Estimation for Batch Normalized Deep Networks / M. Teye, H. Azizpour, K. Smith. // arXiv e-prints. – 2018.
13. Rokach L. Ensemble-based classifiers / Lior Rokach // Artif. Intell. Rev. / Lior Rokach., 2010. – С. 1–39.
14. Lakshminarayanan B. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles / B. Lakshminarayanan, A. Pritzel, C. Blundell. // arXiv e-prints. – 2016.
15. Krizhevsky A. The CIFAR-10 dataset [Електронний ресурс] / A. Krizhevsky, V. Nair, G. Hinton – Режим доступу до ресурсу: https://www.cs.toronto.edu/~kriz/cifar.html.
16. VGG16 – Convolutional Network for Classification and Detection [Електронний ресурс] // neurohive.io. – 2018. – Режим доступу до ресурсу: https://neurohive.io/en/popular-networks/vgg16/.

Додаток А

(обов'язковий)

Лістинг програмного коду

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pickle, os, time
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras import models, layers
from tensorflow.keras.applications import ResNet50, ResNet101, ResNet152,
ResNet50V2, ResNet101V2, ResNet152V2
from tensorflow.keras.applications import DenseNet121, DenseNet169,
DenseNet201, VGG16
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping


# In[2]:


np.random.seed(1)
tf.random.set_seed(1)


# In[3]:


def unpickle(file):
    with open(file, 'rb') as fo:
        d = pickle.load(fo, encoding='latin1')
    x = d['data'].reshape(10000, 3, 32, 32).transpose(0,2,3,1) / 255.
    y = np.asarray(d['labels'])
    return x, y

#---read cifar-10 dataset---#
train_x = []
train_y = []
filenames =
['data_batch_1','data_batch_2','data_batch_3','data_batch_4','data_batch_
5']
for filename in filenames:
    x, y = unpickle(os.path.join('../input/cifar10-python/cifar-10-
batches-py', filename))
    train_x.append(x)
    train_y.append(y)
train_x = np.concatenate(train_x)
```

```
train_y = np.concatenate(train_y)
test_x, test_y = unpickle('../input/cifar10-python/cifar-10-batches-
py/test_batch')

classes = None
with open('../input/cifar10-python/cifar-10-batches-py/batches.meta',
'rb') as fo:
    classes = dict(enumerate(pickle.load(fo,
encoding='latin1')['label_names']))
#--------------------------#


# In[4]:


classes


# In[5]:


# make 10th class out of distribution data
ood_ids_train = np.where(train_y == 9)
ood_ids_test = np.where(test_y == 9)

ood_train_x = train_x[ood_ids_train]
ood_test_x = test_x[ood_ids_test]

train_x = train_x[np.where(train_y != 9)]
test_x = test_x[np.where(test_y != 9)]

train_y = train_y[np.where(train_y != 9)]
test_y = test_y[np.where(test_y != 9)]

print(train_x.shape)
print(test_x.shape)

print(ood_train_x.shape)
print(ood_test_x.shape)


# In[6]:


train_x, valid_x, train_y, valid_y = train_test_split(train_x, train_y,
test_size=0.2, random_state=1)


# In[7]:


print(train_x.shape)
print(valid_x.shape)
print(test_x.shape)


# In[8]:
```

```
# Parameters
N_CLASSES = len(classes) - 1 # excluded 1 class as OOD
BATCH_SIZE = 32
EPOCHS = 25
TARGET_SIZE = 32


# In[9]:


def dropout_layer_factory(rate):
    return layers.Dropout(rate=rate)

def insert_intermediate_layers(model, layer_ids, rates, factory):
    layers = [l for l in model.layers]
    x = layers[0].output
    j = 0
    for i in range(1, len(layers)):
        if (i in layer_ids):
            new_layer = factory(rates[j])
            j+=1
            x = new_layer(x, training=True) # makes layer apply during
inference
        x = layers[i](x)

    new_model = models.Model(layers[0].input, x)
    return new_model

def create_model_dropout():
    base = VGG16(
        include_top = False,
        weights = 'imagenet',
        input_shape = (TARGET_SIZE, TARGET_SIZE, 3),
    )

    model = base.output

    # restoring top layers architecture of VGG16
    model = layers.Flatten()(model)
    model = layers.Dense(units = 4096, activation = 'relu')(model)
    model = layers.Dense(units = 4096, activation = 'relu')(model)

    model = layers.Dense(N_CLASSES, activation = "softmax")(model)
    model = models.Model(base.input, model)

    # create dropout layers after each of the two dense layers at the top
of the network
    model = insert_intermediate_layers(model, layer_ids=[21, 22],
rates=[0.5, 0.5],
                                       factory=dropout_layer_factory)
    model.compile(
        optimizer = Adam(lr = 0.0001),
        loss = "sparse_categorical_crossentropy",
        metrics = ['acc'])
```

```
    return model


# In[10]:


ensemble_names=['ResNet50','ResNet101','ResNet152','ResNet50V2','ResNet10
1V2',

'ResNet152V2','DenseNet121','DenseNet169','DenseNet201','VGG16']
def create_model_ensemble():
    base_models = [ResNet50(include_top=False, weights='imagenet',
input_shape=(32,32,3)),
                   ResNet101(include_top=False, weights='imagenet',
input_shape=(32,32,3)),
                   ResNet152(include_top=False, weights='imagenet',
input_shape=(32,32,3)),
                   ResNet50V2(include_top=False, weights='imagenet',
input_shape=(32,32,3)),
                   ResNet101V2(include_top=False, weights='imagenet',
input_shape=(32,32,3)),
                   ResNet152V2(include_top=False, weights='imagenet',
input_shape=(32,32,3)),
                   DenseNet121(include_top=False, weights='imagenet',
input_shape=(32,32,3)),
                   DenseNet169(include_top=False, weights='imagenet',
input_shape=(32,32,3)),
                   DenseNet201(include_top=False, weights='imagenet',
input_shape=(32,32,3)),
                   VGG16(include_top=False, weights='imagenet',
input_shape=(32,32,3)),
                  ]

    for i, base in enumerate(base_models):
        model = base.output

        model = layers.Flatten()(model)
        model = layers.Dense(units = 4096, activation = 'relu')(model)
        model = layers.Dropout(rate=0.5)(model)
        model = layers.Dense(units = 4096, activation = 'relu')(model)
        model = layers.Dropout(rate=0.5)(model)

        model = layers.Dense(N_CLASSES, activation = "softmax")(model)
        model = models.Model(base.input, model)

        model.compile(
            optimizer = Adam(lr = 0.0001),
            loss = "sparse_categorical_crossentropy",
            metrics = ['acc'])

        base_models[i] = model

    return base_models


# In[11]:
```

```
# mdl =
models.load_model('../input/uncertaintyestimation/VGG16_Dropout.h5')
# mdl_ensemble = []
# for name in ensemble_names:
#
mdl_ensemble.append(models.load_model('./input/uncertaintyestimation/{}_E
nsemble.h5'.format(name)))


# In[12]:


mdl = create_model_dropout()
mdl_ensemble = create_model_ensemble()
mdl.summary()


# In[13]:


early_stopping = EarlyStopping(
    monitor = 'val_loss', min_delta = 0.01, patience = 5,
    verbose = 1, mode = 'min', restore_best_weights = True)


# In[14]:


# TRAINING
print('Dropout model training.')
start_time = time.time()
mdl.fit(x=train_x, y=train_y, batch_size=BATCH_SIZE, epochs=EPOCHS,
verbose=1,
                  callbacks=[early_stopping], validation_data=(valid_x,
valid_y),
                  workers = 4)
end_time = time.time()
print('MCDO training (36000 images + 9000 validation) time, in seconds:',
      end_time-start_time)

print('Ensemble models training.')
start_time = time.time()
for i, model in enumerate(mdl_ensemble):
    print('Model', i, 'training')
    model.fit(x=train_x, y=train_y, batch_size=BATCH_SIZE, epochs=5,
verbose=1,
              validation_data=(valid_x, valid_y), workers = 4)
end_time = time.time()
print('Ensemble (10 models) training (36000 images + 9000 validation)
time, in seconds:',
      end_time-start_time)


# In[15]:
```

```
mdl.save('VGG16_Dropout.h5')
for i, model in enumerate(mdl_ensemble):
    model.save('{}_Ensemble.h5'.format(ensemble_names[i]))


# In[16]:


def make_softmax_predictions(model, test_input, n_iter=100,
ensemble=False):
    if ensemble:
        preds = np.zeros(shape=(test_input.shape[0], len(model),
N_CLASSES))
        for i, mdl in enumerate(model):
            preds[:,i] = mdl.predict(test_input, batch_size = BATCH_SIZE,
workers = 4)
    else:
        preds = np.zeros(shape=(test_input.shape[0], n_iter, N_CLASSES))
        for i in range(n_iter):
            preds[:,i] = model.predict(test_input, batch_size =
BATCH_SIZE, workers = 4)
    return preds

def variation_uncertainty(preds):
    T = preds.shape[1]
    means = np.sum(preds, axis=1) / T
    preds_prob = np.max(means, axis=1)
    preds_labels = np.argmax(means, axis=1)
    preds_variance = np.array([np.sqrt((1/T) * np.sum((pred - mean) **
2)) for pred,mean in zip(preds, means)])
    return preds_prob, preds_labels, preds_variance


# In[17]:


start_time = time.time()
preds_dropout = make_softmax_predictions(mdl,
test_x[:ood_test_x.shape[0]])
preds_dropout_ood = make_softmax_predictions(mdl, ood_test_x)
end_time = time.time()
print('MCDO inference (2000 images, 100 per-example iterations) time, in
seconds:',
      end_time-start_time)

start_time = time.time()
preds_ensemble = make_softmax_predictions(mdl_ensemble,
test_x[:ood_test_x.shape[0]], ensemble=True)
preds_ensemble_ood = make_softmax_predictions(mdl_ensemble, ood_test_x,
ensemble=True)
end_time = time.time()
print('Ensemble inference (2000 images) time, in seconds:', end_time-
start_time)


# In[18]:
```

```
prob_dropout, label_dropout, uncertainty_dropout =
variation_uncertainty(preds_dropout)
prob_dropout_ood, label_dropout_ood, uncertainty_dropout_ood =
variation_uncertainty(preds_dropout_ood)

prob_ensemble, label_ensemble, uncertainty_ensemble =
variation_uncertainty(preds_ensemble)
prob_ensemble_ood, label_ensemble_ood, uncertainty_ensemble_ood =
variation_uncertainty(preds_ensemble_ood)


# In[19]:


print('Dropout model accuracy:', sum(label_dropout ==
test_y[:len(label_dropout)]
                                     ) / len(label_dropout))
print('Ensemble accuracy:', sum(label_ensemble ==
test_y[:len(label_ensemble)]
                                 ) / len(label_ensemble))


# In[20]:


# plot dropout graphs
figure, axis = plt.subplots(1, 3, figsize=(24,6))
idx_correct_dropout = np.where(label_dropout ==
test_y[:len(label_dropout)])
idx_incorrect_dropout = np.where(label_dropout !=
test_y[:len(label_dropout)])

axis[0].scatter(prob_dropout[idx_correct_dropout],
uncertainty_dropout[idx_correct_dropout], label='Correct')
axis[0].scatter(prob_dropout[idx_incorrect_dropout],
uncertainty_dropout[idx_incorrect_dropout], label='Incorrect')
axis[0].set_xlabel('Probability')
axis[0].set_ylabel('Uncertainty')
axis[0].set_title('MCDO (In distribution)')

axis[1].scatter(prob_dropout, uncertainty_dropout, label='In
distribution')
axis[1].scatter(prob_dropout_ood, uncertainty_dropout_ood, label='Out of
distribution', alpha=0.6)
axis[1].set_xlabel('Probability')
axis[1].set_ylabel('Uncertainty')
axis[1].set_title('MCDO (In/Out of distribution)')

axis[2].scatter(prob_dropout_ood, uncertainty_dropout_ood)
axis[2].set_xlabel('Probability')
axis[2].set_ylabel('Uncertainty')
axis[2].set_title('MCDO (Out of distribution)')

plt.legend()
```

```
# In[21]:


# plot ensemble graphs
figure, axis = plt.subplots(1, 3, figsize=(24,6))
idx_correct_ensemble = np.where(label_dropout ==
test_y[:len(label_dropout)])
idx_incorrect_ensemble = np.where(label_dropout !=
test_y[:len(label_dropout)])

axis[0].scatter(prob_ensemble[idx_correct_ensemble],
uncertainty_ensemble[idx_correct_ensemble], label='Correct')
axis[0].scatter(prob_ensemble[idx_incorrect_ensemble],
uncertainty_ensemble[idx_incorrect_ensemble], label='Incorrect')
axis[0].set_xlabel('Probability')
axis[0].set_ylabel('Uncertainty')
axis[0].set_title('Ensemble (In distribution)')

axis[1].scatter(prob_ensemble, uncertainty_ensemble, label='In
distribution')
axis[1].scatter(prob_ensemble_ood, uncertainty_ensemble_ood, label='Out
of distribution', alpha=0.6)
axis[1].set_xlabel('Probability')
axis[1].set_ylabel('Uncertainty')
axis[1].set_title('Ensemble (In/Out of distribution)')

axis[2].scatter(prob_ensemble_ood, uncertainty_ensemble_ood)
axis[2].set_xlabel('Probability')
axis[2].set_ylabel('Uncertainty')
axis[2].set_title('Ensemble (Out of distribution)')

plt.legend()


# In[22]:


print('MCDO mean uncertainty (In distribution):',
np.mean(uncertainty_dropout))
print('MCDO mean uncertainty (Out of distribution):',
np.mean(uncertainty_dropout_ood))
print('MCDO mean uncertainty difference:',
np.mean(uncertainty_dropout_ood) - np.mean(uncertainty_dropout))

print('Ensemble mean uncertainty (In distribution):',
np.mean(uncertainty_ensemble))
print('Ensemble mean uncertainty (Out of distribution):',
np.mean(uncertainty_ensemble_ood))
print('Ensemble mean uncertainty difference:',
np.mean(uncertainty_ensemble_ood) - np.mean(uncertainty_ensemble))
```