НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кваліфікаційна наукова
праця на правах рукопису

МЕЛЬНИЧЕНКО АРТЕМ ВАСИЛЬОВИЧ

УДК 004.94

ДИСЕРТАЦІЯ

МЕТОДИ ТА ПРОГРАМНІ ЗАСОБИ ПІДВИЩЕННЯ ШВИДКОДІЇ МОДЕЛЕЙ РОЗПІЗНАВАННЯ ОБРАЗІВ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

121 – Інженерія програмного забезпечення

12 – Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії.

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____Мельниченко А.В.

Наукові керівники:
ШАЛДЕНКО Олексій Вікторович,
кандидат технічних наук, доцент
НЕДАШКІВСЬКИЙ Олексій Леонідович,
доктор технічних наук, доцент

КИЇВ – 2024

АНОТАЦІЯ

Мельниченко А.В. Методи та програмні засоби підвищення швидкодії моделей розпізнавання образів на основі машинного навчання. − Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії з галузі знань 12 Інформаційні технології за спеціальністю 121 Інженерія програмного забезпечення. – Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, 2024.

Дисертаційна робота присвячена аналізу методів оптимізації нейронних мереж і розробці програмних засобів для збільшення швидкодії нейронних мереж під час навчання і виконання.

У сучасному високотехнологічному світі, нейронні мережі вийшли на передній план як ключова технологія. Ця варіація математичних моделей продемонструвала високу ефективність у багатьох задачах, що варіюються від комп'ютерного зору до розуміння природніх мов, тим самим ставши невід'ємною частиною щоденного життя. Втім, розгортання нейронних мереж у реальних сценаріях часто ускладняється їхньою обчислювальною складністю та ресурсоємністю. Великий об’єм енергоспоживання, що потребується для навчання і використання великих моделей нейронних також має негативний вплив на навколишнє середовище.

Обчислювальна складність часто проявляється у вигляді великої кількості параметрів та глибоких архітектур, які вимагають значного об’єму обчислювальної потужності як для навчання, так і для подальшого використання на кінцевих пристроях. Ця складність є особливо проблематичною в застосуваннях нейронних мереж на пристроях Інтернету речей (IoT), де обчислювальні ресурси часто обмежені. Ресурсоємні характеристики включають в себе обчислювальну потужність і використання пам'яті. Це питання є особливо актуальним у мобільних та вбудованих пристроях, де пам'ять є обмеженим

ресурсом. Більше того, затримка, спричинена нестачею ресурсів, часто є неприйнятною в ряді задач, що включає в себе системи автономного керування, де навіть невелика затримка в прийнятті рішень може мати серйозні наслідки.

Оптимізація нейронних мереж є актуальною задачею в технологічній галузі, що підкреслюється емпіричними даними. Об’єм обчислювальних ресурсів, необхідний для навчання найсучасніших нейронних мереж, подвоювався приблизно кожні 3 місяці з 2012 року. Це експоненційне зростання обчислювальних вимог не є сталим на довгострокову перспективу, особливо з урахуванням енергоспоживання та екологічного впливу, пов'язаного з дата-центрами.

Метою дисертації є збільшення ефективності моделей нейронних мереж, а саме зменшення втрати точності при збільшенні швидкодії, після застосування методів оптимізації моделей глибинного навчання, створених для вирішення задач комп’ютерного зору.

В області оптимізації нейронних мереж існуючі дослідження виділяють декілька найбільш поширених методів серед яких методи низькорангової факторизації вагів, квантування і прунінг. Методи низькорангової факторизації використовують концепцію того, що вагові матриці в нейронних мережах часто містять значну кількість надмірності. Апроксимуючи ці вагові матриці факторизаціями нижчого рангу можна досягти зменшення кількості параметрів мережі. Квантування дозволяє зменшити обчислювальну складність нейромереж шляхом представлення числових значень із типами даних нижчої точності, такими як цілі числа, замість традиційних 32-розрядних чисел з плаваючою комою. Попри ти що квантування і низькорангова факторизація є досить ефективними методами, квантування потребує вже натренованої мережі, а низькорангова факторизація вагових матриць часто призводить до вагомих втрат точності мережі.

Прунінг нейронних мереж, ключовий метод оптимізації нейронних мереж, ґрунтується на концепції спрощення складних математичних моделей зі збереженням або покращенням їхньої продуктивності. Цей підхід стає все більш актуальним, оскільки нейронні мережі дедалі частіше стають частиною додатків, що вимагає ефективного використання обчислювальних ресурсів. Галузь методів прунінгу перед навчанням дозволяє оптимізувати мережу перед стадією навчання, тим самим потенційно скорочуючи витрати на навчання.

Прунінг передбачає систематичне видалення менш значущих параметрів нейронної мережі, таких як ваги або зв'язки, таким чином зменшуючи розмір моделі та обчислювальну складність. Цей процес має вирішальне значення для розгортання нейронних мереж в середовищах з обмеженими обчислювальними ресурсами або там, де важлива швидка обробка даних.

Розробка та вдосконалення методів прунінгу є основним напрямком досліджень в галузі оптимізації нейронних мереж. Прунінгу присвячені роботи зарубіжних вчених: Molchanov P., Tyree S., Hu H., Peng R., Li H., Kadav A., He Y., Zhang X., Narang S., Elsen E. Прунінгу і оптимізації перед навчанням присвячені роботи Lee N., Ajanthan T., Frankle J., Carbin M. Розробці методів архітектурної оптимізації присвячені роботи Сінькевич О.О., Терейковський І.А., Кудін О.В., Кривохата А.Г., Howard A. G., Zhu M., Hinton G., Dean J. та інші. Дослідженням методів зниження витрат обчислювальних ресурсів займались Рувінська В.М., Тімков Ю.Ю., Струнін І.В., Прогонов Д.О. Liang T., Li B., Kong Z. Tan M., Wang Z., Frankle J., Carbin M. Han S., Pool J., Li H. та інші.

Методи прунінгу перед навчанням, такі як SNIP (Single-Shot Network Pruning), надають можливість оптимізації нейронних мереж на ранніх етапах розробки. SNIP обчислює оцінку значущості для кожної ваги на основі її внеску у функцію втрат і відсікає найменш значущі ваги перед початком навчання. Однак цей метод може потребувати коригувань для конкретного набору даних.

Незважаючи на високі показники ефективності методів прунінгу, значна кількість досліджень була проведена на типових моделях, тому існує потреба в дослідженнях методів і їх ефективності для нових моделей в області комп`ютерного зору.

Дисертаційна робота виконана відповідно з поточними та перспективними планами наукової та науково-технічної діяльності Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» і є частиною досліджень в рамках науково-дослідницької роботи «Методи і алгоритми оптимізації розпізнавання образів на основі методів машинного навчання» (Державний реєстраційний номер №0121U109207, м. Київ). Особисто автором в НДР №0121U109207 запропоновано вдосконалений метод розрахунку критерію важливості ваг нейронної мережі при видаленні вагів перед навчанням для моделі розпізнавання обличчя, та моделі архітектури трансформер. Проведено експерименти для оцінки ефективності розробленого методу.

Наукова новизна одержаних результатів полягає в наступному.

Удосконалено модель нейронної мережі для виявлення облич RetinaFace, яка на відміну від існуючих використовує метод прунінгу SNIP для оптимізації, що дозволяє використовувати розріджені матриці для зберігання і виконання мережі з метою подальшого удосконалення та збільшення швидкодії.

Удосконалено метод прунінгу SNIP для моделі виявлення облич RetinaFace, який на відміну від існуючих передбачає можливість виключення контекстних модулів з процесу прунінгу. Вдосконалений метод дозволяє досягти більшої точності при незмінній кількості виключених параметрів.

Вперше розроблено метод прунінгу перед навчанням для моделей архітектури трансформер, який на відміну від існуючих враховує важливість механізму «уваги». Використання розробленого методу дозволяє значно збільшити точність класифікації кінцевої моделі в порівнянні з методом SNIP.

Вперше розроблено архітектуру програмного забезпечення для моделювання та дослідження методів прунінгу перед навчанням нейронних мереж, яка на відміну від існуючих дозволяє приводити матриці вагових коефіцієнтів мережі до розрідженого формату, використовуючи запропонований механізм оцінки важливості вагів.

Оптимізована мережа RetinaFace містить на 68% параметрів менше ніж початкова мережа при втраті точності на лише 1.4%. Вдосконалений метод дозволив зменшити втрати точності з 1.4% до 0.7% порівняно з методом SNIP при порівнянні з необрізаною моделлю, при скороченні параметрів на 68%.

Реалізація методу прунінгу для архітектури трансформер дозволила натренувати мережу з покращенням точності до 37% порівняно з методом SNIP при порівнянні з необрізаною моделлю, при скороченні кількості параметрів на 90%.

Встановлено, що результати визначення критеріїв важливості вагів, отриманих розробленим алгоритмом, можуть бути використані для підвищення швидкодії нейронних мереж від 20% до 65% шляхом використання розріджених матриць формату 2:4, в залежності від графічного процесора.

Встановлено, що додаткові виходи для сіамських нейронних мереж, призначених для встановлення схожості двох зображень, не дають приросту в швидкості сходження і точності моделі.

Дослідження показало, що наразі оцінка методів прунінгу проводилась на типових архітектурах моделей нейронних мереж. При дослідженні мережі для виявлення облич і застосуванні методу прунінгу перед навчанням, було виявлено що мережа може зберігати високий рівень точності, при цьому значно знижуючи кількість параметрів що беруть участь у навчанні. Це пришвидшує навчання і полегшує подальше використання моделей, оскільки сучасні CPU і GPU оптимізують обчислення, в яких беруть участь розріджені матриці.

Методика дослідження та отриманні результати можуть також бути використані для створення та оцінки архітектур нейронних мереж для інших доменів, тим самим розширюючи сферу потенційних застосувань. Дослідження може стати основою для оцінки ефективності інших методів оптимізації для моделей комп’ютерного зору та внести вклад у зростаючий обсяг наукової літератури з оптимізації нейронних мереж.

Наукові результати досліджень є внеском у розвиток теоретичних і прикладних основ розробки й дослідження науково-методичного і програмного апарату для оптимізації глибоких нейронних мереж з розпізнавання образів.

Наступними перспективними дослідженнями можуть стати дослідження для вдосконалення критеріїв визначення важливості вагів, ітеративне додавання параметрів під час навчання, дослідження інших форматів стиснених матриць та розширення експериментів на інші задачі та архітектури нейронних мереж.

Ключові слова: інформаційні технології, швидкодія, оптимізація, нейронна мережа, машинне навчання, штучний інтелект, обробка зображень, аналіз даних, розпізнавання облич, хмарне середовище, комп’ютерна система, програмне забезпечення, архітектура програмної системи, мікросервісна архітектура, розріджені матриці.

ANNOTATION

Melnychenko A.V. Methods and software tools for improving the performance of pattern recognition models based on machine learning - Qualifying scientific work on the rights of the manuscript.

Thesis for the degree of Doctor of Philosophy in the field of knowledge 12 Information Technologies, specialty 121 Software Engineering. - National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, 2024.

The dissertation is devoted to the analysis of neural network parameter reduction methods and the development of software tools to increase the performance of neural networks during training and execution.

In today's high-tech world, neural networks have become a crucial technology. This variation of mathematical models has demonstrated high performance in many tasks ranging from computer vision to natural language understanding, thereby becoming an integral part of everyday life. However, the deployment of neural networks in real-world scenarios is often hampered by their computational complexity and resource intensity. The large amount of power consumption required to train and use of large neural models also has a negative impact on the environment.

Computational complexity often manifests itself in the form of a large number of parameters and deep architectures that require a significant amount of computing power both for training and for further use on end devices. This complexity is particularly problematic in applications of neural networks on Internet of Things (IoT) devices, where computing resources are often limited. Resource-intensive characteristics include computing power and memory usage. This issue is particularly relevant in mobile and embedded devices where memory is a limited resource. Moreover, the latency caused by a lack of resources is often unacceptable in a number of tasks, including autonomous control systems, where even a small delay in decision-making can have serious consequences.

Optimization of neural networks is an urgent task in the technology industry, which is emphasized by empirical data. The amount of computing resources required to train state-of-the-art neural networks has doubled approximately every 3 months since 2012. This exponential growth in computational requirements is not sustainable in the long term, especially considering the energy consumption and environmental impact associated with data centers.

The purpose of this thesis is to increase the efficiency of neural network models, namely, to reduce the loss of accuracy while increasing performance, after applying methods for optimizing deep learning models created to solve computer vision problems.

In the field of neural network optimization, existing research identifies several of the most common methods, including low-rank weight factorization, quantization, and pruning. Low-rank factorization methods use the concept that weight matrices in neural networks often contain a significant amount of redundancy. By approximating these weight matrices with lower-rank factorizations, the number of network parameters can be reduced. Quantization reduces the computational complexity of neural networks by representing numerical values with lower precision data types, such as integers, instead of the traditional 32-bit floating point numbers. Although quantization and low-rank factorization are quite effective methods, quantization requires an already trained network, and low-rank factorization of weight matrices often leads to significant losses in network accuracy.

Neural network pruning, a key method of neural network optimization, is based on the concept of simplifying complex mathematical models while maintaining or improving their performance. This approach is becoming increasingly relevant as neural networks are increasingly becoming part of applications that require efficient use of computing resources. The field of pre-training pruning methods allow to optimize the network before the training stage, thereby potentially reducing training costs.

Pruning involves the systematic removal of less significant neural network parameters, such as weights or connections, thus reducing model size and computational complexity. This process is crucial for deploying neural networks in environments with limited computing resources or where fast data processing is important.

The development and improvement of pruning methods is a major area of research in the field of neural network optimization. The works of foreign scientists are devoted to the pruning: Molchanov P., Tyree S., Hu H., Peng R., Li H., Kadav A., He Y., Zhang X., Narang S., Elsen E. Pruning and pre-training optimization are studied by Lee N., Ajanthan T., Frankle J., Carbin M. Development of architectural optimization methods is studied by Sinkevych O., Tereykovsky A., Kudin O., Krivokhata A., Howard A. G., Zhu M., Hinton G., Dean J. and others. Methods for reducing the cost of computing resources were studied by Ruvinska V. M., Timkov Y. Y., Strunin I. V., Progonov D. O. Liang T., Li B., Kong Z. Tan M., Wang Z., Frankle J., Carbin M. Han S., Pool J., Li H. and others.

Pre-training pruning methods, such as SNIP (Single-Shot Network Pruning), provide an opportunity to optimize neural networks in the early stages of development. SNIP calculates a salience score for each weight based on its contribution to the loss function and prunes the least significant weights before training. However, this method may require adjustments for a particular data set.

Despite the high performance of the pruning methods, a significant number of studies have been conducted on typical models, so there is a need to investigate the methods and their effectiveness for new models in the field of computer vision.

The dissertation was performed in accordance with the current and future plans of scientific and scientific and technical activities of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" and is part of the research within the research work "Methods and algorithms for optimizing pattern recognition based on machine learning methods" (State registration number 0121U109207, Kyiv).The

author personally proposed an improved method for calculating the criterion of importance of neural network weights when removing weights before training for the face recognition model and the transformer architecture model in the research work No. 0121U109207. Experiments were conducted to evaluate the effectiveness of the developed method.

The scientific novelty of the results is as follows.

An improved model of the RetinaFace neural network for face detection is proposed, which, unlike the existing ones, uses the SNIP pruning method for optimization, which allows the use of sparse matrices for storing and executing the network for further improvement and performance.

An improved SNIP pinning method for the RetinaFace face detection model is proposed, which, unlike the existing ones, provides for the possibility of excluding contextual modules from the pinning process. The improved method allows achieving higher accuracy with the same number of excluded parameters.

For the first time, a pre-training tuning method for transformer architecture models has been developed, which, unlike the existing ones, takes into account the importance of the "attention" mechanism. The use of the developed method allows to significantly increase the accuracy of classification of the final model compared to the SNIP method.

For the first time, a software architecture for modelling and studying pre-training methods for neural networks has been developed, which, unlike existing ones, allows to reduce the matrix of network weights to a sparse format using the proposed mechanism for assessing the importance of weights.

The optimized RetinaFace network contains 68% fewer parameters than the original network, with a loss of accuracy of only 1.4%. The improved method reduced the accuracy loss from 1.4% to 0.7% compared to the SNIP method when compared to the uncropped model, with a 68% reduction in parameters.

Implementation of the pruning method for the transformer architecture allowed to train the network with an accuracy improvement of up to 37% compared to the SNIP method when compared to the uncut model, while reducing the number of parameters by 90%.

The results of determining the criteria for the importance of weights obtained by the developed algorithm can be used to increase the performance of neural networks from 20% to 65% by using sparse matrices of 2:4 format, depending on the GPU.

The study established that additional outputs for Siamese neural networks designed to establish the similarity of two images do not increase the speed of convergence and model accuracy.

The study has shown that so far, the evaluation of the pruning methods has been carried out on typical neural network model architectures. When studying a network for face detection and applying the pre-training method, it was found that the network can maintain a high level of accuracy while significantly reducing the number of parameters involved in training. This speeds-up training and facilitates further use of the models, as modern CPUs and GPUs optimize computations involving sparse matrices.

The research methodology and results can also be used to create and evaluate neural network architectures for other domains, thereby expanding the scope of potential applications. The research can serve as a basis for evaluating the effectiveness of other optimization methods for computer vision models and contribute to the growing body of scientific literature on neural network optimization.

The scientific results of the research are a contribution to the development of theoretical and applied foundations for the development and research of scientific, methodological and software tools for optimizing deep neural networks for pattern recognition.

Future research may include studies to improve the criteria for determining the importance of weights, iteratively adding parameters during training, exploring other

formats of compressed matrices, and extending experiments to other tasks and neural network architectures.

Keywords: information technology, performance, optimization, neural network, machine learning, artificial intelligence, image processing, data analysis, face recognition, cloud environment, computer system, software, software system architecture, microservice architecture, sparse matrices.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Melnychenko, A., Shaldenko, O. Evaluation of a snip pruning method for a state-of-the-art face detection model. *Computational Problems of Electrical Engineering*, 2023, Vol. 12, №1, pp. 22-27

2. Melnychenko, A., Zdor K. Incorporating attention score to improve foresight pruning on transformer models. *Computer Science and Applied Mathematics*, 2023, №2, pp.22-28

3. Melnychenko, A., Zdor, K. Efficiency of supplementary outputs in siamese neural networks. *Advanced Information Systems*, 2023,Volume *7*, №3, pp. 49–53.

Наукові праці, які засвідчують апробацію матеріалів дисертації:

1. Мельниченко, А., Шалденко, О. Особливості використання прунінгу перед тренуванням нейронної мережі для детекції обличчя, XX Міжнародна науково-практична конференція молодих вчених і студентів, 25–28 квітня 2023 року, Київ, Україна

2. Melnychenko A. Evaluating SNIP pruning method on the state-of-the-art face detection model. Modern scientific research: achievements, innovations and development prospects, XVI Міжнародна науково-практична конференція, 11-13 вересня 2022 року, Берлін, Німеччина. С. 68-72.

3. Melnychenko, A., Zdor, K. Applying classification and regression supplemetary output in siamese neural network using fashion MNIST and plantvillage datasets, VII Міжнародна науково-практична конференція “Modern problems of science, education and society”, 11-13 вересня 2023 Київ, Україна, С. 126-129.

4. Melnychenko, A., & Zdor, K. Appling classification and regression supplemetary outputs in siamese neural network using plantvillage dataset, I Міжнародна науково-практична конференція “Current challenges of science and education”, 18-20 вересня 2023, Берлін, Німеччина. С. 79-82.

5. Melnychenko A., Zdor K. Appling classification and regression supplemetary output in siamese neural network using fashion MNIST and plantvillage datasets, X Міжнародна науково-практична конференція “Innovations and prospects in modern science”, 25-27 вересня 2023, Стокгольм, Швеція. С. 87-92.

6. Мельниченко А., Здор К. Збільшення ефективності оптимізації моделей архітектури ViT перед навчанням шляхом включення активацій механізму самоуваги, І міжнародна науково–практична конференція “Сучасні аспекти інженерії програмного забезпечення”, 14 грудня 2023, Київ, Україна.

7. Мельниченко А.В., Здор К.А. Врахування механізмів самоуваги при прунінгу моделей нейронних мереж Vision Transformer. Збірник матеріалів ІІІ Міжнародної науково-технічної конференції “Системи і технології зв’язку, інформатизації та кібербезпеки: актуальні питання і тенденції розвитку”, 30 листопада 2023 року, Київ, Україна. С. 214 – 215.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

Прунінг	– процес обрізки нейронної мережі шляхом видалення частини параметрів
Трансформер	– мережі архітектури transformer
DNN	– глибока нейронна мережа
CNN	– згорткова нейронна мережа
NLP	– обробка природніх мов
SNIP	– single shot network pruning
L1	– функція найменшого абсолютного відхилення
ReLU	– випрямлений лінійний вузол
SGD	– стохастичний градієнтний спуск
GPU	– графічний процесор
CPU	– центральний процесор
JWT	– JSON web token
VM	– віртуальна машина
WAL	– write-ahead logging
СУБД	– система управління базами даних.
ρ	– ступінь стиснення нейронної мережі
$L(W)$	– функція втрат
W	– ваги (параметри) нейронної мережі
W_{enc}	– ваги (параметри) кодера
W_{dec}	– ваги (параметри) декодера
q_i	– вектор запитів
k_i	– вектор ключів
v_i	– вектор значень
Softmax	– нормована експоненційна функція

ВСТУП

Актуальність теми.

У сучасному високотехнологічному світі, нейронні мережі вийшли на передній план як ключова технологія. Ця варіація математичних моделей продемонструвала високу ефективність у задачах, що варіюються від комп'ютерного зору до розуміння природніх мов та автоматизації систем інформаційної безпеки, тим самим ставши невід'ємною частиною щоденного життя [1, 2, 3]. Втім, розгортання нейронних мереж у реальних сценаріях часто ускладняється їхньою обчислювальною складністю та ресурсоємністю. Великий об’єм енергоспоживання, що потребується для навчання і використання великих моделей нейронних також має негативний вплив на навколишнє середовище [4].

Обчислювальна складність часто проявляється у вигляді великої кількості параметрів та глибоких архітектур, які вимагають значної обчислювальної потужності як для навчання, так і для використання. Найбільше обчислювальна складність створює виклики в застосуваннях нейронних мереж на пристроях Інтернету речей IoT де обчислювальні ресурси часто обмежені [5]. Характеристики ресурсоємності включають в себе обчислювальну потужність і об’єм використання пам'яті. Затримка, спричинена вимогами до ресурсів, часто є неприйнятною в ряді задач, що включає в себе системи автономного керування, де навіть невелика затримка в прийнятті рішень може мати серйозні наслідки [6].

Оптимізація нейронних мереж є актуальною задачею в технологічній галузі, про що свідчить дослідження про зростання вимог нейронних мереж до ресурсів. Об’єм обчислювальних ресурсів, необхідний для навчання найсучасніших нейронних мереж, подвоювався приблизно кожні 3 місяці з 2012 року. [7]. Це експоненційне зростання обчислювальних вимог не є сталим на

довгострокову перспективу, особливо з урахуванням енергоспоживання та екологічного впливу, пов’язаного з дата-центрами.

Зростаючий тренд до edge computing накладає додаткові обмеження на об’єм наявних обчислювальних ресурсів. Дослідження від Gartner містить прогноз, що до 2025 року до 75% даних, згенерованих підприємствами, буде оброблятися на периферії, а не в централізованих дата-центрах [8]. Виявлена тенденція спонукає до оптимізації нейронних мереж.

Техніки оптимізації, такі як квантування, прунінг та дистиляція знань, дозволяють зменшити кількість параметрів нейронних мереж без значущого падіння в точності. Ряд методів прунінгу дозволяє визначати значущість параметрів ще до початку навчання нейронної мережі.

Зв’язок роботи з науковими програмами, планами, темами.

Дисертаційна робота виконана відповідно з поточними та перспективними планами наукової та науково-технічної діяльності Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» для подальшого розвитку інженерії програмного забезпечення. Дослідження тісно пов'язано з розробкою науково-дослідницькою роботою (НДР), в яких автор приймав особисту участь, а саме: «Методи і алгоритми оптимізації розпізнавання образів на основі методів машинного навчання» №0121U109207, що виконувалась в Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського» у 2020 – 2023 рр. Особисто автором в НДР запропоновано удосконалений метод прунінгу для моделі розпізнавання облич, удосконалений метод прунінгу для моделей архітектури трансформер з урахуванням оцінок уваги при розрахунку критеріїв важливості, використання оцінок методу прунінгу перед навчанням для формування розріджених матриць формату 2:4 для підвищення швидкодії.

Тому актуальною є наукова задача дослідження методів оптимізації і збільшення швидкодії нейронних мереж.

Мета і завдання дослідження.

Метою дисертації є збільшення ефективності моделей нейронних мереж, а саме зменшення зменшення втрати точності при збільшенні швидкодії, після застосування методів оптимізації моделей глибинного навчання, створених для вирішення задач комп’ютерного зору.

Для досягнення мети в дисертації вирішено такі **наукові завдання:**

1. Виконано аналіз існуючих методів оптимізації нейронних мереж.
2. Досліджено алгоритми тренування моделей нейронних мереж, способи ініціалізації вагів нейронних мереж і підходи до вирішення задач розпізнавання образів.
3. Досліджено ефективність існуючих методів оптимізації при застосуванні їх на мережах призначених для розпізнавання образів.
4. Удосконалена модель нейронної мережі для виявлення облич RetinaFace.
5. Удосконалено метод прунінгу SNIP для моделі виявлення облич RetinaFace.
6. Вперше розроблений метод прунінгу перед навчанням для моделей архітектури трансформер.
7. Вперше розроблена архітектура програмного забезпечення для моделювання та дослідження

Об'єкт досліджень: методи та підходи оптимізації навчання нейронних мереж для вирішення задач розпізнавання образів з метою підвищення швидкодії.

Предмет досліджень: оптимізація процесу навчання нейронних мереж для вирішення задач розпізнавання образів з метою підвищення швидкодії.

Методи дослідження. Для досягнення поставленої в роботі мети використано методи алгоритмічного та порівняльного аналізу (для визначення актуальності та постановки наукового завдання дисертаційної роботи). Для

оцінки роботи алгоритмів оптимізації і прунінгу були використані методи дослідження на основі математичної статистики і машинного навчання. Для дослідження моделей для вирішення задач комп`ютерного зору було використано методи обробки сигналів і алгоритми ітеративної оптимізації.

Наукова новизна одержаних результатів полягає в тому, що в дисертаційній роботі:

- Вперше застосовано метод прунінгу перед навчанням для моделі виявлення обличчя на зображенні архітектури RetinaFace. Використання даного методу дозволило скоротити кількість параметрів в нейронній мережі на 68% при втраті точності на 0.7%;

- Удосконалено метод прунінгу перед навчанням для моделі виявлення облич, що враховує важливість контекстних модулів в архітектурі нейромережі, який в свою чергу дозволив підвищити точність на 0.7% порівняно з методом SNIP;

- Вперше розроблено метод прунінгу перед навчанням для моделей трансформерів, в якому на відміну від існуючих методів враховано важливість механізму уваги. Реалізація даного методу дозволила натренувати мережу з покращенням точності до 37% порівняно з методом SNIP;

- Встановлено, що результати визначення критеріїв важливості вагів, отриманих розробленим алгоритмом, можуть бути використані для підвищення швидкодії нейронних мереж від 20% до 65% шляхом використання розріджених матриць формату 2:4;

- Встановлено, що додаткові виходи для сіамських нейронних мереж, призначених для встановлення схожості двох зображень, не дають приросту в швидкості сходження і точності моделі;

- Розроблено програмний комплекс для аналізу і застосування методів прунінгу на моделях нейронних мереж, що використовує хмарні ресурси Azure для хостингу обчислювальних ресурсів.

Практичне значення дисертаційного дослідження полягає у можливості його застосування для планування та розробки методів оптимізації для моделей глибинного навчання, розроблених для вирішення задач комп'ютерного зору. Дослідження призвело до створення бази даних, що складається з попередньо оброблених та оцінених моделей та методів оптимізації, яка створює основу для подальших експериментів та оцінки.

Дослідження показало, що наразі оцінка методів прунінгу проводилась на типових архітектурах моделей нейронних мереж. При дослідженні мережі для виявлення облич, і застосуванні методу прунінгу перед навчанням, було виявлено що мережа може зберігати високий рівень точності, при цьому значно знижуючи кількість параметрів що беруть участь у навчанні. Це пришвидшує навчання і полегшує подальше використання моделей, оскільки сучасні CPU і GPU мають можливість оптимізувати обчислення, в яких беруть участь розріджені матриці.

Була розроблена інноваційна модифікація методу прунінгу перед навчанням для моделей трансформерів, використання якої до покращення точності на 37% у порівнянні з існуючим алгоритмом SNIP при показнику зменшення вагів, що беруть учать в тренування в 10 разів. Оптимізація особливо актуальна для нейронних мереж даної архітектури, адже вони зазвичай мають нелінійну складність обчислень, і потребують значних обчислювальних ресурсів для тренування.

Практичне значення одержаних результатів полягає в їх застосуванні для оптимізації та збільшення швидкодії глибоких нейронних мереж при навчанні і виконанні. Створено програмне забезпечення для прунінгу і тренування нейронних мереж в хмарному середовищі Azure. Встановлено, що архітектурні особливості нейронних мереж архітектури трансформер можуть бути використані для підвищення ефективності прунінгу нейронних мереж. Розроблений вдосконалений метод прунінгу дозволяє отримати точність на 34%

більше, ніж існуючий метод, а використання розріджених матриць дозволяє пришвидшити виконання нейронної мережі від 20% до 65%.

Методика дослідження та отриманні результати можуть також бути використані для створення та оцінки архітектур нейронних мереж для інших доменів, тим самим розширюючи сферу потенційних застосувань. Дослідження може стати основою для оцінки ефективності інших методів оптимізації для моделей комп'ютерного зору та внести вклад у зростаючий обсяг наукової літератури з оптимізації нейронних мереж.

Результати досліджень прийняті до впровадження в Товаристві з обмеженою відповідальністю «ВОТЧЕД» (акт від 20.10.2023р.); в навчальному процесі Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (акт впровадження від 19.01.2024р.) при викладанні дисципліни «Технології розпізнавання образів та машинний зір»» для студентів освітньо-кваліфікаційного рівня «Магістр» спеціальності 122 «Комп'ютерні науки».

Особистий внесок здобувача. Дисертаційна робота є самостійно виконаною науковою працею. Всі представлені наукові результати, приклади та експериментальні розрахунки, викладені у дисертації, одержані здобувачем одноосібно. У наукових роботах, що опубліковані у співавторстві, в дисертаційній роботі використані лише ті результати, які становлять індивідуальний внесок автора. У друкованих працях, опублікованих у співавторстві, здобувачеві належать:

Удосконалений методу прунінгу нейронних мереж для виявлення облич на зображеннях. Зазначений метод дозволяє виключити з оцінки важливі шари в мережі, при цьому зберігаючи кількість вагів, що виключаються незмінною. Завдяки цьому точність результуючої мережі збільшується на 0.7% на тестовій вибірці.

Виявлення падіння точності і неефективність оптимізації шляхом додавання додактових виходів для вирішення задач класифікації і регресії в сіамських нейронних мережах.

Удосконалений метод прунінгу перед навчанням для застосування на моделях трансформерів шляхом включення оцінок уваги в розрахунок критерію важливості вагів. Дана модифікація дозволила натренувати мережу що має точність на 37% більше порівняно з оригінальним методом.

Вдосконалений метод прунінгу в поєднанні з використанням розріджених матриць може бути використаний для підвищення швидкодії при тренуванні і виконанні нейронних мереж від 25% до 65%.

Апробація матеріалів дисертації. Результати дисертаційного дослідження апробовано на міжнародних науково-практичних конференціях та семінарах:

– XVI Міжнародна науково-практична конференція “Modern scientific research: achievements, innovations and development prospects”, 11-13 вересня 2022 року, Берлін, Німеччина;

– XX Міжнародна науково-практична конференція молодих вчених і студентів, 25–28 квітня 2023 року, Київ, Україна;

– VII Міжнародна науково-практична конференція “Modern problems of science, education and society”, 11-13.09.2023 Київ, Україна;

– I Міжнародна науково-практична конференція “Current challenges of science and education”, 18-20.09.2023 Берлін, Німеччина;

– X Міжнародна науково-практична конференція “Innovations and prospects in modern science”, 25-27.09.2023 Стокгольм, Швеція;

– I міжнародна науково–практична конференція «Сучасні аспекти інженерії програмного забезпечення», Київ, Україна;

– III Міжнародна науково-технічна конференція “Системи і технології зв’язку, інформатизації та кібербезпеки: актуальні питання і тенденції розвитку”, Київ, Україна.

Публікації. За результатами досліджень опубліковано 3 наукові праці. Основні наукові положення викладено в 3 наукових статтях [88,91,98] у спеціалізованих фахових виданнях України. Із них одна наукова стаття [91] опублікована у періодичному видання, що входить до наукометричної бази SCOPUS. За матеріалами виступів на науково-технічних конференціях опубліковано 7 тез доповідей [111-117].

Структура і обсяг роботи. Дисертація складається зі вступу, 4 розділів, висновків, 3 додатків та списку використаних джерел із 117 найменувань на 14 сторінках. Повний обсяг дисертації складає 178 сторінок, з них 145 сторінок основного тексту.

1. АНАЛІЗ ІСНУЮЧИХ ПРАКТИЧНИХ ТА НАУКОВО-МЕТОДОЛОГІЧНИХ ПІДХОДІВ ЩОДО РІШЕННЯ ЗАДАЧ РОЗПІЗНАВАННЯ ОБРАЗІВ

1.1 Аналіз практичного застосування нейронних мереж для вирішення задач розпізнавання образів

Розпізнавання образів — є актуальною задачею, метою якої є класифікація об'єктів у кілька категорій або класів для знаходження та розпізнавання об’єктів на зображеннях, сигнальних форм хвиль та подібних задач, які потребують аналізу. Розпізнавання образів є актуальною задачею вже багато десятирічь і до 1960-х років цей напрямок розвивався як результат теоретичних досліджень у галузі статистики. Поява та збільшення застосування сучасного високопродуктивного обладнання збільшило попит практичного застосування розпізнавання образів, що, в свою чергу, висунуло нові вимоги до подальших досліджень [9, 10]. З розвитком суспільства від промислової до постіндустріальної фази, автоматизації в промисловому виробництві та потреба в обробці та пошуку інформації стають все більш актуальними задачами. Розпізнавання образів є невід’ємною частиною більшості систем штучного інтелекту створених для прийняття рішень та аналізу, і для ефективного вирішення задачі розпізнавання використовуються методи комп’ютерного зору.

У своєму найбільш зрозумілому та простому визначенні, комп'ютерний зір – це технології, які надають здатність розпізнавати та аналізувати зображення комп’ютерним системам. Основна мета задач комп’ютерного зору — зберігання, перетворення та визначення характеристик інформації. В свою чергу метою комп'ютерного зору — є задача зробити машини здатними розуміти світ, через обробку цифрових сигналів.

За останні два десятиліття спостерігається тенденція до поширення використання нейронних мереж для вирішення задач комп’ютерного зору, зокрема у вирішення задачі виявлення і розпізнавання облич. Ранні підходи до розпізнавання облич в основному базувалися на класифікаторах, побудованих на основі вручну створених ознак, виявлених з локальних областей зображення, таких як каскади Хаара та гістограми напрямлених градієнтів. Однак ці підходи не змогли досягти високої точності на зображеннях з неконтрольованих середовищ.

Перші успішні приклади застосувань глибоких нейронних мереж для вирішення задач комп’ютерного зору у 2012 році мали значний вплив на досліджувані методи [11]. Були проведені дослідження з ефективності застосувань нейронних мереж в прикладних задачах, а саме для обробки зображень [12]. Глибокі архітектури і велика кількість досліджень в області глибоких нейронних мереж зумовили появу моделей, таких як RetinaFace, заснованих на глибокому навчанні, що призвело до збільшення точності зокрема в задачі розпізнавання облич [13]. Технології розпізнавання облич застосовуються в широкому спектрі прикладних задач, від аналізу відео з камер спостереження до біометричної автентифікації, що допомагає покращити інформаційну безпеку підприємств [14].

Протягом останніх двох десятиліть сфера сегментації зображень активно розвивалась, головним чином завдяки використанню методів машинного навчання та глибокого навчання. Ці інновації перетворили ландшафт комп’ютерного зору, підвищуючи точність та ефективність аналізу зображень в різних сферах застосування.

1.1.1 Нейронні мережі для сегментації зображень

На початку 2000-х років для вирішення сегментації зображень в основному використовувались традиційні методи, такі як порогування (thresholding) та виявлення контурів [15, 16, 17]. Дані методи відрізнялись простотою і швидкодією, втім мали низьку точність при роботі зі складними зображеннями та змінами в освітленні і перспективи. Для багатьох прикладних задач була наявна потреба у більш стійких та адаптивних підходах.

У середині 2000-х років набуло поширення використання згорткових нейронних мереж (CNN). CNN дозволили досягти більшої точності ніж традиційні методи, за рахунок більш складного підходу. Мережі даної архітектури можуть виділяти ієрархічні ознаки в процесі навчання безпосередньо з даних. Застосування CNN у завданнях сегментації пікселів дозволило виділяти деталізовані межі об'єктів на цифровому зображенні. Такий підхід дозволив покращити точність сегментації зображень і спонукав до розробки більш складних архітектур [18].

У кінці 2000-х та на початку 2010-х років була розроблена архітектура U-Net, що змогла ще більше покращити результати сегментації, особливо в біомедичній сегментації зображень. U-Net використовувала унікальну структуру зі стисненням та розширенням векторів, що продемонстровано на рисунку 1.1, і дозволяє точно виділяти об'єкти навіть в складних сценах. Її успіх у аналізі медичних зображень продемонстрував можливості глибокого навчання в прикладних задачах, таких як виявлення пухлин і аналіз клітин [19].

У 2018 році була розроблена архітектура Mask R-CNN, що базується на попередньому дослідженні Faster R-CNN, але відрізняється додатковою гілкою обчислень для вирішення задачі сегментації. Такий підхід дозволив виявляти та сегментувати об'єкти на зображеннях одночасно, сприяючи узагальненню виявлених ознак та швидкодії. Універсальність такого підходу зумовила його

обширне використання для сегментації, дозволяючи ідентифікувати та виділяти кілька об'єктів у складних сценах [20].

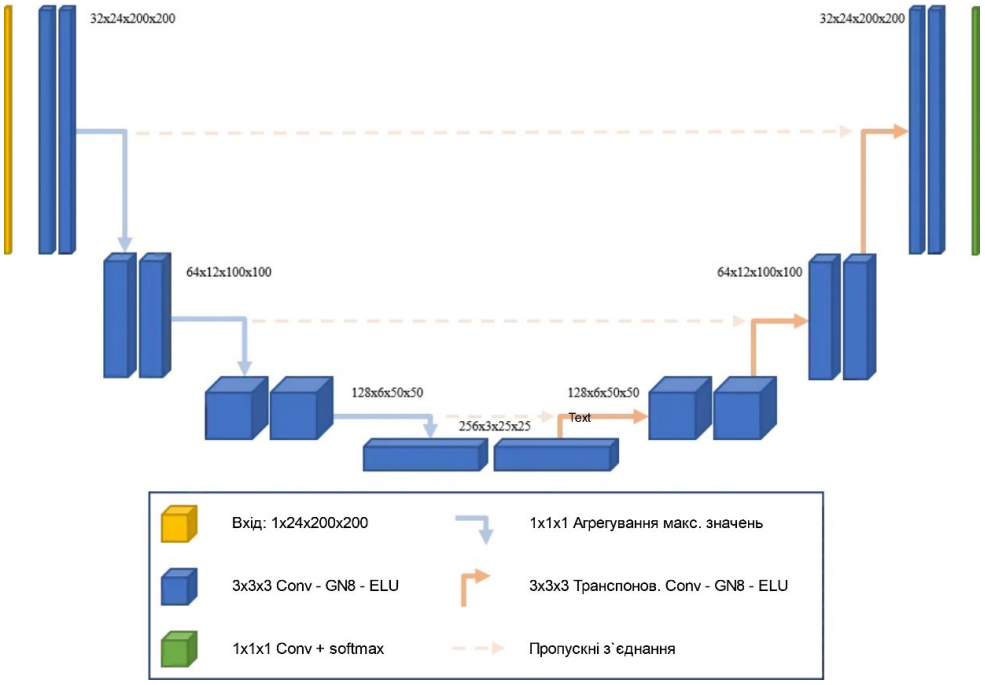

Рисунок 1.1 - Архітектура U-Net.

Паралельно з цим почались дослідження щодо концепції семантичної сегментації. У цьому підході кожному пікселю на зображенні призначається класова мітка, що дозволяє зрозуміти зображення на рівні пікселів. Архітектура PSPNet зробила значний внесок у розвиток цієї області, використовуючи модулі пірамідального пулінгу для покращення репрезентації особливостей об’єктів. Ця модель семантичної сегментації використовувалась для таких прикладних задач, як керування автономними автомобілями, де розуміння сегментованого простору має вирішальний вплив на якість кінцевого продукту [21].

Одним з ключових аспектів останніх досягнень у сегментації зображень є transfer learning (передача знань через попередньо навчені моделі). Дослідження в цій сфері показали, що попередньо навчені моделі на великих наборах даних, таких як ImageNet, можуть служити відмінними вихідними точками для завдань сегментації. Шляхом адаптації цих моделей на менших задачах із власними

наборами даними, стало можливим досягнення ще більших показників точності [22]. Наприклад, архітектури, такі як VGG16, були адаптовані для сегментації, і продемонстрували гнучкість та здатність до узагальнення знань глибоких моделей [23].

Машинне навчання та глибокі нейронні мережі дали можливість покращити точність в багатьох підзадачах сегментації зображень, за рахунок можливості моделей нейронних мереж вивчати складні зв’язки та представлення ознак безпосередньо з даних. Збільшення доступності графічних процесорів (GPU) прискорила навчання глибоких нейронних мереж, роблячи можливим їх використання для ефективної обробки великих наборів даних.

1.1.2 Нейронні мережі для класифікації зображень

На початку 2000-х років для вирішення задач класифікації зображень використовувались традиційні методи комп'ютерного зору, зокрема було поширене ручне виявлення ознак об’єктів та використання поверхневих класифікаторів, таких як машини опорних векторів (SVM) [24]. Дані методи дозволяли вирішувати прості задачі класифікації, втім вони мали низьку точність у роботі зі складними та різноманітними реальними зображеннями. Це зумовило потребу в розробці більш адаптивних методів.

У 2012 році було розроблено архітектуру AlexNet - глибока згорткова нейронна мережа (CNN), після розробки якої набуло поширення використання глибоких моделей нейромереж в класифікації зображень. Саме застосування глибоких шарів та згорток в мережі AlexNet показала кращі результати у порівнянні з традиційними методами в міжнародному змаганні ImageNet Large Scale Visual Recognition. [25]

В наступні роки архітектури глибокого навчання набули ще більшого розвитку, зокрема у 2014 році архітектура GoogLeNet ввела нові концепції в

побудову глибоких нейронних мереж, такі як модулі «Inception» та глобальне агрегування з усередненням (*global average pooling*). Дана архітектура встановила нову найкращу точність на наборі даних ImageNet, зберігаючи при цьому обчислювальну ефективність [26].

У 2015 році було розроблено архітектуру ResNet, що ввела новий підхід до глибоких нейронних мереж - з'єднання з пропуском (Residual block), що продемонстровано на рисунку 1.2. Цей підхід дозволив навчання глибоких нейронних мереж з великою кількістю шарів, а підхід ResNet до збільшення глибини мережі став важливим здобутком в галузі і призвів до покращення результатів в завданнях класифікації зображень.

Технологія transfer learning стала ключовим елементом у класифікації зображень. Було досліджено, що ваги моделі, попередньо навчені на великих наборах даних, таких як ImageNet, можуть використовуватись для виділення ознак замість випадково ініціалізованих вагів для різних задач класифікації. Це дослідження поклало початок розвитку технік transfer learning, що дозволяють налаштовувати моделі, попередньо навчені на великих наборах даних, для менших задач. Популярні архітектури, такі як VGG16, ResNet та Inception, були адаптовані для transfer learning, сприяючи розробці точних класифікаторів з обмеженими даними [27].

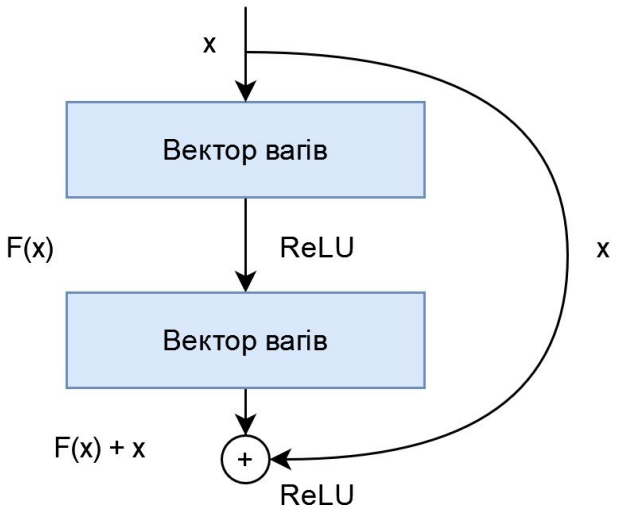

Рисунок 1.2 - З’єднання з пропуском в архітектурі ResNet.

Ансамблеві методи також відіграли важливу роль у покращенні результатів класифікації зображень. Основна перевага використання ансамблевих методів полягає в об'єднанні результатів декількох окремих моделей для підвищення ефективності та запобігання надлишковому впливу перенавчених моделей. Ці підходи часто застосовуються у міжнародних змаганнях з класифікації зображень та в застосунках, які використовують глибокі нейронні мережі [28].

1.1.3 Нейронні мережі для виявлення об’єктів

Протягом останніх двох десятиліть в галузі виявлення об'єктів відбулося багато покращень, переважно завдяки інтеграції методів машинного навчання та глибокого навчання.

На початку 2000-х років задача виявлення об'єктів вирішувалась за допомогою традиційних методів комп'ютерного зору, такі як каскади Хаара та гістограми напрямлених градієнтів. Різні варіації і модифікації цих методів використовувались для вирішення задачі виявлення об’єктів, поки не набули поширення глибокі нейронні мережі [29, 30].

У 2016 році автори архітектури YOLO запропонували переосмислити задачу виявлення об’єктів, і розглянути її як задачу регресії. Розроблений алгоритм обробляє все зображення за один прохід через нейромережу, що дозволяє аналізувати зображення в реальному часі з високою швидкістю та точністю. Нейромережі на основі цієї архітектури використовують в таких прикладних задачах як відеоспостереження в режимі реального часу та при автономне керування автомобілем [31].

Ще одним ефективним одноетапним алгоритмом є SSD (Single Shot MultiBox Detector), розроблений в 2016 році. SSD поєднує кілька конволюційних шарів різної розмірності для виявлення об’єктів на різних етапах проходу по мережі, як показано на рисунку 1.3. Алгоритм покращив точність локалізації

об’єктів, особливо малих, що робить його корисним для застосувань, таких як виявлення облич та пішоходів. [32]

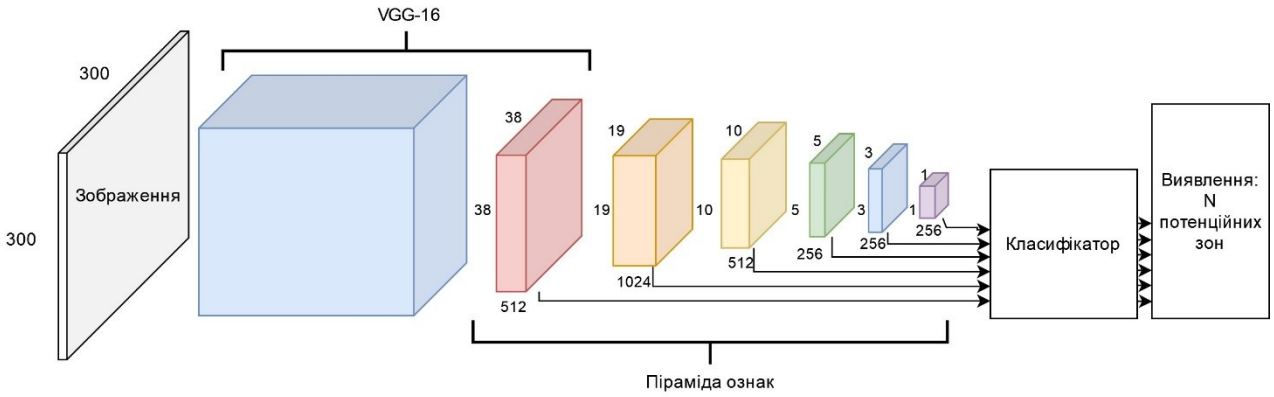

Рисунок 1.3 - Архітектура SSD

Паралельно розвивалися двоетапні методи виявлення об’єктів, які включають стадії пропозиції регіону та класифікації об’єкта, що часто призводило до більш точної локалізації.

Сімейство методів R-CNN, починаючи з оригінального R-CNN розробленого у 2013 році , поклало початок розвитку алгоритмів у двоетапному виявленні об’єктів. R-CNN використовував вибірковий пошук регіонів з потенційними виявленими об’єктами та CNN для класифікації об’єктів. Попри те що архітектура показала покращення точності, її застосування потребувало великого об’єму обчислювальних ресурсів через окремий блок обчислення пропозицій регіону [33]. Архітектура Fast R-CNN вирішувала деякі проблеми обчислювальної неефективності, спільно використовуючи згорткові характеристики для пропозицій регіону та класифікації, проте вона все ще покладалась на зовнішній алгоритм пропозицій регіону [34].

Наступним кроком розвитку була архітектура Faster R-CNN, автори якої інтегрували пропозицію регіону та класифікацію об’єкта в одну нейронну мережу. Нововведенням була мережа Region Proposal Network (RPN), що

дозволила досягти ще більшої точності і швидкості, при цьому сприяючи розвитку підходу end-to-end [35].

Методи, спрямовані на виявлення об’єктів на основі регіонів, продовжили розвиватися, з моделями, такими як Mask R-CNN. Mask R-CNN розширив Faster R-CNN, додаючи гілку сегментації, дозволяючи одночасно виявляти, класифікувати та сегментувати об’єкти на зображеннях як показано на рисунку 1.4.

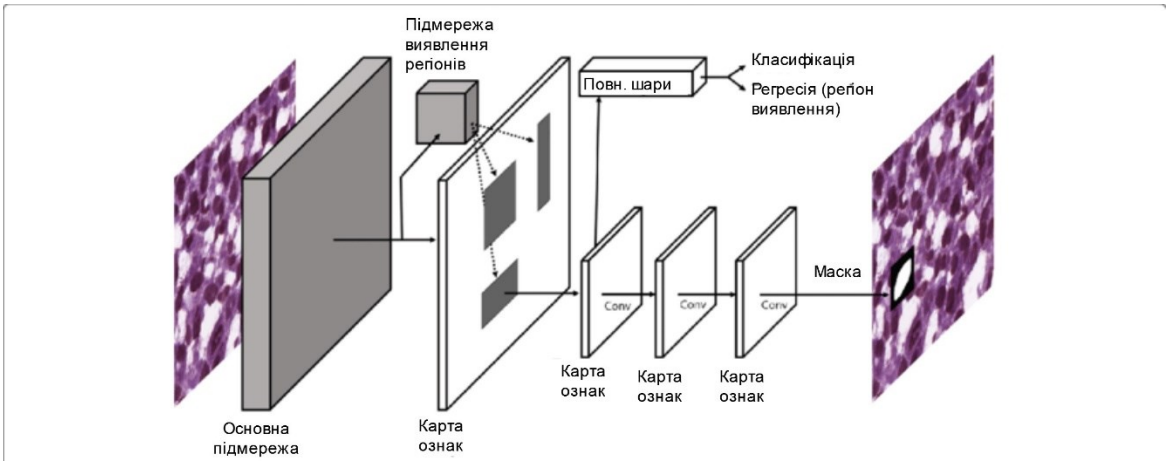

Рисунок 1.4 - Архітектура Mask R-CNN

1.1.4 Нейронні мережі для відстеження об’єктів

Напрямок відстеження об’єктів також зміг значно покращити точність в застосуваннях завдяки інтеграції машинного навчання і глибоких нейронних мереж. Традиційні методи відстеження об’єктів головним чином базувалися на підборі ознак та евристичних підходах. Ці методи стикалися з труднощами у роботі зі складними сценаріями, такими як затемнення, зміни масштабу та деформації об’єктів, проте поява машинного та глибокого навчання дозволила отримати моделі, інваріантні до таких сценаріїв.

Одним з ключових досягнень у виявленні об’єктів було введення Дискримінативних Кореляційних Фільтрів (DCF) у 2010 році, як показано на рисунку 1.5. Моделі для відстеження на основі DCF використовують машиннее навчання для створення дискримінативної моделі для цільового об’єкта.

Подальший розвиток відстеження пов’язаний з глибоким навчанням. У 2015 році було представлено використання кореляційних фільтрів на основі згорткових нейронних мереж. Цей метод продемонстрував здатність моделей глибокого навчання покращити точність відстеження за допомогою апроксимації функцій безпосередньо з даних [36].

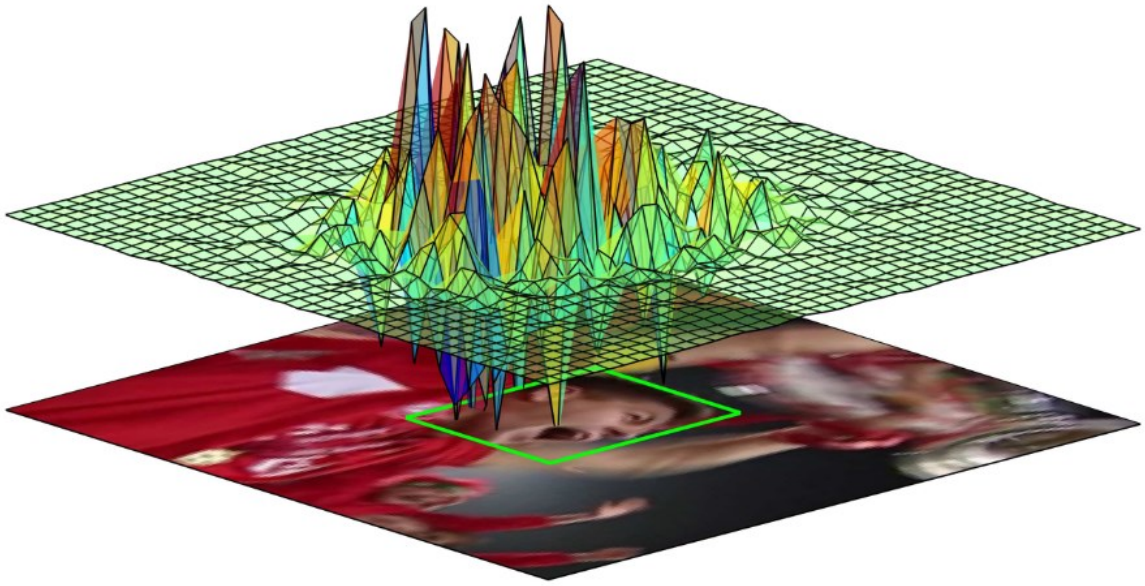

Рисунок 1.5 – Візуалізація коефіцієнтів DCF при використанні просторової регуляризації алгоритму SRDCF

Ще одним важливим кроком у виявленні об’єктів була розробка сіамських мереж. Сіамські мережі, які були представлені у 2016 році, широко використовуються для завдань відстеження. Ці мережі призначені визначати схожість між цільовим об’єктом та пропонованими областями, де може бути розташований об’єкт, що робить їх дуже ефективними для вирішення задачі відстеження об’єктів [37].

Алгоритм DeepSORT (Deep Simple Online and Realtime Tracking), представлений у 2017 році, поєднував глибоке навчання з алгоритмом відстеження SORT. DeepSORT показав покращення точності системи відстеження, зокрема в умовах складних сценаріїв, і впровадив інтеграцію опису зовнішності для кращого відстеження об’єктів у великих областях [38].

Ще одним значущим досягненням стала поява відстежувачів на основі мереж з довгою короткочасною пам’яттю (LSTM). LSTM - це тип рекурентної нейронної мережі, який може виявляти часові залежності у русі об’єктів. У 2018 році був запропонований відстежувач на основі LSTM, який продемонстрував високу точність відстеження, особливо в складних сценаріях з затемненнями та різкими рухами об’єктів [39].

Наразі активно використовуються підходи відстеження через виявлення об’єктів, поєднуючи виявлення об’єктів та їх відстеження в єдину систему. Впровадження засобів виявлення об’єктів на основі глибоких нейронних мереж, таких як Faster R-CNN та YOLO, великою мірою покращило точність методів відстеження через виявлення за рахунок надійної локалізації об’єктів, що є важливим для відстеження.

Завдяки великій кількості камер з високою роздільною здатністю, додатки для спостереження широко застосовуються в різних сферах, починаючи від моніторингу дорожнього руху і закінчуючи національною безпекою. Оскільки зображення та відео в реальному часі мають дуже великі розміри і високу частоту генерації, розгортання виділених оптоволоконних мереж є поширеним рішенням для підтримки високошвидкісної передачі медіа-даних для систем відеоспостереження.

Тим часом, з розвитком штучного інтелекту додатки для відеоспостереження інтегруються із сучасними механізмами машинного навчання для забезпечення розширених функцій, таких як виявлення об’єктів і відстеження цілей [40, 41]. Для того, щоб проаналізувати дані з величезної

кількості пристроїв, додатки можуть проектуватись з процесом обробки даних в хмарі, або у великих дата-центрах. У традиційній хмарній архітектурі кінцеві пристрої безпосередньо відправляють величезну кількість медіа-даних в режимі реального часу в хмару через виділені високошвидкісні оптоволоконні мережі, що призводить до високої вартості роботи таких систем [42].

1.2 Аналіз задачі оптимізації нейронних мереж

Використання глибоких нейронних мереж (DNN) набуло поширення у різних областях проте їх продуктивність значно залежить від параметрів моделі та обчислювальних витрат. Наприклад, широко використовувана мережа ResNet-50 вимагає пам'яті для зберігання понад 95 МБ, містить понад 23 мільйони навчальних параметрів і потребує 4 мільярди операцій з плаваючою точкою (GFLOP) для обчислень [43]; розмір мережі VGG-16, навченої на ImageNet, становить понад 500 МБ [44]; модель мережі архітектури трансформер GPT-3 містить до 175 мільярдів параметрів, а модель GPT-4 використовує відповідно ще більше параметрів [45]. Поточна тенденція до збільшення розміру нейронних зберігається, проте, чим більше параметрів у DNN, тим більше часу та пам'яті вони зазвичай потребують для обробки вхідних даних. Високі витрати обчислювальних ресурсів на навчання та використання, пов'язані з цими моделями, становлять значний виклик для їх впровадження на пристроях, які обмежені обчислювальними ресурсами (такими як ЦП, ГП та пам'ять), енергією та пропускною спроможністю. Наприклад, реальні застосування нейронних мереж, такі як автономний рух, рятувальні роботи та системи для запобігання лісовим пожежам, вимагають високої точності та ефективного використання ресурсів, включаючи швидку реакцію в режимі реального часу та малі вимоги до пам'яті. Велика обчислювальна складність та значний необхідний обсяг пам'яті

для виконання глибоких нейронних мереж можуть зробити їх непрактичними для використання на різних пристроях [46].

Обсяг даних, що генеруються пристроями Інтернету речей, різко збільшився, створюючи затримку при відправці і додаткове навантаження для хмарних обчислень. В умовах обмеженої пропускної здатності і зростаючого навантаження на Інтернет передача всіх даних в хмару для їх обробки більше не є розумним підходом для чутливих до затримок і великомасштабних додатків спостереження. Це мотивує використання кінцевих пристроїв для обчислень доповнити архітектуру хмарних обчислень для виконання обробки та аналізу медіа-даних в IoT [47, 48]. Використовуючи можливості зберігання і обробки даних, що надаються прилеглими інфраструктурами і пристроями, периферійні обчислення можуть надавати аналітичні сервіси для додатків відеоспостереження [49].

Головним викликом залишається обмеженість ресурсів на кінцевих пристроях. Навіть пристрої з вбудованими GPU мають досить невисоку пропускну здатність при виконанні сучасних моделей для виявлення об’єктів, таких як YOLOv3 (Таблиця 1.1).

Таблиця 1.1

Швидкість обробки (кадрів в секунду) на пристроях Nvidia Jetson

Size	Nano	TX2	NX
128×128	8.9	19.1	27.5
256×256	3.4	8.1	17.4
416×416	1.2	2.8	9.6
608×608	0.7	1.8	5.6

Зі зростанням популярності великих мовних моделей в останні роки зросло зацікавлення в стисненні нейронних мереж для комп’ютерів з гнучкими вимогами до апаратного забезпечення [50]. Крім того, глибокі нейронні мережі,

які містять надлишкові параметри, можуть порушити їх стійкість до шумів, підвищуючи ризик адверсивних атак [51].

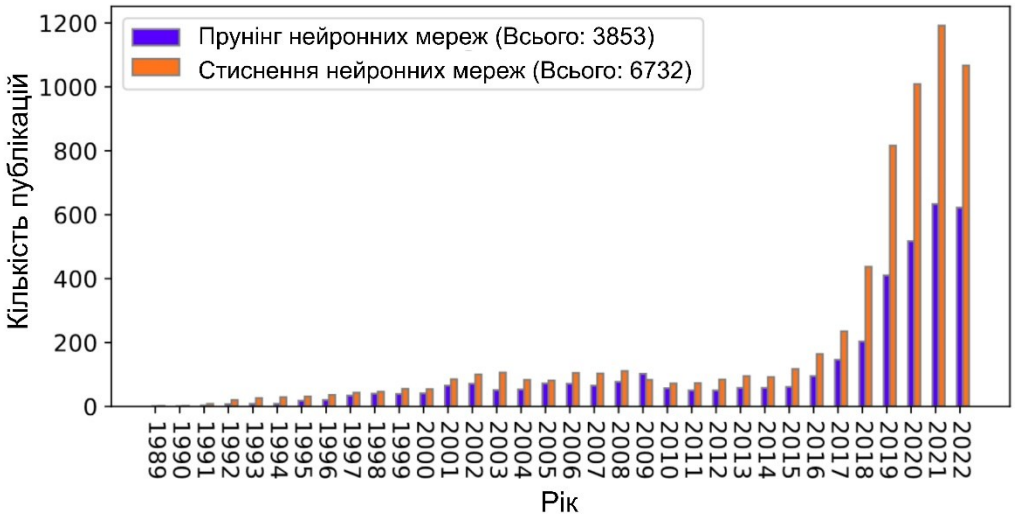

Рисунок 1.6 – Зростання кількості статей про прунінг і стиснення мереж за 1989 – 2022 роки

Для вирішення цієї проблеми було запропоновано різні техніки стиснення нейронних мереж для створення легких моделей, включаючи прунінг нейронної мережі, низькорангові розкладання матриць ваг, квантизацію, дистиляція знань [52]. Зростає інтерес до прунінгу нейронних мереж, яке було доведено як ефективний спосіб ефективного використання обсягу пам’яті та часу обчислень, при цьому забезпечуючи відповідну або навіть кращу продуктивність порівняно початковими моделями. Як показано на рисунку 1.6, кількість статей про прунінг значно збільшилася з 2015 по 2022 рік. Вони становлять більше половини статей про стиснення нейронних мереж [53].

1.2.1 Методи низькорангової факторизації

Методи низькорангової факторизації є фундаментальним підходом до стиснення нейронних мереж, спрямованим на зменшення вимог до обчислень і пам’яті глибоких нейронних мереж (DNN), зберігаючи або навіть покращуючи їх

продуктивність. Ця техніка використовує концепцію того, що вагові матриці в нейронних мережах часто містять значну кількість надмірності. Апроксимуючи ці вагові матриці факторизаціями нижчого рангу, можна представити ту саму інформацію з меншою кількістю параметрів.

Розкладання за сингулярним значенням (SVD) — це математичний метод, який широко використовується для низькорангової факторизації. SVD розкладає матрицю на три інші матриці: U, Σ і V, де U і V — ортогональні матриці, а Σ — діагональна матриця, що містить сингулярні значення вихідної матриці. Ранг матриці можна зменшити, зберігаючи лише перші k сингулярних значень та їхні відповідні стовпці в U та рядки у V. Це призводить до нижчого рангу наближення вихідної матриці, фактично зменшуючи кількість параметрів.

Для застосування SVD для стиснення нейронних мереж було створено підхід тензорної декомпозиції для факторизації вагових тензорів у згорткових шарах [54]. Вагові тензори в згорткових шарах є високорозмірними, що вимагає більшого об’єму ресурсів для їх обчислення. Застосовуючи тензорне розкладання, було показано, що можна апроксимувати ці вагові тензори за допомогою комбінації менших тензорів, значно зменшуючи як розмір моделі, так і вимоги до обчислень.

Тензорна декомпозиція є розширенням SVD, яке зосереджується на структурах даних вищої розмірності, таких як тензори. Ідея полягає в тому, щоб розкласти тензор на суму простіших тензорів, як правило, з меншими розмірами. CPD (канонічний поліадичний розклад, Canonical polyadic decomposition) і розклад Такера є широко використовуваними методами тензорної факторизації. Ці методи особливо ефективні при роботі з ваговими тензорами в згорткових шарах, які за своєю суттю є багатовимірними [55, 56].

Ще один важливий аспект методів факторизації низького рангу полягає в тому, що їх можна застосовувати не лише до вагових матриць, але й до інших частин нейронних мереж, таких як вагові фільтри в згорткових шарах. У 2016

році було досліджено використання факторизації низького рангу у мережах LSTM. Застосовуючи апроксимації низького рангу до вагових матриць у LSTM, автори досягли зменшення розміру моделі та обчислювальних витрат, при мінімальних втратах точності[57].

Переваги методів факторизації низького рангу виходять за межі зменшення розміру моделі. Вони також можуть призвести до пришвидшення обчислень, що робить їх придатними для застосувань в реальному часі, де низька затримка є вирішальною. Крім того, зменшений обсяг пам'яті дозволяє розгортати ці стиснені моделі на пристроях з обмеженими ресурсами та обмеженим об'ємом пам'яті, наприклад на периферійних і мобільних пристроях. Втім, кількість архітектур, на яких проводились дослідження, досить обмежена, і наявні методи не набули широкого поширення в індустрії.

1.2.2 Квантування

Квантування — це фундаментальна техніка в області стиснення нейронних мереж, призначена для зменшення обсягу пам'яті та обчислювальної складності глибоких нейронних мереж (DNN) шляхом представлення числових значень із типами даних нижчої точності, такими як цілі числа, замість традиційних 32-розрядних чисел з плаваючою комою. Цей підхід особливо ефективний для розгортання DNN на пристроях з обмеженими ресурсами, оскільки він значно зменшує вимоги до пам'яті та прискорює виконання, мінімізуючи вплив на точність моделі.

Концепція квантування висуває ідею, що не всі числові значення в нейронній мережі потрібно представляти з високою точністю. У багатьох випадках для підтримки бажаного рівня точності достатньо використовувати представлення з меншою точністю. Квантування можна застосовувати як до ваг (параметрів), так і до активацій (проміжних значень) у нейронній мережі.

У 2014 було запропоновано двійкові та потрійні методи вагового квантування, де вагові коефіцієнти нейронної мережі мають двійкові (-1 або 1) або потрійні (-1, 0 або 1) значення. Зменшивши точність ваг, було досягнуто значного зменшення розміру моделі та вимог до обчислень. Крім того, було представлено концепцію мереж із квантованими вагами, і продемонстровано, що навчання з урахуванням квантування може створювати високоефективні моделі [58, 59, 60].

Одним із фундаментальних аспектів квантування є вибір відповідної розрядності для представлення числових значень. Загальні розрядності, що використовуються в квантуванні, включають 8-бітове та 16-бітне представлення з фіксованою точкою. Наприклад, у 8-розрядному представленні з фіксованою точкою значення обмежені діапазоном із 256 дискретних рівнів як показано на рисунку 1.7. Компроміс полягає у виборі такої бітової ширини, яка врівноважує зменшення вимог до пам’яті та обчислень із збереженням точності моделі.

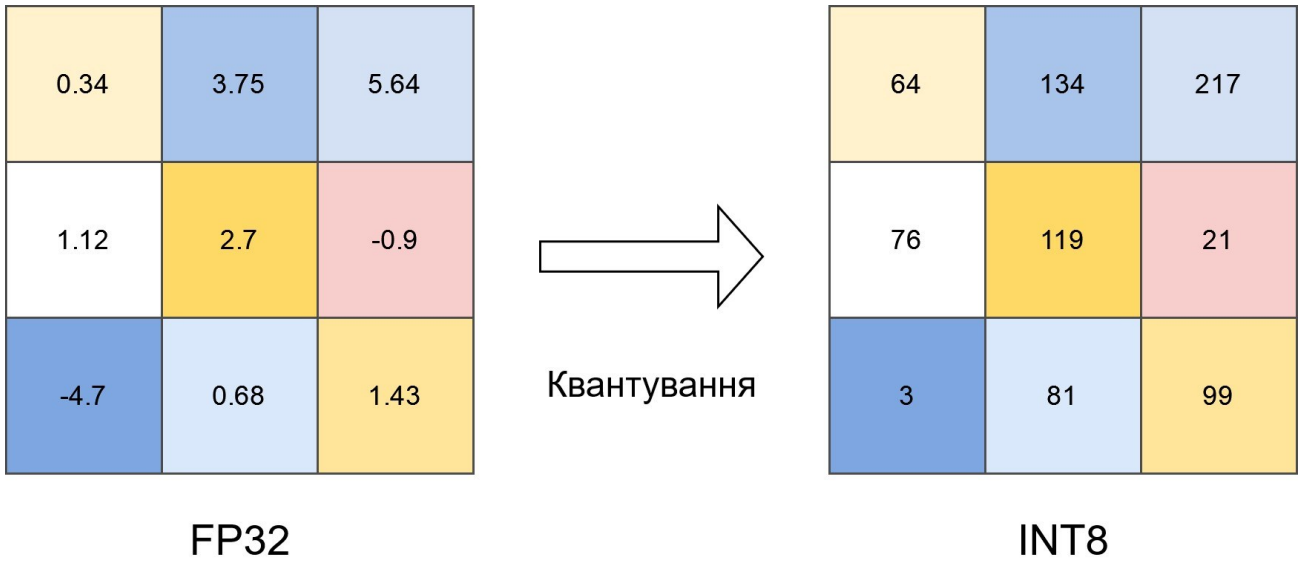

Рисунок 1.7 - Приклад застосування квантування для переведення у 8-розрядне представлення

Динамічне квантування адаптує точність числових значень на основі їх динамічного діапазону під час висновку. Наприклад, значення з вузьким динамічним діапазоном можуть бути представлені з нижчою точністю, тоді як значенням із ширшим динамічним діапазоном надається вища точність. Динамічне квантування допомагає максимізувати ефективність, використовуючи мінімальну необхідну точність для кожного значення.

Методи квантування не обмежуються представленнями з фіксованою точкою. Методи квантування з плаваючою точкою, такі як пониження точності (наприклад, половинна точність, float16), спрямовані на досягнення балансу між ефективністю пам’яті представлень із фіксованою точкою та достовірністю передачі інформації числами із плаваючою точкою повної точності.

Останні публікації зосереджені на оптимізації процесу квантування та розробці апаратних прискорювачів, які можуть ефективно виконувати обчислення з квантованими значеннями. Спеціалізоване апаратне забезпечення, таке як модулі обробки тензорів (TPU) і програмовані вентильні матриці (FPGA), може бути розроблено для прискорення виконання за допомогою квантованих моделей, ще більше підвищуючи ефективність розгортання нейронної мережі.

1.2.3 Дистиляція знань

Дистиляція знань — це складна техніка в області стиснення нейронної мережі та оптимізації моделі. Вона спрямована на передачу знань від великої складної мережі вчителя (яку часто називають моделлю вчителя) до меншої та більш ефективної мережі учнів (модель учня) як показано на рисунку 1.8. Цей процес дозволяє моделі учня імітувати продуктивність моделі вчителя, використовуючи при цьому знижені вимоги до пам’яті та обчислень. Дистиляція знань набула значного поширення, оскільки вона дозволяє розгортати ефективні моделі в середовищах з обмеженими ресурсами з мінімальною втратою точності.

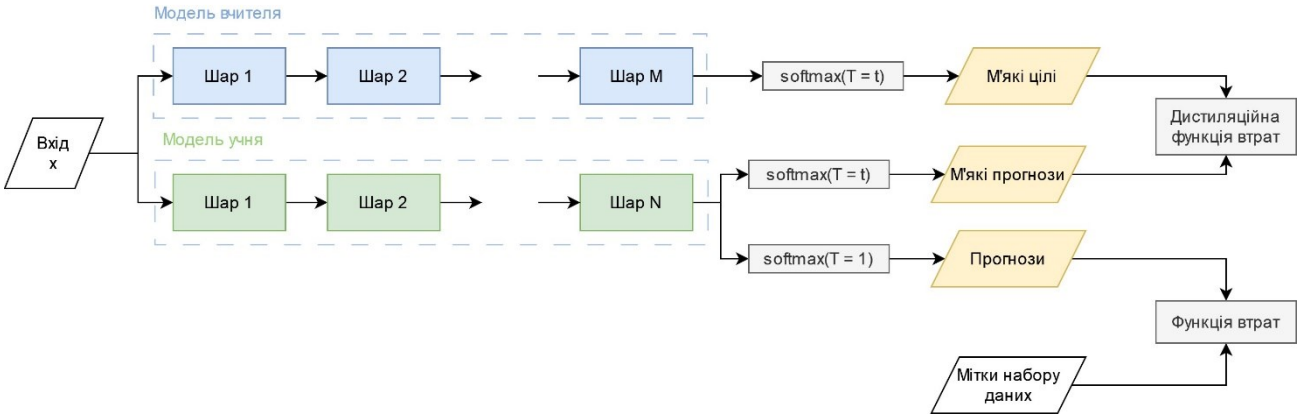

Рисунок 1.8 - Приклад дистиляції знань.

Ідею методу дистиляції знань було закладено у 2015 році, де було описано процес передачі знань від добре навченої великої нейронної мережі до меншої. Аргумент на користь використання методу полягав в тому, що модель вчителя надає не лише передбачення, але й інформацію про зв'язки та невизначеності в даних, що можна використовувати для ефективного навчання моделі учня [61].

Процес передачі знань зазвичай складається з двох основних етапів:

1. Навчання моделі вчителя - навчання більш складної і глибокої нейронної мережі (моделі вчителя) на цільовому завданні з досягненням високої точності. Ця модель служитиме джерелом знань, які будуть передані в модель учня.

2. Навчання моделі учня - навчання меншої нейронної мережі (моделі учня) для того самого завдання, але з додатковою корекцією прогнозів від моделі вчителя. Мета полягає в тому, щоб переконатися, що прогнози моделі учня узгоджуються з прогнозами моделі вчителя.

Для полегшення дистиляції знань використовується кілька методів:

- Замість того, щоб використовувати закодовані мітки як цілі під час навчання, розподіл імовірності в кінцевому шарі моделі вчителя по класах використовується як «м'які цілі» для моделі учня. Такий підхід вносить рівень

невизначеності та допомагає краще передати схожіть між класами в процесі навчання, полегшуючи навчання моделі учня.

- Дистиляція знань зазвичай передбачає використання додаткової функції дистиляційних втрат на додаток до стандартної функції втрат (наприклад, крос-ентропії). Функції дистиляційних втрат спонукають модель учня відповідати прогнозам моделі вчителя. Загальні функції дистиляційних втрат включають розбіжність Кульбака-Лейблера (KL) і середньоквадратичну помилку (MSE) між прогнозами викладача та студента.

- Температурний параметр вводиться для контролю ступені пом’якшення прогнозів моделі вчителя. Більш високі температури призводять до більш м’якого розподілу, при цьому даний параметр часто коригується під час навчання, щоб точно налаштувати процес передачі знань на пізніх епохах.

Дистиляція знань була успішно застосована в різних областях і програмах. Наприклад, даний метод було використано для стиснення великомасштабних моделей класифікації зображень у більш ефективні версії. Одним із яскравих прикладів є приклад дистиляції знань із більшої моделі в архітектуру MobileNet, що призвело до створення високоефективної, але точної моделі, придатної для використання на мобільних і вбудованих пристроях [62]. Також, в сфері обробки природніх мов було представлено дистиляцію знань із великої моделі BERT для створення меншого варіанту під назвою TinyBERT [63]. Також, було випробувано підхід з використанням ансамблю моделей в якості моделі вчителя для тренування меншої моделі архітектури MobileNetV2 [64].

Крім того, дистиляцію знань можна застосовувати ітеративно, створюючи ієрархію моделей різного розміру та складності, що дозволяє точно контролювати розмір моделі та компроміси продуктивності [65].

1.2.4 Пошук архітектури

Пошук нейронної архітектури (NAS) — це метод у сфері глибокого навчання та проектування нейронних мереж, що являє собою зміну парадигми в тому, як створюються та оптимізуються архітектури нейронних мереж. Замість того, щоб покладатися на ручне проектування та експертні знання, NAS автоматизує процес пошуку найбільш ефективної архітектури нейронної мережі для певного завдання.

Концепція NAS набула значного поширення у 2017 році. Було представлено дослідження з ідеєю використання навчання з підкріпленням для пошуку архітектур нейронної мережі. Замість того, щоб створювати архітектури вручну, NAS використовує алгоритми пошуку, щоб виявити архітектури, які відмінно справляються з конкретними завданнями як показано на рисунку 1.9 [66].

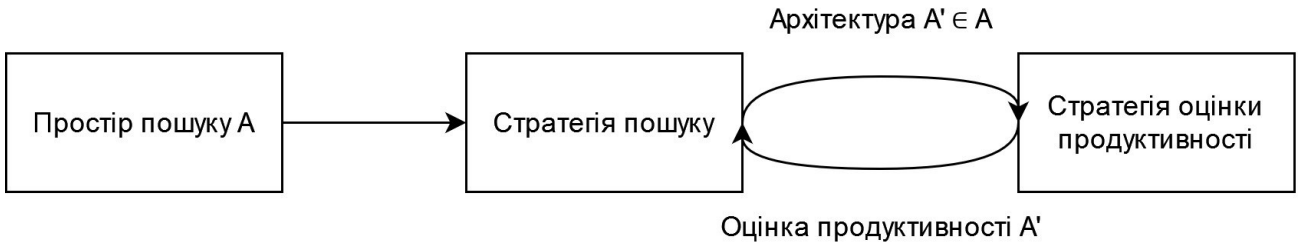

Рисунок 1.9 - Приклад роботи системи NAS

Процес NAS зазвичай включає такі кроки:

- Першим важливим кроком є визначення простору пошуку можливих архітектур нейронної мережі. Цей простір охоплює різні архітектурні рішення, включаючи кількість шарів, типи шарів (згортковий, рекурентний тощо), розміри ядра, функції активації тощо. Простір пошуку може варіюватися від простих архітектур до дуже складних.

- NAS використовує алгоритми пошуку для навігації у просторі всіх можливих архітектур та визначення найбільш перспективних архітектур. Ці алгоритми можуть включати навчання з підкріпленням, генетичні алгоритми, еволюційні стратегії тощо. Вибір алгоритму впливає на ефективність і ефективність NAS;

- під час процесу пошуку архітектури-кандидати оцінюються на основі їхньої продуктивності в конкретному завданні або набір даних для порівняння. Точність, ефективність та інші відповідні показники керують вибором перспективних архітектур;

- NAS часто використовує навчання з підкріпленням, щоб вивчати та адаптувати свою стратегію пошуку з часом. Агент прагне максимізувати функцію винагороди, який зазвичай відповідає продуктивності виявлених архітектур. По ходу пошуку агент уточнює свою стратегію, щоб досліджувати більш перспективні області простору пошуку.

Одним із помітних аспектів NAS є його здатність відкривати нетрадиційні та високоефективні архітектури нейронних мереж. Ці архітектури можуть включати нові комбінації шарів або операцій, які могли бути опущені при проектуванні вручну.

Одним із проривів у NAS є створення ефективних нейронних мереж, зокрема у сфері комп’ютерного зору, зокрема архітектури EfficientNet у 2019 році. EfficientNet досягла найсучаснішої точності на ImageNet, одночасно значно зменшивши кількість параметрів і вимог до обчислень порівняно з попередніми архітектурами. Це стало можливим завдяки систематичному пошуку ефективних архітектурних конфігурацій з використанням NAS [67].

Ще одним важливим досягненням у NAS є врахування апаратних обмежень. Дослідники інтегрували оптимізацію з урахуванням апаратного забезпечення в NAS, щоб створити архітектури, які не тільки точні, але й адаптовані до конкретних апаратних платформ. Це особливо важливо для

використання нейронних мереж на кінцевих пристроях. Було представлено метод, який враховує як цільове завдання, так і апаратну платформу під час пошуку архітектури, і дозволяє тренувати високоефективні моделі [68].

1.3 Аналіз прунінгу для оптимізації нейронних мереж на різних стадіях розрбки

Прунінг нейронних мереж – це метод глибокого навчання, що спрямований на зменшення обсягу та обчислювальної складності нейронних мереж, при цьому зберігаючи їхню ефективність. Цей метод має чотири основні підходи: прунінг на етапі проектування, прунінг перед навчанням, прунінг під час навчання та прунінг після навчання. В даному підрозділі розглядаються ці категорії та надаються відомості про їхню методологію та дослідження, які ілюструють кожен з підходів.

1.3.1 Прунінг на етапі проектування

Прунінг на етапі проектування, також відомий як структурований прунінг або архітектурний прунінг, передбачає створення архітектур нейронних мереж з урахуванням розрідженості вже на етапі їхнього проектування. Мета - створити мережі, які природно виявляють розріджені зв’язки, роблячи їх більш ефективними.

Одиним з прикладів є техніка структурованого прунінгу для згорткових нейронних мереж (CNN), шляхом визначення структурної надлишковості фільтрів. Дослідження підкреслило переваги створення архітектур, які вже включають розрідженість, що призводить до зменшення обсягу пам’яті та кількості обчислень [69].

Іншим прикладом є архітектура MobileNet. MobileNet розроблено з використанням згорткових шарів з окремою глибиною, що автоматично зменшує кількість параметрів і обчислень, що робить її ефективною при використанні для обмежених за ресурсами середовищ, таких як мобільні пристрої. Такий підхід при проектуванні архітектури допомагає забезпечити широке застосування MobileNet для обчислень в реальному часі на мобільних платформах [70].

1.3.2 Прунінг перед навчанням

У сфері оптимізації нейронних мереж перед навчанням найвідомішими копцепціями є гіпотеза лотерейного квитка та алгоритм SNIP (Single-shot Network Pruning - одномоментна мережева обрізка). Ці підходи вирішують проблеми обчислювальної складності та надмірної параметризації в моделях глибокого навчання, пропонуючи шляхи до більш ефективних мережевих архітектур.

Прунінг перед навчанням являє собою метод оптимізації нейронних мереж, що характеризується прогностичним підходом до виявлення та усунення надлишкових або менш значущих нейронів і зв’язків у мережі. Основна мета полягає в тому, щоб передбачити, які зв’язки або нейрони зроблять мінімальний внесок у навчання та продуктивність мережі. Випереджальне відсікання цих елементів дозволяє ефективно зменшити розмір і складність мережі, що може призводити до підвищення обчислювальної ефективності без суттєвого зниження точності.

Гіпотеза лотерейного квитка, забезпечує теоретичне підґрунтя, яке доповнює практичне застосування прунінгу перед навчанням. Ця гіпотеза стверджує, що в межах випадково ініціалізованої щільної нейронної мережі існують менші підмережі (так звані "виграшні квитки"), які, якщо їх навчати ізольовано з самого початку, можуть досягти порівнянної точності з вихідною мережею, але зі значно меншою кількістю параметрів. Вважається, що ці

«виграшні квитки» мають ініціалізацію, яка робить їх особливо сприятливими для ефективного навчання [71].

Спираючись на принципи прунінгу перед навчанням та гіпотезу лотерейного квитка, алгоритм SNIP стає практичним інструментом для оптимізації мережі. SNIP розшифровується як Single-shot Network Pruning, і, як випливає з назви, він передбачає одномоментний процес обрізання, що застосовується до початку навчання. Алгоритм працює, оцінюючи чутливість функції втрат до видалення окремих з'єднань, ефективно оцінюючи внесок кожного з'єднання в продуктивність мережі. З'єднання з мінімальним впливом на функцію втрат обрізаються, в результаті чого мережа стає більш розрідженою [72].

1.3.3 Прунінг під час навчання

Прунінг під час навчання, також відомий як динамічний прунінг, передбачає поступове видалення менш важливих зв'язків під час процесу навчання. Мета полягає в ідентифікації та вилученні надлишкових зв'язків під час навчання, що призводить до створення обрізаної мережі.

У 2015 було впроваджено концепцію прунінгу під час навчання шляхом одночасної оптимізації ваг та зв'язків. Було запропоновано метод, який ідентифікує незадіяні ваги та обрізає їх під час навчання, що призводить до розрідженої мережі. Цей підхід продемонстрував значне зменшення обсягу моделі та обчислень при збереженні точності [73].

1.3.4 Прунінг після навчання

Прунінг після навчання є широко використовуваним підходом, де натренеровані моделі піддаються обрізці для зменшення їхнього обсягу та обчислювальних витрат, при цьому зберігаючи їхню точність.

Одним із прикладів є запропонована техніка прунінгу на рівні фільтрів для CNN. Було проведено ряд досліджень для виявлення та обрізання фільтрів, які мають менший внесок у продуктивність мережі. Під час досліджень було продемонстровано, що можна досягти значного стиснення моделі з мінімальним втратами точності [74].

1.4 Постановка наукового завдання дослідження та часткових завдань

Проведений аналіз методів вирішення задач розпізнавання образів показав, що методи, що базуються на основі нейронних мереж змогли суттєво покращити точність в цих задачах. Нейронні мережі успішно застосовуються для класифікації зображень, виявлення і відстеження об’єктів. Нейронні мереж згорткової архітектури дозволяє тренувати глибокі мережі з ієрархічною структурою виділення ознак. Обсяги даних збільшуються експоненційно, дозволяючи тренувати все глибші моделі, що містять мільярди параметрів у вигляді вагових коефіцієнтів.

Паралельно з цим, активно розвивається індустрія IoT та вбудованих пристроїв. Використання нейронних мереж на кінцевих пристроях потребує стиснення і оптимізації нейронних мереж для оптимального використання обмежених обчислювальних ресурсів і пам’яті при досягненні бажаної швидкодії.

Тренування глибоких нейронних мереж є ресурсоємним процесом, і для тренування сучасних архітектур на великих наборах даних залучаються великі дата-центри і кластери потужних комп’ютерів з GPU. Такі експерименти мають значну вартість і можуть мати негативний вплив на навколишнє середовище, і суттєво обмежують доступність технології.

Існуючі методи стиснення і оптимізації дозволяють скоротити число параметрів нейронної мережі. Методи низькорангової факторизації використовують концепцію того, що вагові матриці в нейронних мережах часто містять значну кількість надмірності. Апроксимуючи ці вагові матриці факторизаціями нижчого рангу можна досягти зменшення кількості параметрів мережі. Квантування дозволяє зменшити обчислювальну складність нейромереж шляхом представлення числових значень із типами даних нижчої. Попри те що квантування і низькорангова факторизація є досить ефективними методами, квантування потребує вже натренованої мережі, а низькорангова факторизація вагових матриць часто призводить до вагомих втрат точності мережі. Методи прунінгу дозволяють виділити значущі параметри нейронної мережі на різних етапах навчання, в тому числі і перед навчанням мережі, до того ж можуть комбінуватись і з іншими методами, такими як квантування, втім самі по собі не демонструють реального збільшення швидкодії.

В результаті аналізу існуючих досліджень було виявлено, що розроблені методи зменшення параметрів і збільшення швидкодії надають поодинокі програмні реалізації, що не дає змогу оцінити вплив на швидкодію в порівнянні з іншими методами. У зв’язку з цим виникає потреба в побудові архітектури системи, що має наступні властивості:

- здатність поповнювати бібліотеку методів зменшення параметрів мереж;
- здатність працювати з різними наборами даних та архітектурами мереж;
- розгортання навчання в хмарному середовищі для великих моделей;
- містити реалізації методів збільшення швидкодії для сучасних CPU і GPU;
- здатність зберігати метрики для порівняння методів і архітектур;
- мати розширювану та масштабовану архітектуру;
- містити модуль автентифікації для різних користувачів.

Окреслене сформувало протиріччя, коли з одного боку, нейронні мережі є ефективним і перспективним інструментом для вирішення задач розпізнавання образів, і з насиченням даними і появою нових глибоких архітектур дозволяють отримати кращі результати при вирішенні багатьох задач, з іншого боку, нейронні мережі потребують великий обсяг обчислювальних ресурсів для тренуванні і запуску, і відповідно, методів оптимізації при використанні на кінцевих пристроях для досягнення бажаної швидкодії. Таке протиріччя може бути вирішено за допомогою розробки методів та програмних засобів підвищення швидкодії моделей розпізнавання образів на основі машинного навчання.

У рамках даного дослідження важливо врахувати певні обмеження, пов’язані із доступністю даних, а також з доступними обчислювальними ресурсами для тренування мереж. Розробка і обчислювальні експерименти проводились на потужностях у розпорядженні здобувача, використовуючи дані, доступні виключно у відкритих джерелах.

Метою даного дисертаційного дослідження є збільшення ефективності моделей нейронних мереж, а саме збільшенні швидкодії при мінімальній втраті точності, шляхом застосування методів оптимізації моделей глибинного навчання, створених для вирішення задач розпізнавання образів.

Об’єктом досліджень є методи та підходи оптимізації навчання нейронних мереж для вирішення задач розпізнавання образів з метою підвищення швидкодії.

Предметом досліджень є оптимізація процесу навчання нейронних мереж для для вирішення задач розпізнавання образів з метою підвищення швидкодії.

Для досягнення мети в дисертації вирішено такі **наукові завдання:**

1. Виконати аналіз існуючих методів оптимізації нейронних мереж.
2. Дослідити алгоритми тренування моделей НМ, способи ініціалізації вагів нейронних мереж і підходи до вирішення задач розпізнавання образів.

3. Дослідити ефективність існуючих методів оптимізації на різних етапах розробки нейронних мереж.

4. Вдосконалити метод для оптимізації нейронних мереж для вирішення задач розпізнавання облич.

5. Розробити метод для оптимізації нейронних мереж архітектури трансформер для вирішення задач розпізнавання образів.

6. Провести експерименти з тренування та оцінки точності для підтвердження ефективності розробленого методу на моделі нейронної мережі для вирішення задач розпізнавання образів.

7. Розробити архітектуру системи для проведення експериментів і порівняння методів оптимізації нейронних мереж оцінки швидкодії, що призначені для вирішення задач розпізнавання образів.

Висновки до розділу 1

Розділ містить аналіз використання нейронних мереж для вирішення задач в області комп’ютерного зору і розпізнавання образів, використовуючи методи машинного навчання. Нейронні мережі розглянуті як інструмент, що дозволяє досягти кращих результатів при вирішення задач комп’ютерного зору, втім, обчислювальна складність мереж робить їх тренування і практичне застосування обмеженим.

Проаналізовано основні архітектури нейронних мереж для вирішення задач комп’ютерного зору, і типові застосування таких мереж. Проведений аналіз показав необхідність використання методів оптимізації для підвищення швидкодії нейронних мереж, а також зменшення вимог до обчислювальної потужності, особливо в умовах обмежених обчислювальних ресурсів, таких як предметна область IoT.

Аналіз наявних методів оптимізації показав, що в області оптимізації нейронних мереж після тренування наявний ряд сталих методів, що використовуються для підвищення швидкодії з допустимою втратою точності. Разом з тим, перспективним є напрям нейромережевого прунінгу перед навчанням, оскільки дані методи дозволяють визначати і видаляти найменш важливі звязки ще до початку навчання, потенційно спрощуючи мережу і зменшуючи час тренування.

В результаті аналізу методів прунінгу перед навчанням і наявних прикладів використання було виявлено потребу в адаптації даних методів до нових архітектур нейронних мереж, зокрема мереж архітектури трансформер.

2. ВДОСКОНАЛЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ НЕЙРОННИХ МЕРЕЖ З ВРАХУВАННЯМ ОСОБЛИВОСТЕЙ АРХІТЕКТУР ДЛЯ ВИРІШЕННЯ ЗАДАЧ РОЗПІЗНАВАННЯ ОБРАЗІВ

2.1 Дослідження підходів до вирішення задачі виявлення облич

Задача виявлення облич полягає у виявленні людських облич у зображеннях та поверненні просторових розташувань обличчя за допомогою обмежувальних рамок, як показано на рисунку 2.1.

Рисунок 2.1 – Приклади вирішення задачі виявлення облич

Це одне з найпоширеніших основних та практичних завдань в області комп'ютерного зору. З моменту появи детектора Віоли-Джонса у 2001 році, алгоритми з виявлення обличчя значно покращились в тому числі завдяки переходу від ручного створення ознак, таких як Хаар-подібні ознаки, до наскрізних (end-to-end) згорткових нейронних мереж (CNN) та архітектур трансформер. Особливість наскрізних архітектур полягає в тому, що замість ручного проєктування та конструювання людиною ознак наскрізна модель вчиться витягувати відповідні ознаки безпосередньо з даних та на основі цього вирішувати конкретну задачу.

Вирішення задачі виявлення обличчя є першим кроком у ряді практичних рішень, пов'язаних з обробкою облич, таких як розпізнавання обличчя, відстеження обличчя, розпізнавання виразів обличчя, розпізнавання координат та орієнтації обличчя тощо. Тому загалом точність та швидкість роботи таких рішень залежить від відповідних властивостей детектору обличчя, що входить до складу такого рішення.

До початку періоду широкого застосування глибокого навчання та глибоких нейронних мереж відповідно, домінуючим методом для вирішення задачі виявлення об'єктів був каскадний класифікатор AdaBoost [75]. Деякі алгоритми були спеціально розроблені для задачі виявлення обличчя, використовуючи певні види ознак, таких як Хаар-подібні ознаки, SURF та Multi-Block LBP [76, 77]. Глибоке навчання своїм розвитком довело ефективність у вирішенні задачі створення ознак та досягло кращої точності за класичні підходи виявлення об'єктів на зображеннях.

Глибокі нейронні моделі переважно створювалися для загальної задачі виявлення багатьох об'єктів різних класів. Ця задача є складнішою, ніж задача виявлення обличчя, оскільки вирішується задача знаходження об'єктів багатьох класів, а не одного класу. Для вирішення задачі виявлення обличчя ряд моделей

для виявлення об’єктів було адаптовано для виявлення об’єкту конкретного класу - обличчя.

Використовуючи такий підхід можна натренувати нейронну модель для виявлення обличчя, використовуючи Faster R-CNN YOLO або SSD, які попередньо були створені для вирішення задачі виявлення об’єктів багатьох класів. Цей підхід дозволяє отримувати набагато вищі показники метрик точності виявлення обличчя, ніж при використанні традиційних каскадних класифікаторів. Прикладами таких адаптацій моделей під конкретну задачу є Face R-CNN та Face R-FCN, які модифіковані та вдосконалені на основі Faster R-CNN, R-FCN відповідно [78, 79]. Деякі інші детектори, такі як MTCNN, SSH, спеціально розроблені для виявлення обличчя на зображенні [80,81].

Також варто зауважити, що деякі техніки, що застосовувались для загального виявлення об’єктів на зображеннях, також були модифіковані для задачі виявлення обличчя, такі як механізм multi-scale від SSD, покращення ознак від FPN та функція втрат від RetinaNet [82, 83]. Модифіковані техніки призвели до створення більш сучасних детекторів обличчя, таких як RetinaFace.

Спостерігається, що протягом останніх років відбувається зростання попиту на периферійні пристрої, такі як смартфони та камери спостереження. Наслідком цього є те, що кожного дня генерується величезна кількість даних; користувачі роблять селфі, фотографії інших людей, проводять тривалі відеоконференції тощо. Сучасні камери спостереження записують відео з роздільною здатністю 1080P та сталою частотою 30 кадрів в секунду. Зростання попиту і використання таких периферійних пристроїв призвело до зростання попиту на аналіз даних, особливо даних пов’язаних з обличчями, при цьому треба зауважити, що обсяг цих даних є дуже великим. Периферійні пристрої мають обмежену обчислювальну здатність, обсяг пам’яті та тривалість роботи від батареї, що є значним викликом для запуску передових алгоритмів на основі глибокого навчання. Маючи ці обмеження для периферійних пристроїв

підвищення ефективності виявлення облич є важливим напрямком для ряду застосунків обробки зображень та відео з обличчями на периферійних пристроях.

У даній роботі для експериментів було використано архітектуру RetinaFace [84]. Опис особливостей архітектури та обгрунтування використання зазначено у наступному підрозділі 2.2.

2.2 Дослідження та обґрунтування вибору архітектури RetinaFace

Модель RetinaFace — це сучасний алгоритм виявлення обличчя, що базується на підході глибокого навчання. Він не лише ідентифікує обличчя на зображеннях, але і в реальному часі виводить високоякісні ознаки обличчя, такі як положення очей, носа та рота. Ця модель була розроблена з метою усунення обмежень існуючих алгоритмів виявлення обличчя, які мають труднощі при роботі з обличчями в різних умовах, таких як різні орієнтації, розміри та роздільні здатності.

Підхід має 3 ключові особливості: мережу піраміди ознак, одноетапність (на противагу мультиетапним підходам) та контекстний модуль (рисунок 2.2).

Одноетапні методи, на відміну від двоетапних методів (наприклад, Faster R-CNN), вимагають лише однієї ітерації по всій мережі для створення обмежувальних рамок об’єкта, який потрібно виявити. Це робить його набагато ефективнішим.

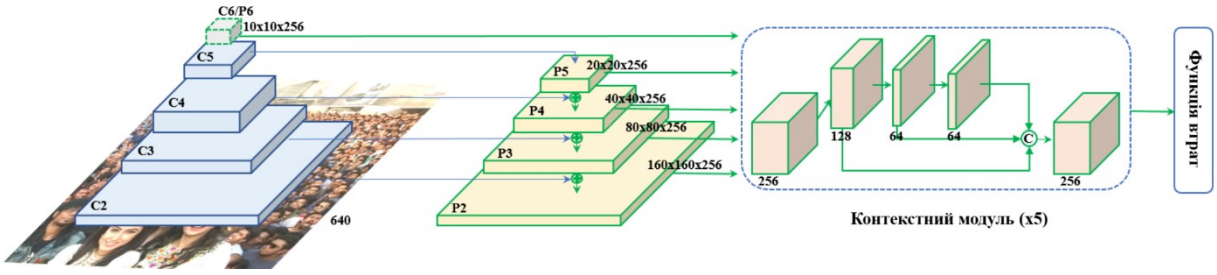

Рисунок 2.2 - Схематичне зображення архітектури

RetinaFace використовує згорткову нейронну мережу (CNN) як свою основну мережу (backbone network). Зазвичай вибирають ResNet або MobileNet. Основна мережа служить для витягування ознак з вхідного зображення, які потім передаються в мережу піраміди ознак (FPN). FPN є важливим для захоплення представлень для об’єктів різного масштабу та комбінації ознак з різних рівнів основної мережі. Такі архітектурні властивості є суттєвим фактором у виявленні обличчя, оскільки обличчя можуть з’являтися в різних розмірах і на різних відстанях від камери.

Після блоку FPN ознаки подаються на вхід до контекстного модуля (context module). Ідея цього модуля полягає у об’єднанні карт ознак з різних рівнів FPN мережі. На основі такого підходу покращується локалізація та класифікація облич в контексті інваріантності щодо масштабу обличчя. Крім цього, в мережі використовуються деформовані згорткові шари (Deformable convolution layers, DC), принцип роботи яких зображено на рисунку 2.3.

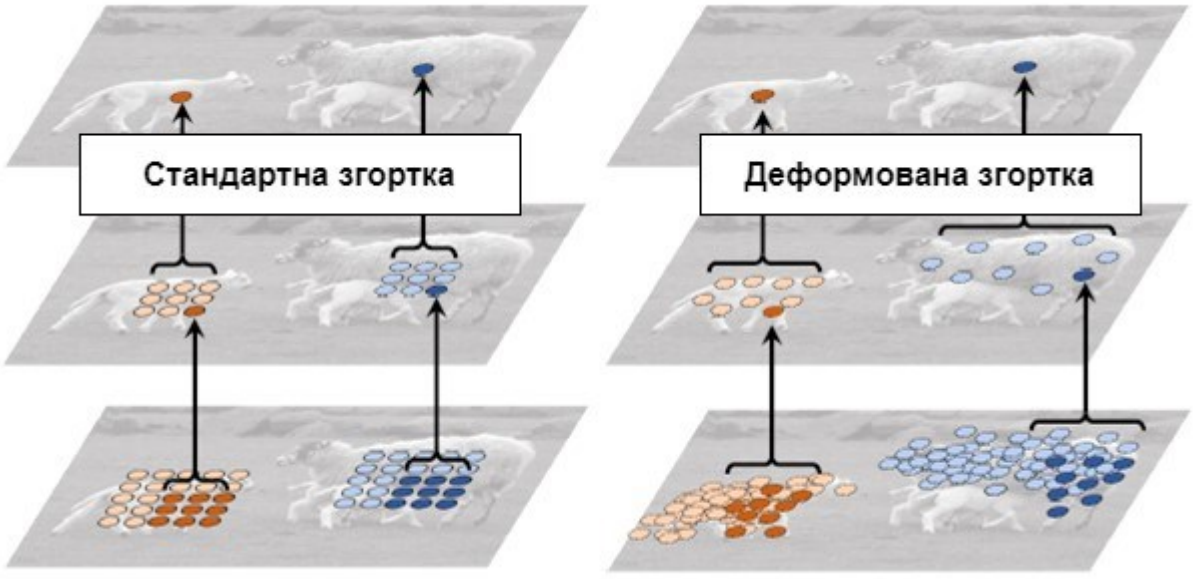

Рисунок 2.3 - Схематичне порівняння роботи звичайної згортки та деформованої

Звичайна згортка здійснюється на попередньо визначеній прямокутній сітці з вхідного зображення або набору вхідних карт ознак на основі визначеного розміру фільтра. Ця сітка може бути розміром 3 × 3 і 5 × 5 тощо. Однак об’єкти, які мають бути виявлені та класифіковані, можуть бути деформовані або закриті в межах зображення.

У DC ця сітка деформується таким чином, що кожна точка сітки переміщується на зсув, значення якого визначається на етапі навчання. І згортка обчислюється на основі цих переміщених точок сітки.

Результати обчислення контекстного блоку подаються на вхід до функцій втрат. Модель має три види функції втрат: одна для бінарної класифікації (для відповіді на питання, чи містить обличчя конкретний регіон зображення); інша для регресії обмежувального прямокутника, щоб визначити координати обличчя; та третя для локалізації орієнтирів для ідентифікації ознак обличчя. Під час навчання ці три функції втрат оптимізуються одночасно.

Архітектура використовує якорні прямокутники (anchors) на різних масштабах та пропорціях, щоб підвищити шанси виявлення облич в різних орієнтаціях. Якорні прямокутники (якорні блоки) — це попередньо визначені прямокутники різних масштабів і співвідношення сторін, додані до кожного пікселя в картах ознак, що слугують у якості пропозицій для обмежувальних прямокутників (рисунок 2.4). Для кожного якорного блоку модель передбачає ймовірність присутності об’єкта і виконує регресію обмежувального прямокутника, щоб уточнити положення та розміри прив’язки, якщо об’єкт дійсно присутній. Такий двозадачний підхід прогнозування забезпечує здатність моделі обробляти об’єкти різних розмірів і форм.

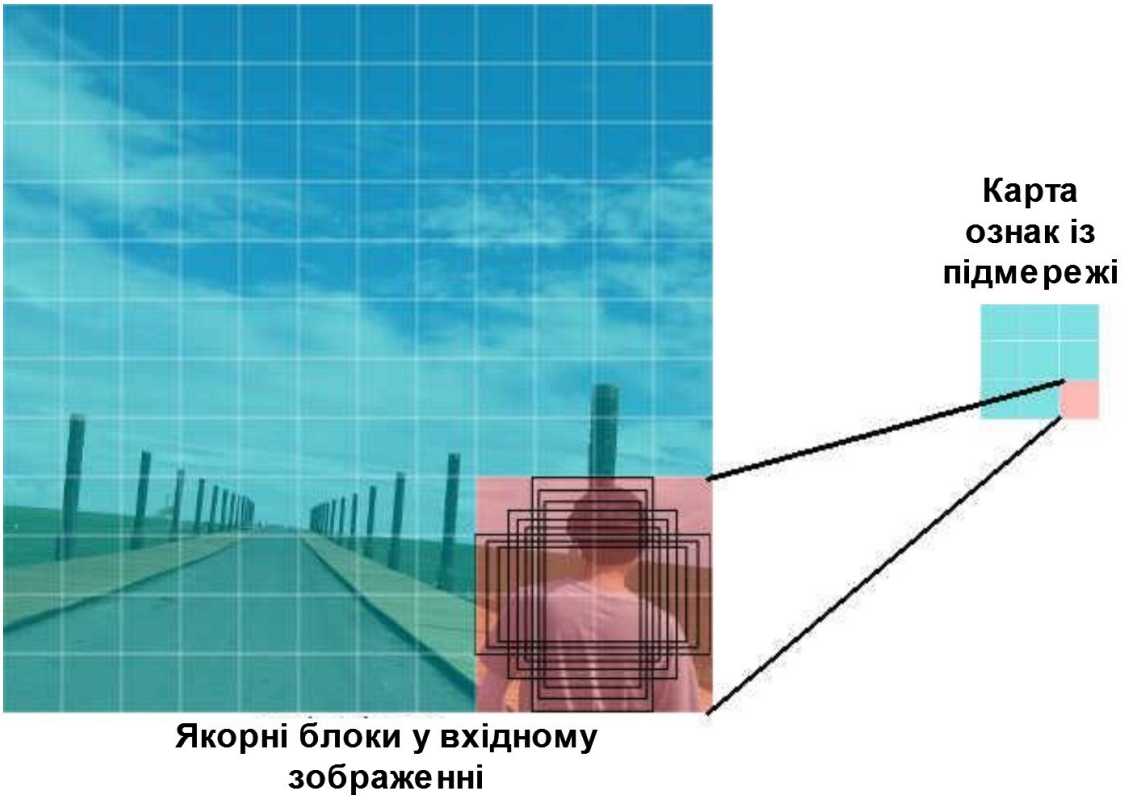

Рисунок 2.4 – Якорні блоки з потенційним регіоном виявлення

Водночас модель оцінює орієнтири обличчя для кожного з цих якорних прямокутників. Потім використовується метод немаксимального придушення (Non-maximal supression, NMS) для видалення зайвих та прогнозів з низькою впевненістю.

NMS – це метод комп’ютерного зору, який вибирає один прямокутник з об’єктом із багатьох інших аналогічних прямокутників, що перекриваються (наприклад, обмежувальні прямокутники під час виявлення об’єктів). Він працює шляхом порівняння значень вірогідностей для запропонованих обмежувальних прямокутників. На цьому етапі видаляються ті прямокутники, які значно збігаються з обмежувальним прямокутником з вищим значенням вірогідності.

Значення вірогідності отримується в результаті бінарної класифікації (рисунок 2.5)

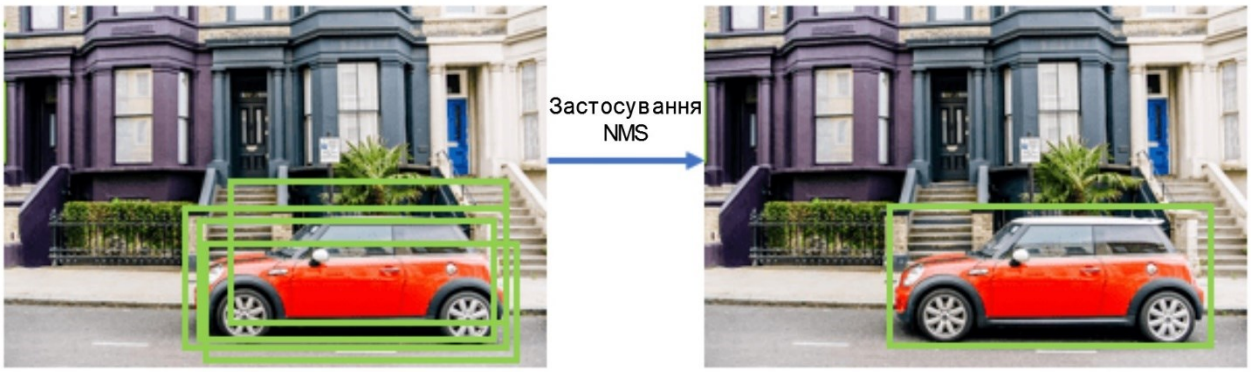

Рисунок 2.5 - Візуальне представлення результату роботи алгоритму NMS

Нехай *B* - множина усіх знайдених граничних областей для одного зображення, де кожна гранична область *b* представлена у вигляді *b=(x,y,w,h,s)*. Тут:

- *(x,y)* - координати лівого верхнього кута прямокутнику;

- *(w,h)* - ширина і висота прямокутника відповідно;

- *s* позначає довірчу оцінку, пов'язану з прямокутником.

Алгоритм NSM включає в себе наступні кроки:

1. Ініціалізація: Відсортувати множину *B* за спаданням на основі довірчих оцінок s , в результаті чого отримати *B′*.

2. Вибір: Вибрати обмежувальну область b_m з найвищою довірчою ймовірністю з множини *B′* і додати її до остаточної множини виявлення *D*.

3. Просіювання: Для кожної другої обмеженої множини b_n у *B′* обчислити перетин об'єднання (IoU) з b_m за формулою 2.1:

$$IoU(b_m, b_n) = \frac{Area(b_m \cap b_n)}{Area(b_m \cup b_n)} \quad (2.1)$$

Якщо $IoU(b_m, b_n)$ перевищує заданий поріг θ, b_n видаляється з B'.

4. Ітерація: Повтор кроків 2 і 3, поки B' не стане порожнім.

RetinaFace була перевірена на ряді нормативних наборів даних, таких як WIDER FACE і FDDB, і показала одні з кращих показників за різними метриками. Вона особливо стійка до роботи в широкому діапазоні складних умов, включаючи низьку роздільну здатність, екстремальні положення та перешкоди перед обличчям [85].

2.3 Набір даних WIDER FACE

У даній роботі для експериментів було використано набір даних WIDER FACE.

Набір даних WIDER FACE є еталонним набором для комплексної оцінки алгоритмів виявлення облич. Цей набір даних з'явився у відповідь на помітний розрив між існуючими показниками якості розпізнавання облич і вимогами реального світу.

Однією з відмінних рис набору даних WIDER FACE є його обсяг; він у 10 разів більший за свої попередні аналоги. Цей набір даних охоплює загалом 32 203 зображення, на яких ретельно анотовано 393 703 обличчя. Ці анотації не обмежуються лише рамками навколо облич, але також поширюються на такі деталі, як оклюзії (перекриття облич), різноманітні пози, певні категорії подій тощо. Зображення, отримані для цього набору даних, зібрані з різних категорій подій, що забезпечує широкий спектр сценаріїв і передумов. Обличчя в цьому наборі даних демонструють велику варіативність, особливо щодо масштабу, пози та оклюзії (рисунок 2.6)

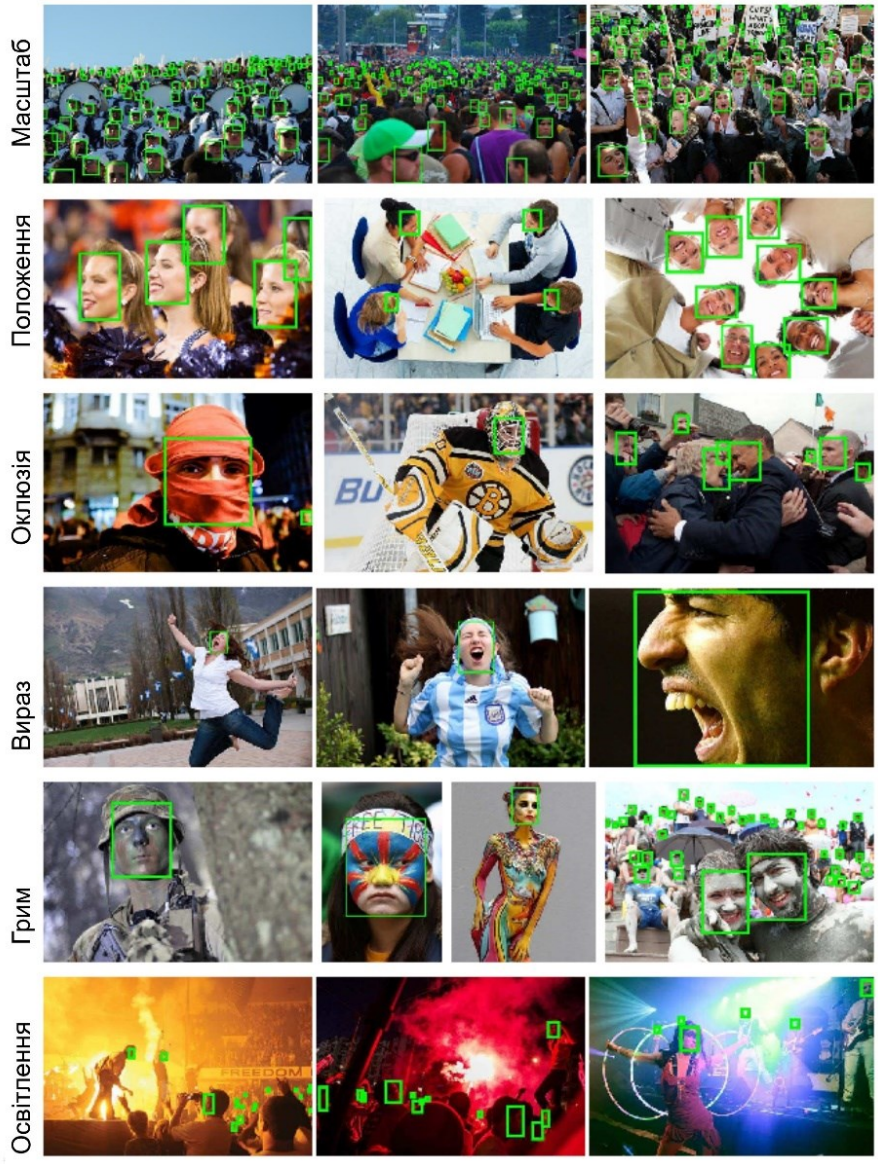

Рисунок 2.6 - Приклади зображень з набору даних WIDER FACE для виявлення обличчя, який має високий ступінь варіабельності масштабу, пози, оклюзії, виразу, зовнішнього вигляду та освітлення.

Набір даних WIDER FACE використовується не лише як еталон для оцінки якості, але й набір даних для навчання алгоритмів розпізнавання облич, і є у вільному доступі.

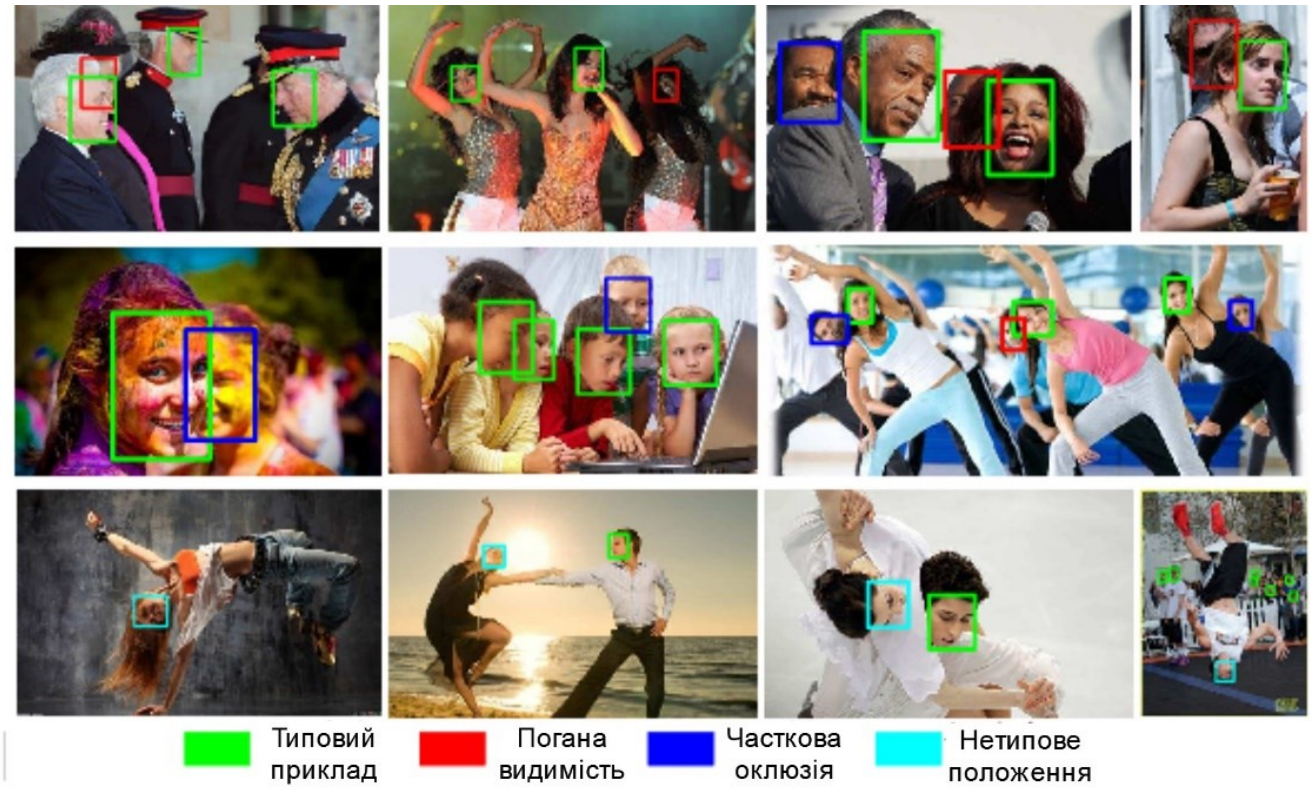

Рисунок 2.7 - Приклади анотацій WIDER FACE для різних особливостей положення обличчя на зображеннях

Структуру анотацій наведено в таблиці 2.1.

Таблиця 2.1

Структура міток набору WIDERFace

Мітка	Опис	Можливі значення
File name	Назва файлу зображення	Рядок (наприклад, 'img_001.jpg')
Number of faces	Кількість облич на зображенні	Ціле число (наприклад, 3)
x-coordinate	X-координата лівого верхнього кута	Ціле число (наприклад, 100)

Продовження таблиці 2.1

Мітка	Опис	Можливі значення
y-coordinate	Y-координата лівого верхнього кута	Ціле число (наприклад, 150)
Width	Ширина обмежувальної рамки	Ціле число (наприклад, 85)
Height	Висота обмежувальної рамки	Ціле число (наприклад, 100)
Blur	Рівень розмиття	0: Чітке 1: Незначне розмиття 2: Сильне розмиття
Expression	Вираз обличчя	0: Нейтральний 1: Виразний
Illumination	Стан освітлення	0: Нормальний 1: Екстремальний
Invalid	Достовірність виявлення обличчя	0: Дійсне обличчя 1: Недійсне обличчя
Occlusion	Ступінь оклюзії обличчя	0: Немає оклюзії 1: Часткова оклюзія 2: Сильна оклюзія
Pose	Поза обличчя	0: Типова поза 1: Нетипова поза

На основі точності виявлення облич алгоритмом EdgeBox визначено три рівні складності піднаборів даних: «Легкий», «Середній» і «Складний» із середніми показниками виявлення алгоритмом 92%, 76% і 34% відповідно [86].

2.3 Аналіз методу прунінгу SNIP (Single-shot Network Pruning)

Метод прунінгу SNIP полягає в знаходженні розрідженої підмножини ваг W_s, де $W_s \subseteq W$, такої, щоб значення функції втрат $L(W_s)$ було мінімальним, де W представляє ваги усієї нейронної мережі, а $L(W)$ - це функція втрат, яку мережа намагається мінімізувати під час навчання. Під час застосування алгоритму нейронна мережа спочатку ініціалізується набором випадкових ваг, позначених як W. Також, для імплементації алгоритму застосовується набір масок для векторів вагів *c,* що мають значення {0,1}.

Під час тренування нейронної мережі виконується обчислення функції втрат на тренувальних даних для обчислення градієнтів ∇L функції втрат відносно кожної маски c_i. Отримані градієнти містять значення що показують вплив нескінченно малої зміни масок c на функцію втрат L, тим само індикуючи важливість кожної маскованої ваги.

Важливість s_i кожної ваги w_i в W обчислюється за формулою 2.2:

$$s_i = \left| \frac{\partial L(c_i)}{\partial c_i} \right| \tag{2.2}$$

Де c_i - це маскуючі матриці (маски) зі значеннями 0 або 1 для вектору вагів w_i.

Обчислені оцінки важливості s_i нормалізуються для отримання $\hat{s}_i$, використовуючи рівняння 2.3:

$$\hat{s}_i = \frac{s_i}{\sum_j s_j} \tag{2.3}$$

Нормалізовані оцінки важливості $\hat{s}_i$ сортуються за зростанням та доля ρ ваг, які мають найнижчі нормалізовані оцінки важливості, обрізається з множини W, що створює нову підмножину:

$$\mathcal{W}_s = \{w_i \in \mathcal{W} : \hat{s}_i \geq \rho\} \tag{2.4}$$

Отримавши обрізану мережу, яку тепер представляє W_s, відбувається навчається за допомогою стандартних алгоритмів оптимізації.

Вимогою для ефективної роботи цього алгоритму є використання однакового значення дисперсії для кожного шару нейронної мережі. Цей метод також використовується на нових шарах, які не були включені в попередньо навчену основну мережу. З метою досягнення однакового значення дисперсії використовується ініціалізація Ксав'є (формула 2.5) [87]:

$$w_{ij} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right] \tag{2.5}$$

де U - рівномірний розподіл на, а n_j - розмір j-того шару.

Варіювання кількості обрізних ваг відбувається за допомогою введеного параметру *ступеня стиснення (p),* що обчислюється за формулою 2.6, де p - параметр швидкості обрізання, N_{pruned} - кількість обрізаних вузлів або з'єднань, і N_{total} - загальна кількість вершин або з'єднань до обрізання:

$$p = \frac{N_{\text{pruned}}}{N_{\text{total}}} \tag{2.6}$$

Значення p встановлюється в діапазоні від 0 до 1, де 0 означає, що обрізання не виконується, а 1 - всі можливі вершини або зв'язки обрізаються.

2.4 Вдосконалення методу прунінгу для зменшення кількості параметрів в мережі для виявлення облич RetinaFace

Архітектура RetinaFace використовує піраміди ознак, розраховані з виходів згорткових блоків ResNet-50 (таблиця 2.2).

Таблиця 2.2

Згорткові блоки ResNet-50

Назва блоку	Фільтри (розмір, кількість)	Повтори фільтрів
C2	1×1, 64 3×3, 64 1×1, 256	3
C3	1×1, 128 3×3, 128 1×1, 512	4
C4	1×1, 256 3×3, 256 1×1, 1024	36

Виходи блоків згортки використовуються для розрахунку пірамід ознак з використанням вертикальних і бічних зв'язків (рисунок 2.8)

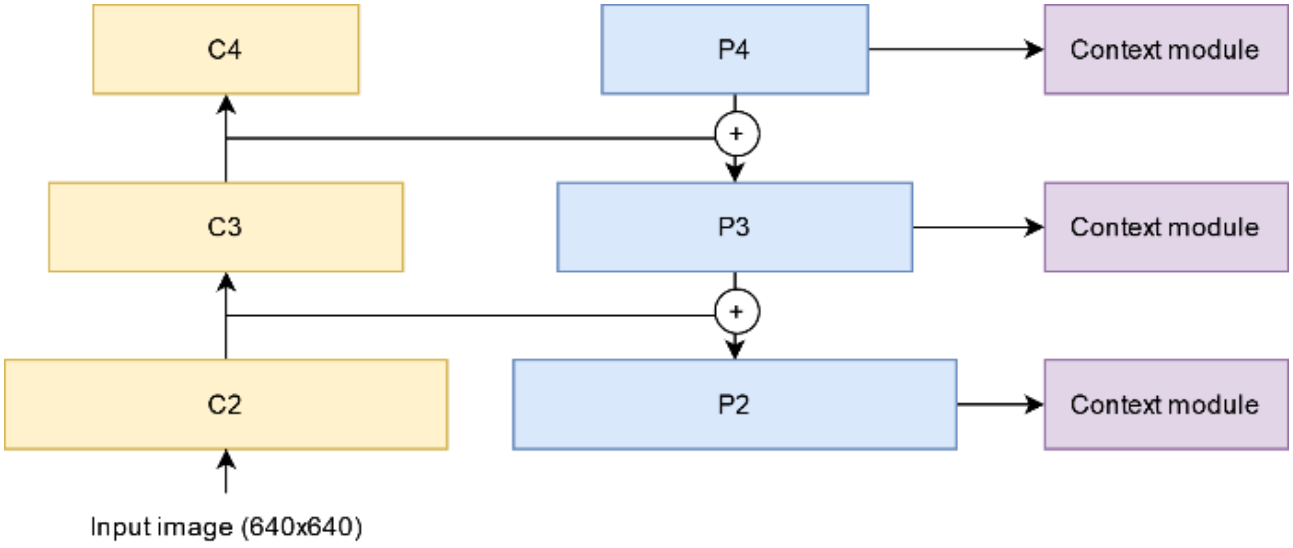

Рисунок 2.8 - Модель RetinaFace з використанням основи ResNet-50

Контекстний модуль RetinaFace використовує згорткові шари з більшим розміром фільтра для збільшення розміру вікна (рисунок 2.9).

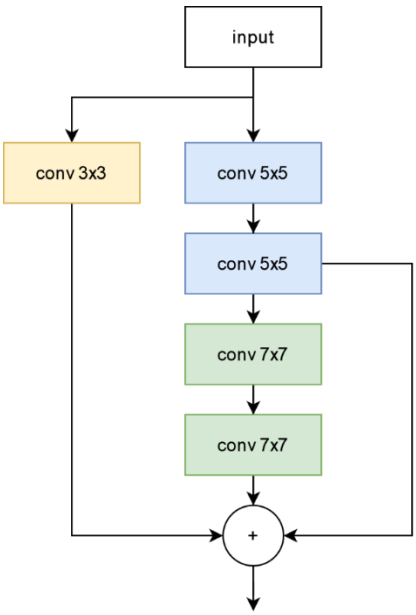

Рисунок 2.9 - Архітектура контекстного блоку.

Далі модель за допомогою контекстних модулів обчислює 3 виходи, що дозволяє вирішити задачу класифікації обличчя та задачу регресії локації обличчя.

- Задача класифікації обличчя полягає у виявленні обличчя на певному якорі. Як функція втрат (L_{cls}) використовується бінарна перехресна ентропія;
- Задача регресії локації обличчя має на меті знаходження координати розташування обличчя. В якості функції втрат використовується згладжена L1 (L_{box}).

Загальна функція втрат обчислюється за рівнянням:

$$L = L_{cls} + \lambda_1 L_{box} + \lambda_2 L_{pts} \quad (2.7)$$

На основі проведених експериментів було доведено ефективність алгоритму прунінгу SNIP на задачі розпізнавання облич, використовуючи однократний прунінг перед навчанням моделі RetinaFace. Навчання проводилося на наборі даних WIDERFace. Для розрахунку градієнтного спуску використовувався SGD з параметром моменту 0,9, розміром батчу 2 і швидкістю зменшення ваги 0,0005. Початкова швидкість навчання була встановлена на 0.001

і зменшувалася 2 рази на 70 і 90 епохах. Загальна кількість епох дорівнювала 100 епохам. Значення функції втрат під час навчання відслідковувалося за допомогою інструменту Tensorboard і показано на Рисунку 2.10.

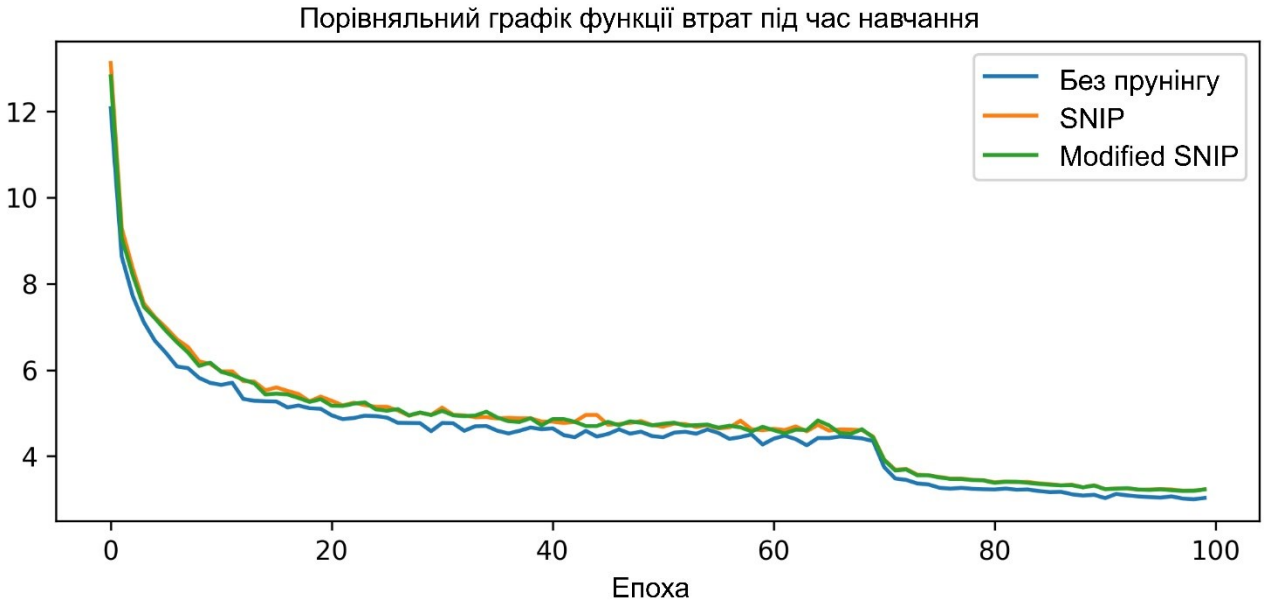

Рисунок 2.10 – Криві функції значення функції втрат під час навчання

Результати точності, отримані після навчання обрізаних та необрізаних моделей, наведено в таблиці 2.3.

Таблиця 2.3

Результати навчання

Модель	WIDERFace (Easy)	WIDERFace (Medium)	WIDERFace (Hard)	Кількість параметрів
Без прунінгу	93.5%	91.6%	75.9%	3756032
Прунінг (SNIP)	93.5%	91.0%	74.5%	1187762
Прунінг (Modified SNIP)	**93.5%**	**91.0%**	**75.2%**	1187762

В результаті навчання нейронної мережі після застосування вдосконаленого методу був відбувся перерозподіл обрізаних ваг по нейронній

мережі, де більшість ваг було обрізано в перших шарах мережі [88]. Результати розподілу обрізаних ваг показано на рисунку 2.11.

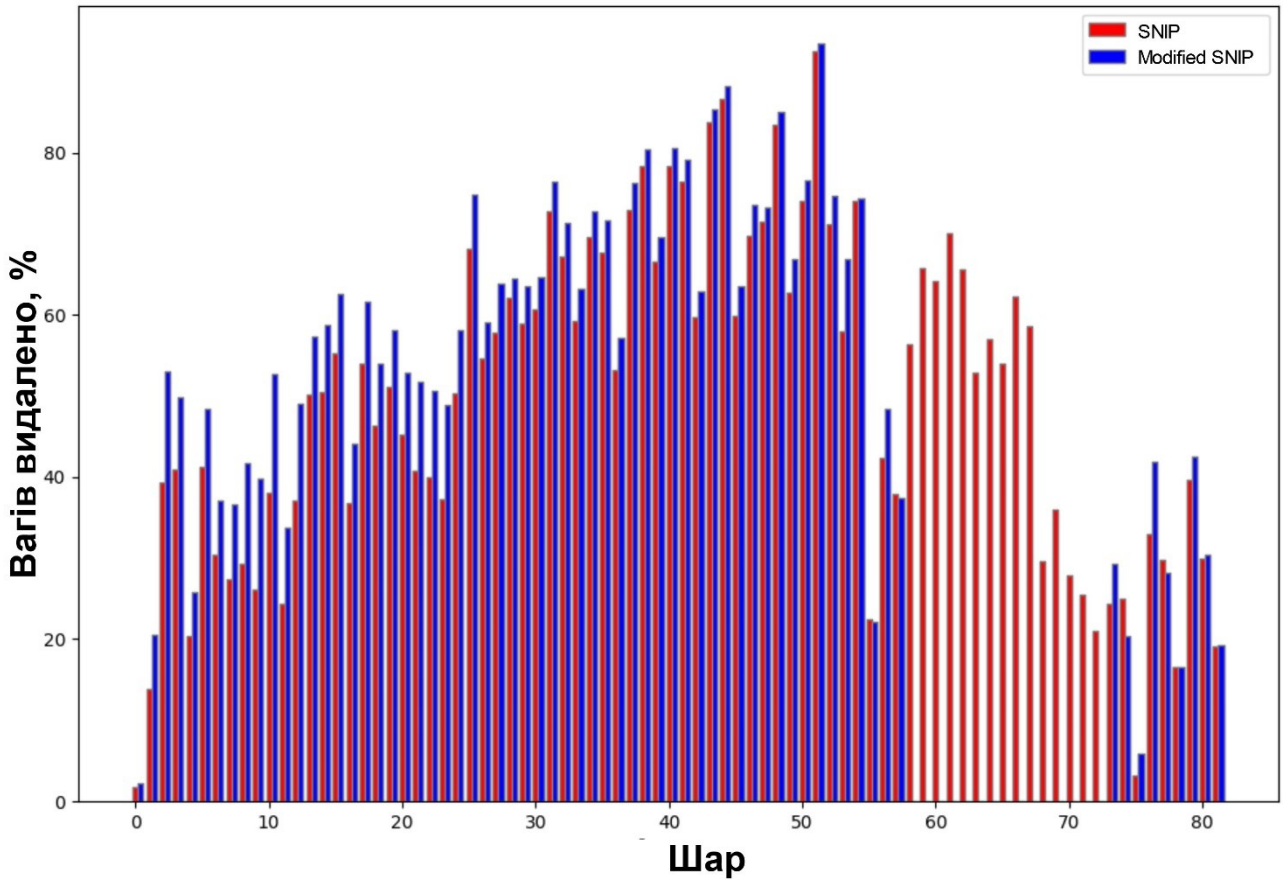

Рисунок 2.11 - Розподіл обрізаних ваг за шарами.

В результаті навчання було отримано мережу з розрідженими вагами, що становлять на 67% вагів менше, ніж початкова мережа, при цьому при використанні алгоритму SNIP точність впала на 1,4%, а при використанні вдосконаленого алгоритму з виключенням контекстних модулів з процедури прунінгу – лише на 0.7%.

2.5 Аналіз впливу додаткових виходів на точність моделей сіамських близнюків

Сіамська нейронна мережа (SNN) зазвичай використовується для кількісної оцінки подібності між двома об’єктами. Термін "сіамська" походить від концепції зрощених близнюків, які фізично з’єднані один з одним. Аналогічно, сіамська нейронна мережа використовує дві або більше ідентичних підмереж, що мають однакові характеристики. Підмережі можуть бути імплементовані використовуючи багатошаровий персептрон (MLP), згорткові нейронні мережі (CNN) або інші архітектури. SNN широко застосовуються в ситуаціях, що вимагають встановлення критерію подібності між або більше сутностями одного типу. Незважаючи на високі обчислювальні витрати, вони працюють краще порівняно з іншими методами визначення подібності, головним чином завдяки здатності до узагальнення на подібних наборах даних [89]. Тому можна використовувати попередньо навчені підмережі, щоб знайти компроміс між обчислювальним часом і точністю [90]. Навчання SNN не вимагає ручного створення ознак, оскільки ознаки можна визначити з завчасно натренованих підмереж.

Архітектура сіамських нейронних мереж складається з частини з кодуванням вхідних даних та алгоритмом їх порівняння. При кодуванні модель приймає на вхід данні, нейронна мережа виділяє ключові ознаки та робить їх представлення у багатовимірному просторі. У випадку, що розглядатиметься, вхідними даними є зображення.

Після кодування моделлю зображення у багатовимірний можливо зробити розрахунок відстані між різними закодованими зображеннями. Самим розповсюдженим способом розрахунку відстані між закодованими даними сіамською нейронною мережею є евклідова відстань, яка розраховується за формулою 2.8:

$$D_W\left(\vec{X}_1, \vec{X}_2\right) = \| G_W(\vec{X}_1) - G_W(\vec{X}_2) \|_2 \quad (2.8)$$

де X_1 та X_2– вхідні зображення, G_W – функція перетворення, в нашому випадку це нейронна мережа, D_W – відстань між зображеннями.

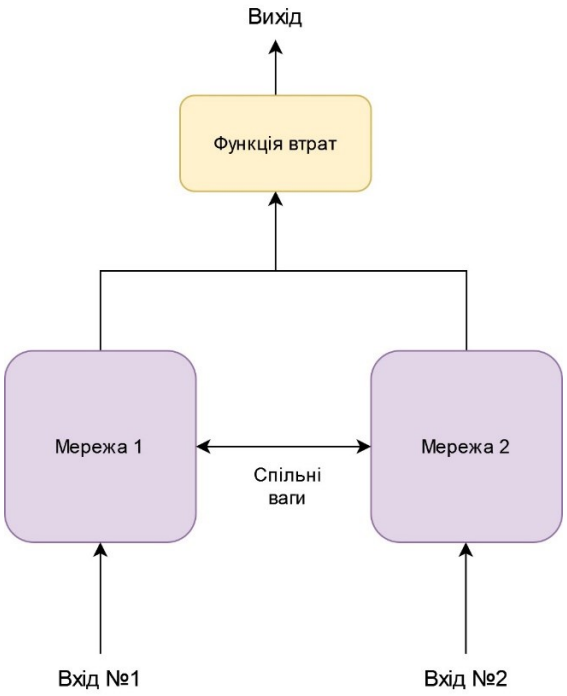

Рисунок 2.12 – Приклад сіамської нейронної мережі

Зображення проходять однакові перетворення, а гілки сіамської нейронної мережі мають спільні ваги (рисунок 2.12). Така архітектура надає можливість залишити лише одну гілку моделі яка відповідає за кодування зображення, а також дозволяє в майбутньому швидко переводити зображення в зручний багатовимірний простір та зберігати результати для подальшого порівняння не застосовуючи повторні виклики операцій моделі для отримання результатів кодування.

При цьому такий формат результатів потребує спеціальної функції визначення помилки – contrastive loss. Contrastive loss рахує помилку в отриманій відстані відносно очікуваної (рисунок 2.13). Помилка розраховується за формулою 2.9

$$L\left(W, Y, \vec{X}_1, \vec{X}_2\right) = (1-Y)\frac{1}{2}\left(D_W\right)^2 + (Y)\frac{1}{2}\{max(0,\ m - D_W)\}^2 \quad I \tag{2.9}$$

де W – параметри системи, Y- очікувана відстань між зображеннями, m – очікувана відстань між різними зображеннями.

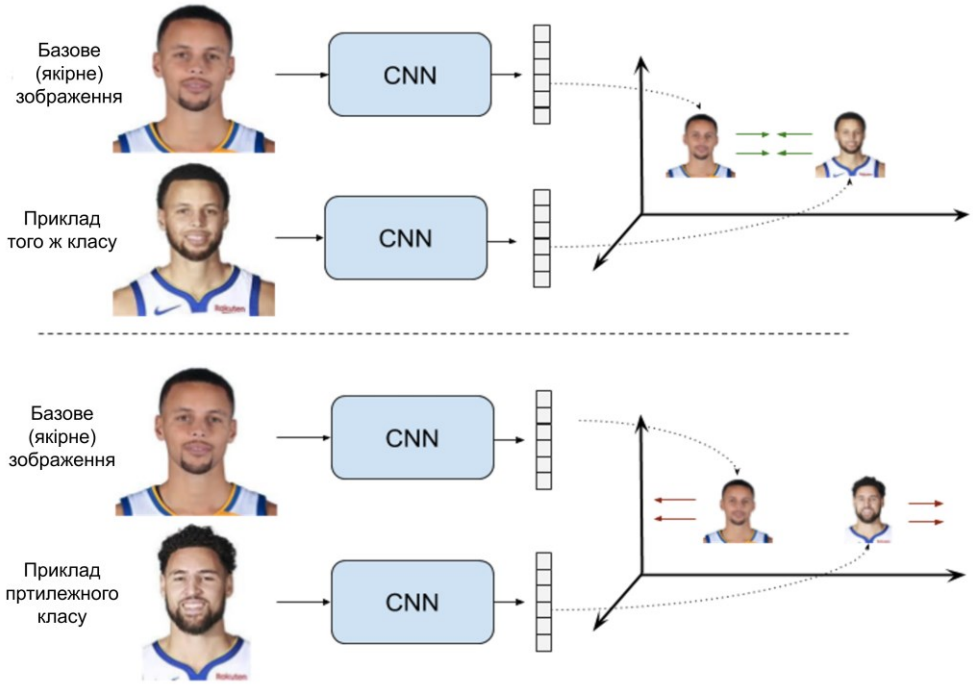

Рисунок 2.13 – Приклад роботи contrastive loss

Покращеним підходом є сіамська нейронна мережа з трьома гілками під час навчання. Особливістю такого підходу є те, що до неї передається базове зображення, відповідне до нього та протилежне до нього (рисунок 2.14). Це необхідно для того, щоб модель відразу групувала однакові зображення в групи, при цьому збільшуючи відстань до протилежних груп. Для обрахунку функції помилки використовується triplet loss, що обчислюється за формулою 2.10:

$$Loss = \sum_{i}^{N}\left[\| f\left(x_i^a\right) - f(x_i^p)\|_2^2 - \| f\left(x_i^a\right) - f(x_i^n)\|_2^2 + \alpha\right] \tag{2.10}$$

де x^a це базове зображення, x^p - зображення того ж типу, що і базове, x^n– протилежне до нього зображення, f – функція перетворення зображення в маловимірне представлення, – мінімальна відстань між різними зображеннями.

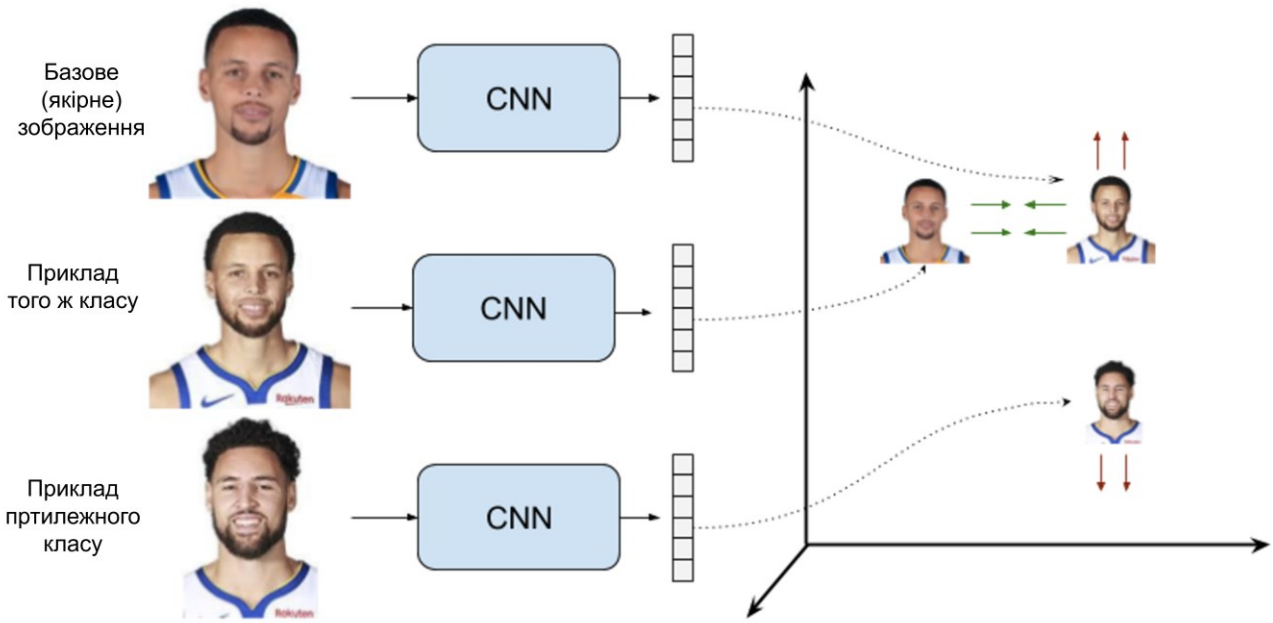

Рисунок 2.14 – Приклад роботи triplet loss

Така архітектура моделі дозволяє групувати зображення навіть маючи невелику кількість даних, що є дуже важливим при вирішенні задач специфічних доменних областей. Проблема полягає в падінні точності через недостатню кількість даних для навчання та недостатню силу узагальнення знань про доменну область, але якщо даних недостатньо то модель може все одно не навчитись виділяти ключові візуальні і концептуальні ознаки, які мають впливати на прийняття рішення нейронною мережею.

Для оптимізації навчання сіамських нейронних мереж, а саме підвищення точності з мінімальною втратою швидкодії, було вирішено використати додаткові виходи, які будуть виконувати задачі класифікації чи пошуку специфічних атрибутів. Цей підхід вперше було використано при розробці моделі GoogLeNet [26], яка запропонувала використання додаткових виходів для класифікації. Проблема яку досліджували розробляючи GoogLeNet полягала в поганому розповсюдженні помилки в великих моделях. Для усунення цієї

проблеми було вирішено додати додаткові виходи класифікації уздовж основної архітектури, щоб градієнт розповсюджувався рівномірно по всій моделі. Цей підхід дозволив значно знизити вплив затухаючого градієнту пришвидшивши збігання моделі та її точність.

Для створення таких виходів необхідно використовувати спеціальні модифікації даних, які будуть відображати більш узагальнені характеристики. Як результат, додаткові виходи будуть виконувати також задачу порівняння більш узагальнених особливостей чи задачу класифікації (рисунок 2.15). Такий підхід вимагає значно більших витрат часу для дослідження даних, виділення особливостей та програмування більш складних наборів даних з метою паралельного навчання моделі для вирішення декількох задач відразу.

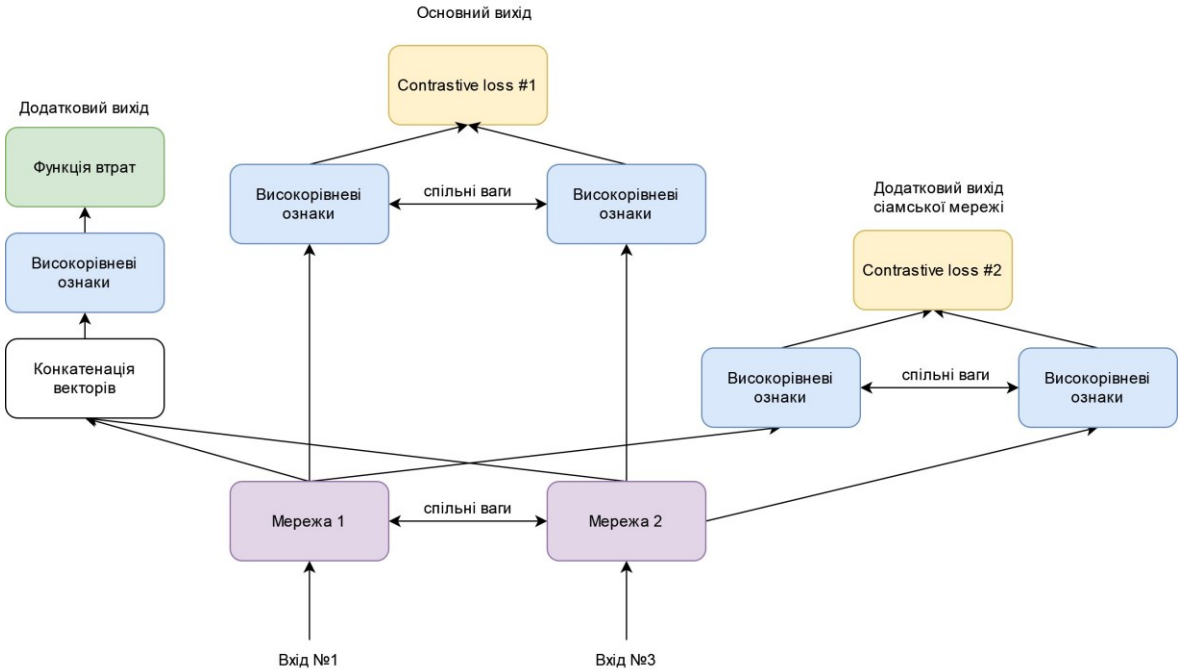

Рисунок 2.15 – Приклад архітектури з додатковими виходами для визначення додаткових особливостей

В результаті такого навчання модель виділяє ознаки на більш низькому рівні, необхідні для вирішення другорядних задач, що має покращити точність та узагальнюючи здібності моделі.

При цьому, розподіл вагів для кожного виходу моделі впливає на якість навчання моделі в цілому, тому важливий механізм для регулювання впливу цих гілок на навчання. Такий механізм має запобігти надлишковому впливу другорядних задач на основну, але дозволить другорядним задачам виступати в якості додаткового інструменту для узагальнення знань нейронної мережі про доменну область.

Для експериментальної перевірки було створено дві сіамські нейронні мережі. В першому випадку модель мала класичну архітектуру з розрахунком відстані між закодованими зображеннями. Інша варіація включала в себе додатковий вихід з класифікатором на кінці. Для обох архітектур застосовувались однакові шари, функція розрахунку помилки – contrastive loss, та алгоритм розрахунку градієнтів RMS Prop.

Для першого експерименту було вирішено взяти класичний набір даних Fashion Mnist та провести експерименти з базовою архітектурою та модифікованою за допомогою додаткових виходів. Даний набір даних складається з рівномірно розподілених даних та має невеликий розмір зображень, що дозволяє провести експерименти в умовах обмежених обчислювальних ресурсів.

Шляхом експериментів було вирішено навчати модель протягом 100 епох, так як цього було достатньо, щоб модель могла досягти свого найкращого значення. Перші результати порівняння показали, що сіамська нейронна мережа без додаткових виходів краще справляться з задачею, а саме швидше сходиться та має кращу точність результатів що дорівнювала 92.3%. Тому наступним кроком було вирішено додати ваги до функцій розрахунку похибки моделі. Було зроблено додатково три експерименти в яких вплив додаткових виходів знижувався шляхом множення градієнтів на коефіцієнти 0.5, 0.2 та 0.1. Експерименти показали, що чим менше вага виходів, тим вище результуюча точність (Таблиця 2.4).

Таблиця 2.4

Порівняння результатів навчання моделі для різних коефіцієнтів градієнту

Виходи	Точність	Різниця порівняно з оригіналом
Classification, loss weights 100%	90,08%	-2,22%
Classification, loss weights 50%	90,93%	-1,37%
Classification, loss weights 20%	91,75%	-0,55%
Classification, loss weights 10%	92,07%	-0,23%

Для наступного експерименту було взято частину набору даних PlantVillage. Розмір набору складає близько 20 тисяч зображень та містить 15 класів для розпізнавання. З метою виділення додаткових особливостей було вирішено розбити цей набір на умовні групи. Перша група це тип рослини, до якої належить клас. Таким чином було отримано 3 класи: “Pepper”, “Potato” та “Tomato”. Іншою групою став стан листів – здорові чи хворі. Так як в наборі описані різні види захворювань та різні стадії цих захворювань було вирішено розбити на 2 класи. Таким чином була отримана можливість створити додаткові виходи для класифікації типу рослини та регресії яка показувала стан здоров’я рослини.

Була розроблена архітектура сіамської нейронної мережі, на основі попередньо навченої моделі ResNet50V2. Модель ResNet50V2 була обрана через те, що вона навчена на великому наборі даних ImageNet, який включає в себе 1,281,167 розмічених зображень різних класів, що дозволяє використовувати цю модель з попередньо навченими вагами, які мають узагальнюючі особливості та здатні виділяти широкий спектр візуальних особливостей зображення. Іншою перевагою цієї моделі є її швидкість та низькі вимоги до ресурсів системи.

Для аналізу особливостей отриманих з ResNet50V2 та побудови вектору ознак для сіамської нейронної мережі були створені додаткові шари. Далі ці вектори передаються в функцію розрахунку Евклідової відстані. В ролі функції

розрахунку помилки використовувалась Contrastive loss. В результаті навчання такої моделі вдалось досягти точності 96.54%.

Для дослідження впливу додаткових виходів було додано вихід з регресією, метою якої було звернути увагу моделі на стан листів. Також була створена варіація моделі, яка мала як додатковий вихід класифікатор. Метою цього виходу було зробити акцент моделі на типу листів. В якості третього дослідного екземпляру було використано модель з обома додатковими виходами – регресія і класифікатор.

Під час навчання цих моделей застосовувались різні варіанти розподілення вагів функції помилки. Було проведено ряд експериментів, де вага впливу додаткових виходів впливали на результати навчання на 100%, 50%, 20% та 10% відносно вагів функції помилки для виходу з Contrastive loss (Таблиця 2.5).

Навчена модель сіамської нейронної мережі досягає точності 96.54%.

Таблиця 2.5

Порівняння результатів навчання моделі з використанням різних додаткових виходів.

Виходи	Точність	Різниця з оригіналом
Класифікація і регресія, вплив вагів 100%	95,19%	-1,35%
Класифікація і регресія, вплив вагів 50%	97,26%	0,72%
Класифікація і регресія, вплив вагів 20%	96,20%	-0,34%
Класифікація і регресія, вплив вагів 10%	96,90%	0,36%
Класифікація, вплив вагів 100%	91,00%	-5,54%
Класифікація, вплив вагів 50%	94,35%	-2,19%
Класифікація, вплив вагів 20%	95,43%	-1,11%
Класифікація, вплив вагів 10%	96,24%	-0,30%
Регресія, вплив вагів 100%	93,92%	-2,62%

Продовження таблиці 2.5

Виходи	Точність	Різниця з оригіналом
Регресія, вплив вагів 50%	94,62%	-1,92%
Регресія, вплив вагів 20%	96,61%	0,07%
Регресія, вплив вагів 10%	96,60%	0,06%

В результаті навчання було встановлено, що на точність моделі додаткові гілки майже не впливають. В середньому точність падала на 1.11%. При цьому найкращий результат показала модель з двома додатковими виходами та впливом їх ваги на функцію помилки в 50%. В моделях для 20% та 10% впливу з обома додактовими виходами, найкраща точність відрізнялась від точності моделі лише з одним виходом на -0.34% та +0.36% відповідно. Така поведінка свідчить, що додаткові виходи мають досить обмежений вплив на результуючу точність моделі [91].

Висновки до розділу 2

Було проаналізовано один з методів оптимізації нейронних мереж на сучасних архітектурах, а саме метод прунінгу перед навчанням SNIP, що дозволяє оцінити важливість кожної ваги в нейронній мережі за допомогою розрахунку критерію важливості на основі вкладку в функцію похибки. В якості архітектури нейронної мережі було обрано архітектуру RetinaFace, що базується на пірамідах ознак (FPN), для вирішення задачі виявлення облич. В якості набору даних було обрано набір даних WIDERFace, що використовується як тренувальний і тестовий набір даних, а також має розбиття на вкладені набори з різним рівнем складності виявлення облич.

Результати досліджень показали ефективність алгоритму прунінгу перед навчанням SNIP, в результаті тренування було отримано мережу, що налічує на

68% параметрів менше, при цьому має точність лише на 1,4% меншу при оцінці на складній частині набору даних, ніж мережа без прунінгу, що доводить ефективність методу.

Крім цього, була розроблена модифікація методу, при якій оцінка критерію розраховувалась на всіх модулях, крім контекстних модулів мережі, що дозволило скоротити розрив до 0,7% порівняно з мережею без прунінгу, при цьому кількість видалених параметрів залишилась незмінно. Такі результати свідчать про можливість покращення методу SNIP для сучасних мереж і важливість врахування архітектурних особливостей при прунінгу нейронних мереж.

Було проаналізовано альтернативу методам прунінгу в оптимізації – внесення змін в архітектуру нейронних мереж задля підвищення точності і прискорення збіжності при тренуванні. В якості мережі було обрано сіамські нейронні мережі, що використовувались для вирішення задачі оцінки схожості двох зображень. Для оптимізації було вирішено обрати підхід додавання додаткових задач для вирішення нейронною мережею, оскільки попередні дані свідчать про кращу здатність мереж узагальнювати ознаки при використанні такого підходу. Результати показали, що при тренуванні сіамських нейронних мереж даний метод оптимізації виявився малоефективним.

3. РОЗРОБКА МЕТОДУ ПРУНІНГУ ДЛЯ ЗБІЛЬШЕННЯ ШВИДКОДІЇ МЕРЕЖ АРХІТЕКТУРИ ТРАНСФОРМЕР

Моделі глибокого навчання архітектури трансформер (трансформери) змогли покращити якість вирішення задач в різних областях штучного інтелекту. Найбільшого поширення моделі трансформер набули в задачах машинного перекладу, реферуванні тексту та системах відповідей на запитання [92, 93].

До появи трансформерів, рекурентні нейронні мереж (RNN) та їх вдосконаленого варіанту, мереж довгої короткострокової пам’яті (LSTM), були основними моделями для завдань NLP. Ці моделі обробляють дані послідовно, підтримуючи приховані стани на ітераціях для врахування тимчасових зв’язків. Однак вони мали обмеження, особливо з точки зору ефективності навчання та врахування попередніх зв’язків.

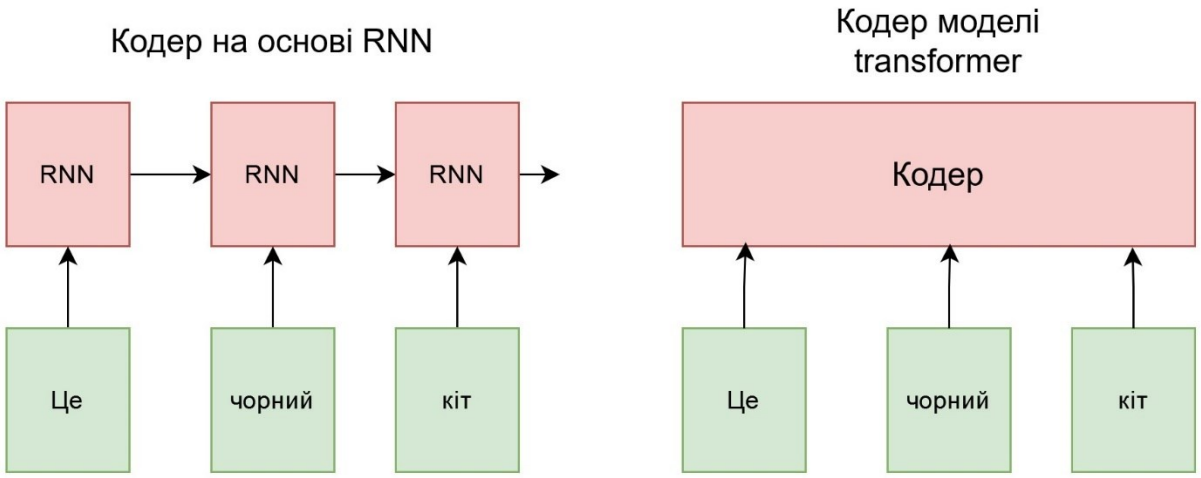

Рисунок 3.1 - Порівняння блоку кодування RNN та трансформерів

Моделі архітектури трансформер оброблюють токени паралельно, усувають необхідність послідовної обробки даних та запроваджують механізми

самоуваги (рисунок 3.1). Цей підхід дозволяє моделям при аналізі елементів в послідовності встановлювати зв’язок з іншими елементами в тій же послідовності, незалежно від їх відстані, дозволяючи моделі більш ефективно фіксувати складні відносини.

3.1 Аналіз особливостей архітектури трансформер

Архітектура трансформер представила новий підхід до навчання нейронних мереж – механізм уваги. Механізм уваги обчислює зважене представлення вхідних даних, дозволяючи кожному токену «відвідувати» кожен інший токен у послідовності вхідних даних.

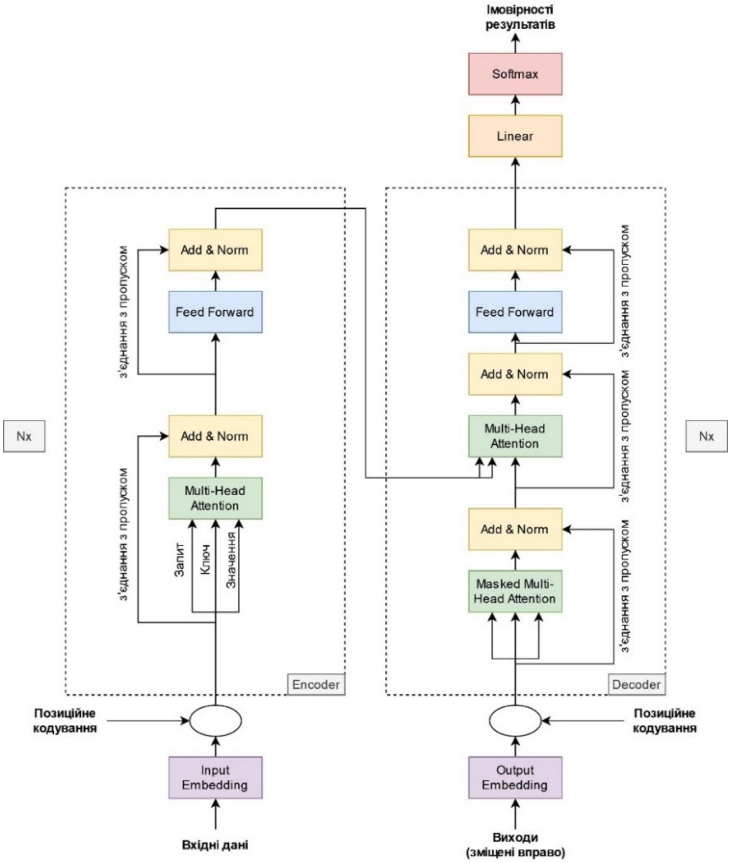

Рисунок 3.2 – Загальна будова архітектури кодера-декодера в моделях трансформерів

Вперше архітектура сімейства трансформерів була представлена у 2017 році, заклавши основу для моделей кодера-декодера на основі трансформера [94].

Аналогічно архітектурам кодера-декодера на основі рекурентних нейронних мереж (RNN), архітектури на основі трансформера також складаються з кодера та декодера які побудовані зі стеків блоків залишкової уваги (рисунок 3.2). Інновація в моделях кодера-декодера на основі трансформера полягає в можливості блоків залишкової уваги обробляти вхідну послідовність $X_{1:n}$ довільної довжини n без застосування рекурентної структури. Відсутність рекурентних залежностей дозволяє досягти високого рівня паралелізації, тим самим значно підвищуючи обчислювальну ефективність на сучасному обладнанні в порівнянні з рекурентними нейронними мережами.

Вирішуючи задачу послідовність-до-послідовності, необхідно встановити відображення від вхідної послідовності $X_{1:n}$ до вихідної послідовності $Y_{1:m}$ змінної довжини m . В цьому контексті моделі кодера-декодера на основі трансформера визначають умовний розподіл імовірностей $p_{W_{\text{enc}},W_{\text{dec}}}(Y_{1:m} \mid X_{1:n})$.

Кодуюча компонента архітектури на основі архітектури трансформера кодує вхідну послідовність $X_{1:n}$ в послідовність прихованих станів $\bar{X}_{1:n}$, тим самим визначаючи відображення $f_{W_{\text{enc}}} : X_{1:n} \rightarrow \bar{X}_{1:n}$.

Подальша декодуюча компонента моделює умовний розподіл імовірностей $p_{W_{\text{dec}}}(Y_{1:n} \mid \bar{X}_{1:n})$. Згідно з теоремою Баєса, цей розподіл може бути факторизований формулою 3.1:

$$p_{W_{\text{dec}}}(Y_{1:n} \mid \bar{X}_{1:n}) = \prod_{i=1}^{n} p_{W_{\text{dec}}}(y_i \mid Y_{0:i-1}, \bar{X}_{1:n}) \qquad (3.1)$$

Декодер відображає послідовність закодованих прихованих станів $\bar{X}_{1:n}$ та всі попередні цільові вектори $Y_{0:i-1}$ на logit-вектор l_i. Цей вектор далі обробляється за допомогою операції softmax для визначення умовного розподілу $p_{W_{\text{dec}}}(y_i \mid Y_{0:i-1}, \bar{X}_{1:n})$. На відміну від рекурентних декодерів, розподіл цільового

вектора y_i базується на всіх попередніх цільових векторах $y_0, \ldots, y_{i-1}$. Початковий цільовий вектор y_0 представлений спеціалізованим вектором "початок речення" (BOS, Beginning of sentence).

Умовний розподіл $p_{W_{dec}}(y_i \mid Y_{0:i-1}, \bar{X}_{1:n})$ сприяє авторегресивній генерації вихідної послідовності, тим самим визначаючи відображення від вхідної послідовності $X_{1:n}$ до вихідної послідовності $Y_{1:m}$ під час виведення.

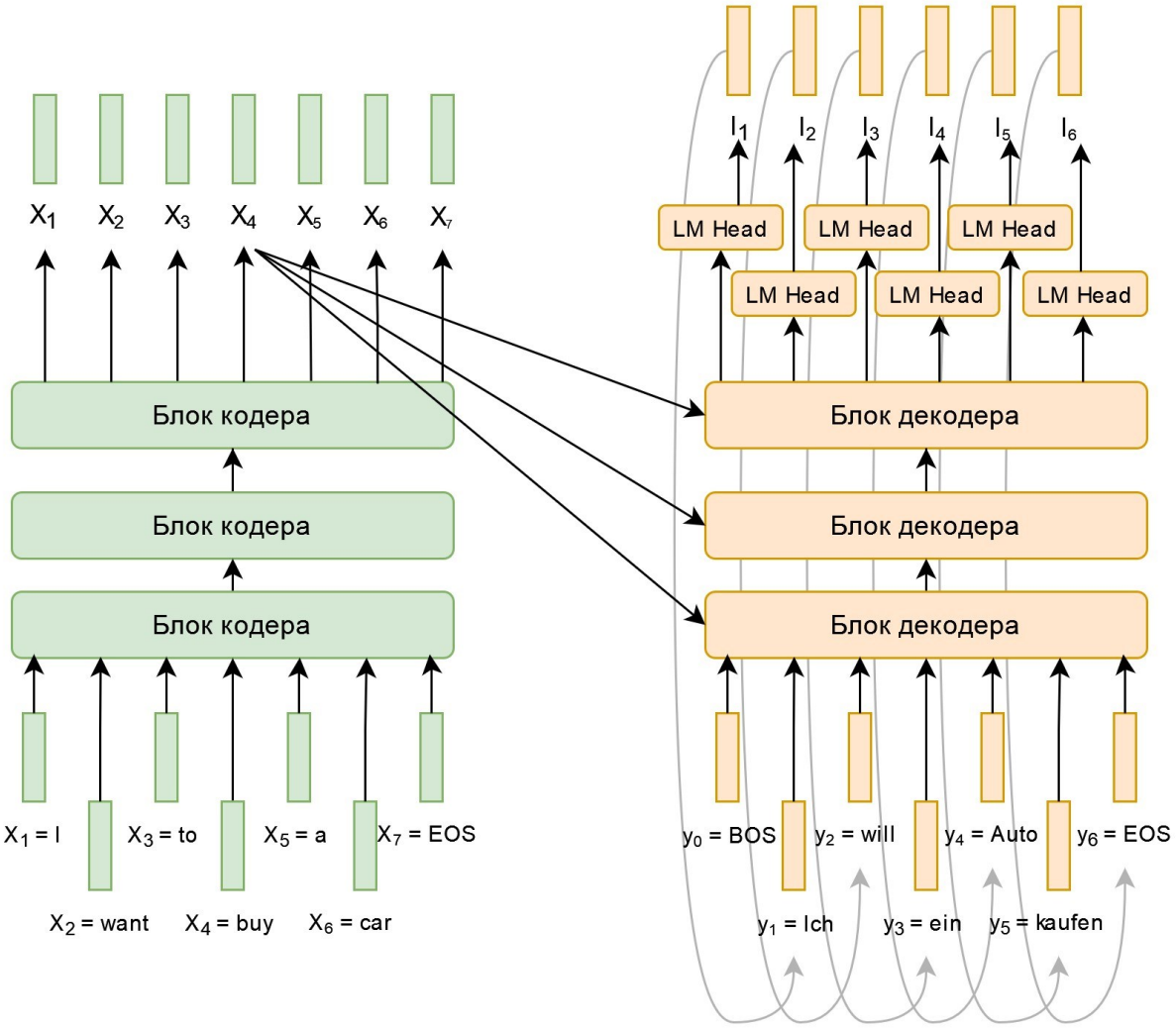

Рисунок 3.3 – Відображення вхідної послідовності до вихідної послідовності в архітектурі транформерів

3.1.1 Обчислення блоку кодеру

Компонент кодера на основі архітектури трансформера відображає вхідну послідовність в послідовність контекстуально збагачених кодувань, формально представлених як $f_{W_{\text{enc}}} : X_{1:n} \rightarrow \bar{X}_{1:n}$

Кодер складається з ряду блоків залишкового кодера, кожен такий блок складається з двонаправленого шару самоуваги, за яким слідують два повнозв'язні шари. Для подальшого моделювання, шари нормалізації опускаються, а роль шарів прямого розповсюдження абстрагується як необхідне вектор-до-вектору перетворення в межах кожного блоку кодера.

Двонаправленний шар самоуваги необхідний для контекстуалізації кожного вхідного вектора x'_j для $j \in \{1, \ldots, n\}$, формуючи співвідношення з усіма іншими вхідними векторами $x'_1, \ldots, x'_n$, та в результаті формуючи уточнене контекстуальну представлення x''_j. Початковий блок кодера перетворює кожний вхідний вектор з контекст-незалежного на контекст-залежне представлення, а подальші блоки кодера уточнюють цю контекстуалізацію, завершуючи кінцевим контекстуальним кодуванням $\bar{X}_{1:n}$.

Механіку двонаправленої самоуваги можна описати наступним чином: кожний вхідний вектор x'_i проектується на вектор ключів k_i, вектор значень v_i та вектор запитів q_i за допомогою матриць ваг W_q, W_v, W_k (формула 3.2):

$$q_i = W_q x'_i, \quad v_i = W_v x'_i, \quad k_i = W_k x'_i, \quad \forall i \in \{1, \ldots, n\} \tag{3.2}$$

Важливо зауважити, що однакові матриці ваг застосовуються однорідно до всіх вхідних векторів. Після цих проекцій кожний вектор запиту q_j порівнюється з усіма ключовими векторами $k_1, \ldots, k_n$. Вихідний вектор x''_j тоді формулюється як зважена сума всіх значущих векторів $v_1, \ldots, v_n$, додатково до вхідного вектора x'_j.

Ваги пропорційні косинусній подібності між q_j та відповідними ключовими векторами, математично виражені як $\mathrm{Softmax}(K_{1:n}^{\top} q_j)$.

Крім полегшення вивчення звязків на великій відстані, можна побачити, що архітектура трансформера здатна обробляти токени паралельно. Математично це можна продемонструвати, використовуючи формулу 3.3 для визначення самоуваги як добуток матриць запиту, ключа та значення:

$$X''_{1:n} = V_{1:n}\mathrm{Softmax}(Q_{1:n}^{\top} K_{1:n}) + X'_{1:n} \tag{3.3}$$

Вихід $X''_{1:n} = x''_1, \ldots, x''_n$ обчислюється через ряд множень матриць та операції softmax, які можна ефективно паралелізувати. Варто зауважити, що в моделі кодера на основі RNN обчислення прихованого стану c вимагає послідовного виконання: спочатку обчислюється прихований стан першого вхідного вектора x_i, потім обчислюється прихований стан другого вхідного вектора, який залежить від прихованого стану першого вхідного вектора, і так далі. Послідовна природа RNN унеможливлює ефективну паралелізацію та робить їх набагато менш ефективними порівняно з моделями кодера на основі трансформера на сучасному обладнанні з GPU.

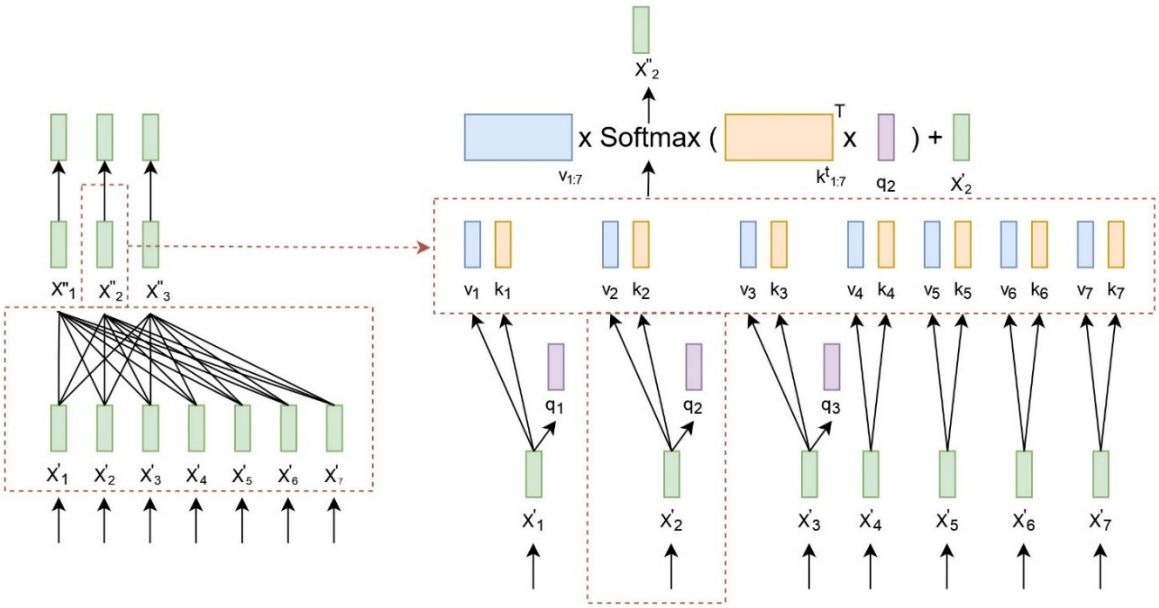

Рисунок 3.4 – Ілюстрація механізму розрахунку уваги для вхідного вектора $x`_2$

3.1.2 Обчислення блоку декодеру

При використанні послідовності кодер-декодер, декодер на основі трансформера формулює умовний розподіл ймовірностей цільової послідовності $Y_{1:m}$ з урахуванням контекстуалізованої кодуючої послідовності $\bar{X}_{1:n}$ - $p_{W_{\text{dec}}}(Y_{1:m} \mid \bar{X}_{1:n})$

Використовуючи теорему Баєса, це можна розкласти на добуток умовних розподілів наступних цільових векторів, залежних від контекстуалізованої кодуючої послідовності та всіх попередніх цільових векторів (формула 3.4):

$$p_{W_{\text{dec}}}(Y_{1:m} \mid \bar{X}_{1:n}) = \prod_{i=1}^{m} p_{W_{\text{dec}}}(y_i \mid Y_{0:i-1}, \bar{X}_{1:n}) \quad (3.4)$$

Щоб зрозуміти, як декодер на основі трансформера визначає цей розподіл ймовірностей, необхідно розглянути його архітектуру: серію блоків декодера, що слідують за щільним шаром, який називається "головою мовної моделі" (LM head). Ця серія блоків декодера відображає контекстуалізовану кодуючу послідовність $\bar{X}_{1:n}$ та послідовність цільових векторів, до якої додано вектор початку послідовності (BOS) і яка обрізана до останнього цільового вектора, тобто $Y_{0:i-1}$, на кодовану послідовність цільових векторів $\bar{Y}_{0:i-1}$. Потім LM head відображає $\bar{Y}_{0:i-1}$ на послідовність векторів логітів $L_{1:n} = l_1, \ldots, l_n$, де розмірність кожного вектора логітів l_i відповідає розміру словника. Таким чином, розподіл ймовірностей по всьому словнику можна отримати, застосувавши функцію softmax до l_i для кожного $i \in \{1, \ldots, n\}$. Логіт вектор, або логіт – це вектор сирих (ненормованих) прогнозів, які генерує модель класифікації, і які зазвичай передаються до функції нормалізації, як softmax.

Математично, умовний розподіл $p_{W_{\text{dec}}}(y_i \mid Y_{0:i-1}, \bar{X}_{1:n})$ для кожного i визначається за формулою 3.5:

$$p_{W_{\text{dec}}}(y \mid \bar{X}_{1:n}, Y_{0:i-1}) = \mathrm{Softmax}(f_{W_{\text{dec}}}(\bar{X}_{1:n}, Y_{0:i-1})) = \mathrm{Softmax}(W_{\text{emb}}^{\top} \bar{y}_{i-1}) = \mathrm{Softmax}(l_i) \quad (3.5)$$

Підсумовуючи, щоб моделювати умовний розподіл цільової послідовності векторів $Y_{1:m}$, цільові вектори $Y_{1:m-1}$ — до яких додано вектор початку послідовності (BOS), тобто y_0 — спершу відображаються разом з контекстуалізованою кодуючою послідовністю $\bar{X}_{1:n}$ на послідовність logit-векторів $L_{1:m}$ (ненормалізовані виходи). Кожний цільовий logit-вектор l_i потім перетворюється на умовний розподіл ймовірностей цільового вектора y_i за допомогою операції softmax. Умовні ймовірності всіх цільових векторів $y_1, \ldots, y_m$ визначаються за формулою 3.4.

На відміну від кодерів на основі трансформера, в декодерах кодований вихідний вектор $\bar{y}_i$ повинен оптимально представляти наступний цільовий вектор y_{i+1}, а не вхідний вектор сам по собі. Крім цього, $\bar{y}_i$ повинен враховувати всю контекстуалізовану кодуючу послідовність $\bar{X}_{1:n}$. Щоб відповідати цим вимогам, кожний блок декодера складається з однонаправленого шару самоуваги, за яким слідують шар перехресної уваги та два повнозв'язних шари. Однонаправлений шар самоуваги корелює кожний зі своїх вхідних векторів y'_j лише з усіма попередніми вхідними векторами y'_i для $i \leq j$ та $j \in \{1, \ldots, n\}$, щоб моделювати розподіл ймовірностей наступних цільових векторів. Шар перехресної уваги корелює кожний зі своїх вхідних векторів y''_j з усіма контекстуалізованими кодуючими векторами $\bar{X}_{1:n}$, щоб умовити розподіл ймовірностей наступних цільових векторів на вхід кодера.

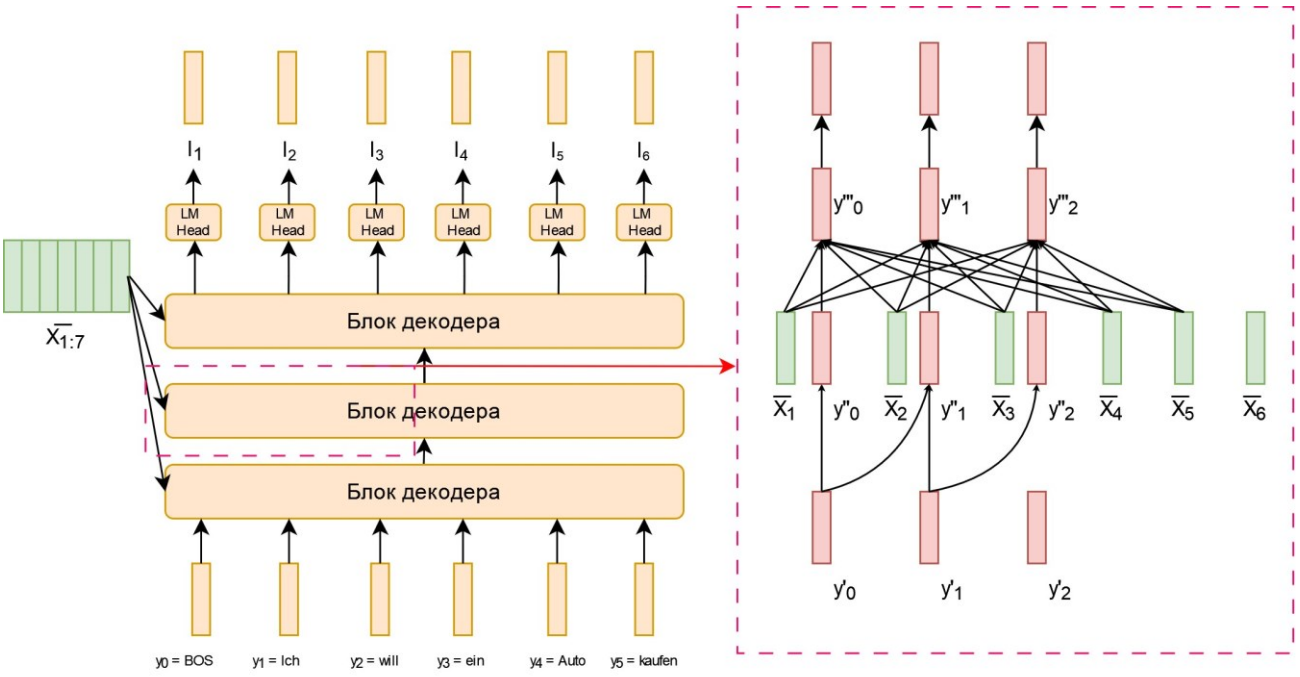

Рисунок 3.5 – Візуалізація механізму перехресної уваги

Серед переваг такої архітектури можна виділити значну паралелізацію операцій. Завдяки відсутності послідовних залежностей шари трансформера можуть паралельно обробляти дані, значно прискорюючи навчання. Механізм самоуваги дозволяє трансформерам враховувати залежності в даних на великій відстані. Архітектура трансформер досягає визначних результатів при тренуванні зі збільшенням даних та обчислювальної потужності. Ця архітектура стала поштовхом/основою для створення таких моделей, як GPT і BERT, які складаються з мільярдів параметрів і досягли найкращих показників на численних тестах NLP.

При цьому обчислювальні вимоги механізму самоуваги квадратично зростають з довжиною послідовності, що може створити проблеми для обробки довгих послідовностей. Методи регуляризації та великі навчальні набори даних часто необхідні для досягнення узагальнення. Це ускладнює використання і навчання таких моделей в середовищах з обмеженими ресурсами.

3.2 Аналіз модифікації архітектури трансформер для вирішення задач розпізнавання образів (ViT)

Крім NLP, універсальність моделей трансформерів була продемонстрована і в інших областях. Наприклад, візуальний трансформер (ViT, Visual Transformer) показав високу ефективність у таких завданнях розпізнавання образів, як класифікація зображень [95].

Основною проблемою застосування трансформерів до зображень є структурований, подібний до сітки характер візуальних даних, що контрастує з послідовним характером текстових даних. Архітектура ViT вирішує цю проблему, розбиваючи зображення на частини (патчі) фіксованого розміру, що не перекриваються, а потім подаючи послідовність патчів у стандартний трансформер.

Перший крок в архітектурі ViT - це розрахунок векторів ознак патчів. Цей крок є ключовим для перетворення вхідного зображення в послідовність ознак, які можна обробити в наступних шарах трансформера. Вхідне зображення I форми $H \times W \times C$ поділяється на неперекриваючі патчі, кожен розміру $P \times P \times C$, де H і W - це висота і ширина зображення, C - кількість каналів, а P - розмір патча. Кількість таких патчів N можна розрахувати за формулою 3.6:

$$N = \left\lfloor \frac{H}{P} \right\rfloor \times \left\lfloor \frac{W}{P} \right\rfloor \tag{3.6}$$

Кожний патч потім розгортається в одновимірний вектор розміру $P \times P \times C$ і лінійно проектується на D -вимірний простір за допомогою навчуваної матриці проекції W_e форми $(P \times P \times C) \times D$. Математично, вектор ознак патча e_i для i^{th} патча розраховується за формулою 3.7:

$$e_i = \text{Linear}(x_i; W_e) = x_i W_e \tag{3.7}$$

де x_i - це розгорнутий вектор i^{th} патча.

Додатково, до кожного вектору ознак патча додається позиційне кодування, щоб надати моделі інформацію про просторове розташування кожного патча в оригінальному зображенні. Кінцевий вектор ознак патча z_i тоді задається за формулою 3.8:

$$z_i = e_i + p_i \tag{3.8}$$

де p_i - це позиційне кодування, що відповідає i^{th} патчу. Послідовність цих кінцевих векторів ознак $\{z_1, z_2, ..., z_N\}$ служить вхідними даними для наступних шарів трансформера для подальшої обробки.

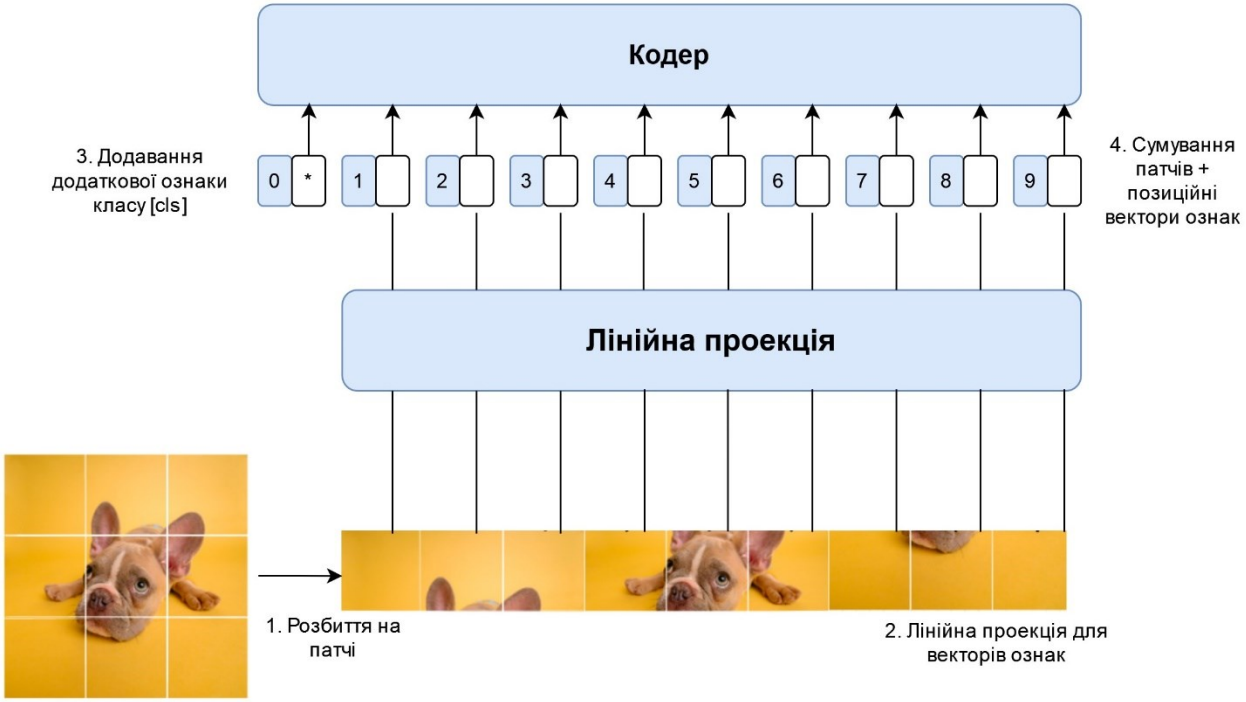

Рисунок 3.6 – Механізм розбиття на патчі і використання трансформерів для обробки зображень

Підсумовуючи, крок обрахування векторів ознак патчів у ViT служить перехідною ланкою між початковими піксельними значеннями вхідного зображення та високорівневими характеристиками, з якими можуть працювати шари трансформера, тим самим дозволяючи застосування трансформерів до візуальних задач.

На відміну від CNN, де рецептивні поля визначаються розміром ядра та операціями об’єднання, механізм самоуваги у ViT використовує динамічні рецептивні поля, дозволяючи моделі зосередитися на глобальних та локальних особливостях одночасно. При використанні ViT, зникає потреба в довготривалому процесі підбору архітектури як в CNN, сприяючи наскрізному навчанню.

Подібно до того, як трансформери в NLP отримують перевагу від попередньо навчених моделей, ViT можуть використовувати величезні обсяги даних для попереднього навчання, а потім дотренеровуватись на менших, специфічних для завдань наборах даних.

3.3 Обгрунтування важливості оптимізації моделей архітектури трансформер

Розглядаючи архітектуру трансформера з боку обчислювальної інтенсивності найбільшу частину займає механізм самоуваги. Механізм самоуваги для кожного токена в послідовності обчислює оцінки уваги по відношенню до кожного іншого токена, що призводить до квадратичного збільшення обчислювальної складності відносно довжини послідовності. Виходить, що для послідовності довжин n, механізм самоуваги вимагає $O(n^2)$ обчислень.

Квадратична складність механізму самоуваги впливає не тільки на обчислювальний час, але і на об’єм використання пам’яті. Зберігання показників уваги для кожної пари токенів у послідовності стає непрактичним для довгих послідовностей. Наприклад, обробка документа з 10 000 токенів вимагатиме зберігання 100 мільйонів показників уваги лише для одного шару однієї частини уваги.

Проблема з квадратичною складністю механізму уваги стає ще більш критичною при навчанні великих моделей трансформерів з мільярдами параметрів, таких як GPT-3 або BERT. Ці моделі, через свою глибину та ширину, вимагають величезних обчислювальних потужностей, тому для їх реалізації потрібно розподілене навчання з використанням декількох графічних процесорів або TPU. Також ці моделі потребують великі об'єми пам'яті, які іноді перевищують можливості навіть високоякісного обладнання, для вирішення чого дослідників використовують методи модельного паралелізму.

Окремим викликом є розгортання нейронних мереж в середовищах з обмеженими ресурсами, такими як периферійні пристрої або вбудовані системи. Одним з рішень може бути використання методів прунінгу. Хоча дані методи початково не були розроблені спеціально для трансформерів, методи прунінгу спрямовані на зменшення складності архітектур нейронних мереж, роблячи їх більш ефективними без значного значення точності.

Розвитком ідеї прунінгу перед навчанням є застосування методу прунінгу нейронних мереж на основі передачі сигналу (SNIP) до архітектур трансформерів, що дозволяє зниження обчислювальної складності, зберігаючи при цьому продуктивність моделі трансформеру. Метод SNIP розроблений для розрахунку важливості вагів за допомогою прямого та зворотного проходу та може слугувати ефективним методом прунінгу перед тренуванням, особливо для архітектур які потребують значних обчислювальних потужностей, таких як трансформери. Проведені емпіричні дослідження підтверджують ефективність методів прунінгу у стисненні моделей без значного зниження продуктивності [96].

3.4 Розробка алгоритму прунінгу для моделей архітектури трансформер

Алгоритм SNIP призначений для прунінгу нейронних мереж перед початком навчання та розраховує критерій важливості для кожної ваги в мережі на основі впливу видалення цієї ваги на функцію втрати. Ваги з нижчими балами обрізаються, що призводить до розрідженої мережі, яку в подальшому можна навчати більш ефективно.

Розрахунок критерію важливості за алгоритмом SNIP описаний в розділі 2. Розроблений алгоритм розрахунку критерію важливості, представлений у цьому дослідженні, передбачає інтеграцію оцінок уваги, які генеруються моделлю трансформера. Механізм уваги дозволяє моделям сфокусуватися на різних частинах вхідних даних з різною силою, подібно до того, як люди акцентують увагу на конкретних деталях при сприйнятті інформації. Увагові оцінки вказують на важливість різних частин вхідної послідовності та використовуються для коригування оцінок важливості ваг.

Запропонований алгоритм включає важливу модифікацію - включення показників уваги до критерію, що оцінюються для обчислення показника важливості. На відміну від алгоритму SNIP, що знаходив розріджені підмножини ваг W_s, де $W_s \subseteq W$, щоб значення функції втрат L було мінімальним, новий критерій включає функцію втрат, активацію уваги та вектор виходу і розраховується за формулою 3.9:

$$L' = L(W) + \sum_j A_j + O \tag{3.9}$$

У новій формулі для критерію, A_j позначає суму виходів для j-го шару уваги, а O позначає суму тензора виходів. Таким чином, доповнена оцінка важливості визначається за формулою 3.10:

$$s_i = \left| \frac{\partial L'(c_i)}{\partial c_i} \right| \quad (3.10)$$

Зворотний прохід для модифікованих критеріїв збільшує значення градієнта в залежності від чутливості кожної окремої ваги в шарах уваги. Враховуючи важливість механізмів уваги в моделях трансформерів, ця інформація є важливою для ефективного прунінгу.

3.5 Підтвердження ефективності розробленого алгоритму для моделей трансформерів

Для проведення експериментів було обрано набір даних Plant Disease Dataset, який містить панорамний огляд листків різноманітних видів рослин (рисунок 3.7).

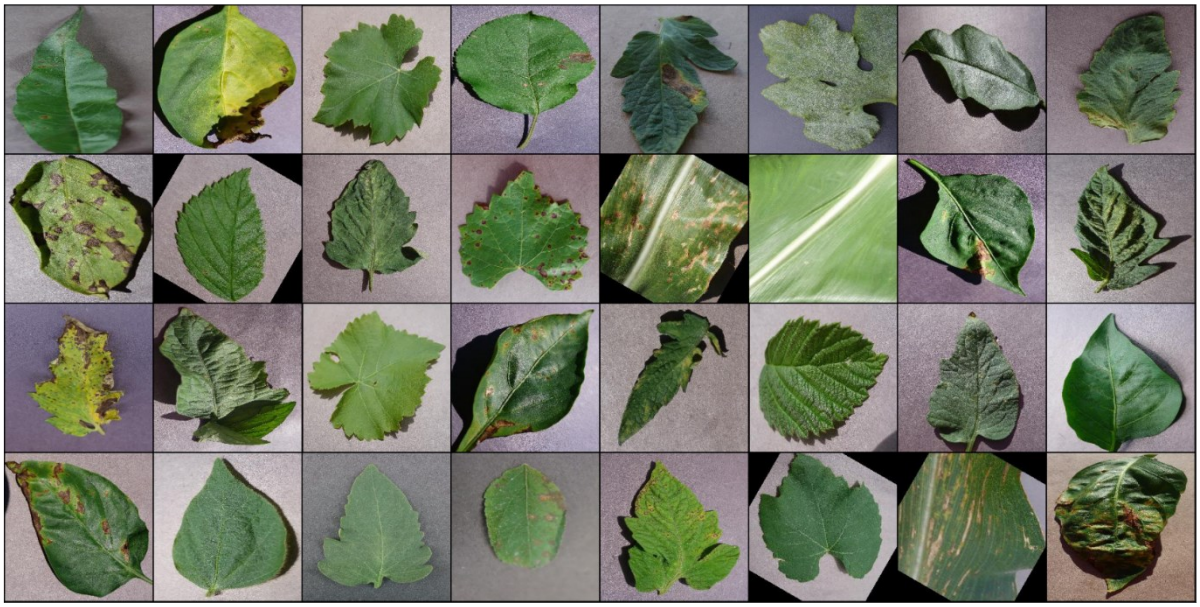

Рисунок 3.7 – Приклад зображень з Plant Disease Dataset

Набір даних Plant Disease Dataset складається з 87 тисяч зображень листків рослин, зображення демонструють різні стадії росту рослин, в різних умовах освітлення і в різних середовищах. Також цей набір даних представляє хвороби,

які вражають широкий спектр рослинних видів, включаючи загальні захворювання як листкова іржа, гниль і в’янення. Кожне зображення в наборі даних має відповідну позначку хвороби, і в деяких розширених версіях також містяться анотації, що включають конкретні області прояву хвороби.

Набір даних Plant Disease Dataset було обрано, тому що він дозволяє натренувати мережу набагато швидше, ніж WIDERFace, оскільки містить меншу складність завдань. При цьому дає можливість оцінити якість отриманої моделі для вирішення задач комп’ютерного зору, а саме класифікації зображень, і методів прунінгу застосованих до неї.

Блок трансформера у моделі ViT модифіковано архітектурою Linformer з метою досягнення обчислювальної ефективності без втрати продуктивності моделі (рисунок 3.8).

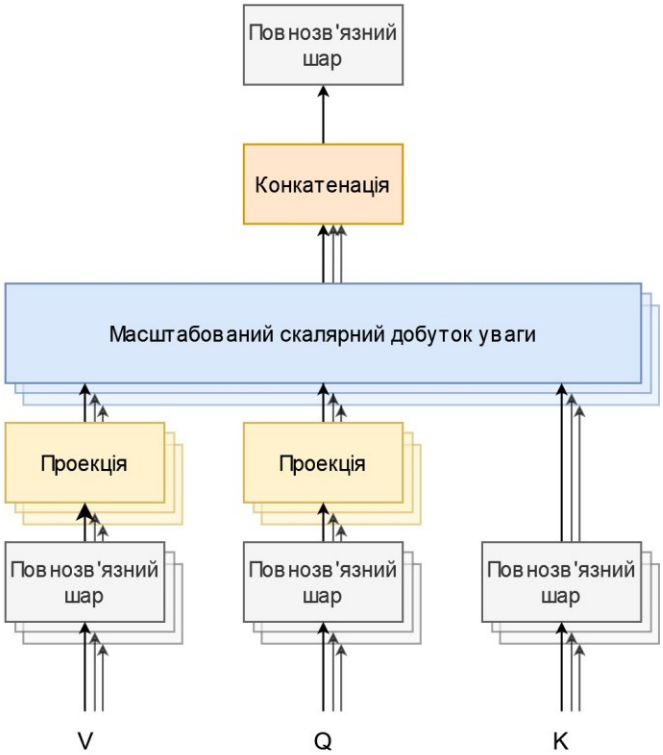

Рисунок 3.8 – Діаграма архітектури моделі Linformer

Архітектура Linformer спроектована для ефективної обробки довгих послідовностей, зменшуючи обчислювальну складність механізму самоуваги з $O(n^2)$ до $O(n)$, де n - це довжина послідовності. Підвищення продуктивності досягається за допомогою фіксованого контексту, лінійній проекції ключів та спільному використанню вагів.

Підхід фіксованого контексту полягає в тому, що можна ефективно наблизити оцінки уваги віддалених токенів в послідовності, не враховуючи кожний інший токен в цій послідовності. Linformer досягає цього, обмежуючи діапазон уваги до фіксованої кількості токенів, забезпечуючи постійну обчислювальну складність незалежно від довжини послідовності [97].

Однією з ключових особливостей Linformer є лінійна проекція ключів і значень у механізмі самоуваги. У стандартному трансформері обчислення самоуваги передбачає взаємодію трьох компонентів для кожного токена у вхідній послідовності, а саме запитів (Q), ключів (K) і значень (V). Вони генеруються шляхом лінійної проекції вхідних даних, потім обчислюються ваги уваги шляхом взяття точкового добутку запитів на ключі, після чого виконується операція softmax. В архітектурі Linformer цей процес змінено шляхом лінійної проекції ключів і значень у простір нижчої розмірності перед обчисленням ваг уваги, така проекція досягається за допомогою лінійного перетворення, що є частиною тренування мережі, і ефективно зменшує розмірність послідовності і обчислювальне навантаження.

Іншим аспектом Linformer є його підхід до ефективності використання вагів. На відміну від стандартних трансформерів, де кожен шар має свій набір вагів, Linformer вводить певний ступінь спільного використання вагів між різними шарами. Це не тільки спрощує архітектуру моделі, але й зменшує загальну кількість параметрів, сприяючи підвищенню ефективності моделі та легкості навчання. Цей аспект спільного використання параметрів є корисним у

сценаріях, де використання пам'яті та ефективність навчання є першочерговим завданням.

У моделі Linformer лінійна увага для кожної голови i обчислюється як $\overline{\text{head}_i} = \bar{P} \cdot (F_i V W_i^V)$ (формула 3.11). Тут $\bar{P}$ – матриця розподілу уваги, яка походить від softmax-нормалізації масштабованого скалярного добутку $\frac{Q W_i^Q (E_i K W_i^K)^T}{\sqrt{d_k}}$. Матриці $Q W_i^Q$, $E_i K W_i^K$, і $F_i V W_i^V$ представляють проекції запитів, ключів і значень відповідно для виходу i-ї голови уваги (рисунок 3.9).

Матриці проекції контексту E_i і F_i включають це наближення низького рангу, ефективно зменшуючи часову складність до $O(n)$:

$$\begin{aligned} \overline{\text{head}_i} &= \text{Attention}(Q W_i^Q, E_i K W_i^K, F_i V W_i^V) \\ &= \underbrace{\text{softmax}\left(\underbrace{\frac{Q W_i^Q (E_i K W_i^K)^T}{\sqrt{d_k}}}_{\bar{P}: n \times k} \right)}_{} \cdot \underbrace{F_i V W_i^V}_{k \times d} \end{aligned} \tag{3.11}$$

Архітектура Linformer в проведеному експерименті налаштована з довжиною послідовності 50, включаючи 49 патчів і один токен класу. Розмір проекційних матриць моделі встановлено на 128, вона складається з 12 шарів (глибина) з 8 головами уваги. Розмір ядра для механізму уваги встановлено на 64.

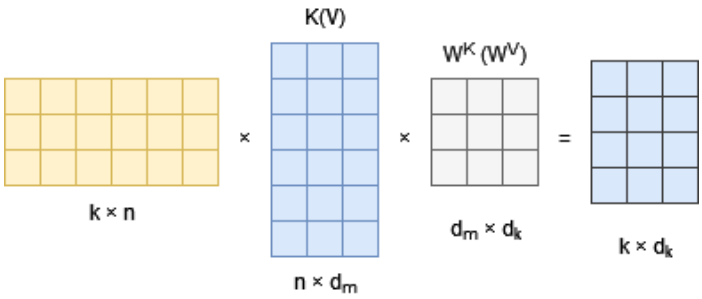

Рисунок 3.9 – Розрахунок проекційних шарів ключів (K) і значень (V) матриць

Блок Linformer інтегровано в модель ViT, спеціально призначену для завдань класифікації зображень. Вхідні зображення масштабуються до розміру

224×224 пікселів і розділяються на патчі розміром 32×32. Потім ці патчі лінійно перетворюються в вектор ознак розмірністю 128. Кількість каналів вхідного зображення встановлено на 3, що відповідає просторові RGB.

Модель навчалася протягом 50 епох, і її ефективність була оцінена порівняно з базовим алгоритмом SNIP. Як показано в таблиці 1, розроблений метод досягнув точності навчання 98,9% і точності перевірки 94,9%, порівняно з немодифікованим методом 67,9% і 57,9%, відповідно [98].

Таблиця 3.1

Результати тренування

Метод	Точність (тренувальна вибірка)	Функція втрат (тренувальна вибірка)	Точність (валідаційна вибірка)	Функція втрат (валідаційна вибірка)
SNIP	67.9	1.01	57.9	1.22
Розроблений метод	98.9	0.03	94.9	0.18

Криві точності і показника F_1, відображені на рисунку 3.10, додатково підтверджують ефективність розробленого алгоритму. Показник F_1 є середнім гармонійним влучності та повноти, і розраховується за формулою 3.12. Для розрахунку показнику обраховують

$$F_1 = \frac{tp}{tp + \frac{1}{2}(fp + fn)} \tag{3.12}$$

, де *tp* - кількості правильно класифікованих представників одного класу, *fp* – кількість об’єктів , хибно віднесених до цього класу класифікатором, а *fn* – кількість представників цього класу, віднесених класифікатором до іншого класу.

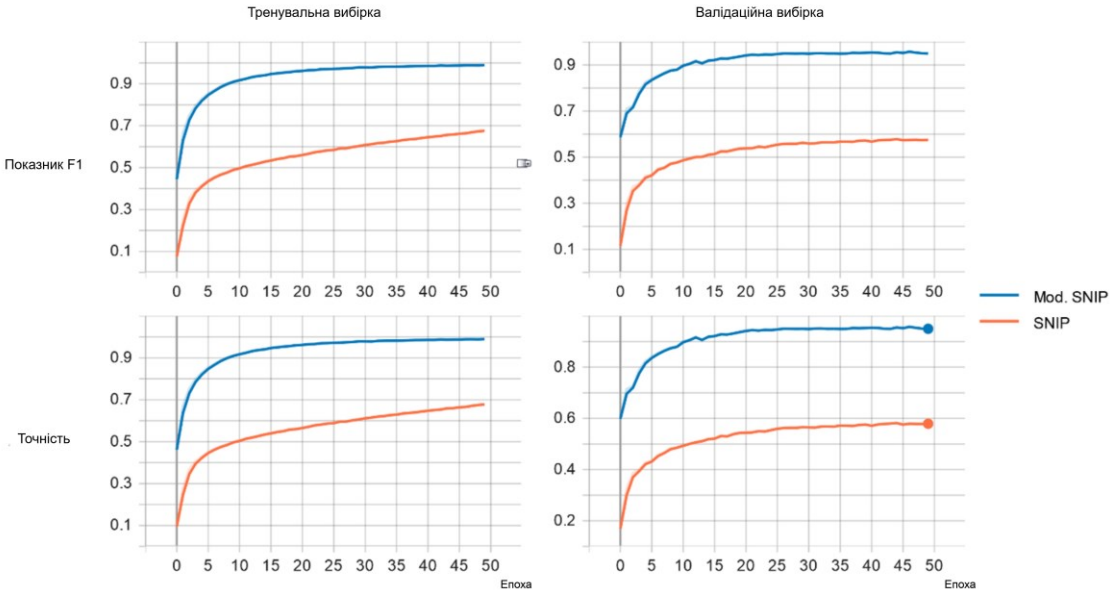

Рисунок 3.10 – Криві точності і показнику F1 відносно навчальних епох

В результаті експерименту було досягнуто покращення точності на тестовій вибірці на 37%. Крім того, модифікований метод також показав значно нижчі значення втрат: втрати при навчанні становлять 0,03, а втрати при перевірці - 0,18, на відміну від 1,01 і 1,22 у базовому варіанті, відповідно. Кінцева модель містить розріджені ваги, що налічуються лише 275,738 параметрів, становлячи лише 10% від початкових параметрів.

3.6 Використання результатів прунінгу для збільшення швидкодії моделей нейронних мереж

Глибокі нейронні мережі - це послідовність обчислювальних блоків, організованих у кілька шарів, які взаємодіють між собою. Обчислювальні витрати на виконання арифметичних операцій можна кількісно оцінити за допомогою операцій з плаваючою комою (FLOP). Операції з плаваючою комою - це міра для кількісної оцінки кількості будь-яких математичних операцій

(додавання, множення тощо) між двома числами з плаваючою комою. Плаваюча кома є стандартним представленням параметрів нейронних мереж (float16, float32). Таким чином, швидкість виконання алгоритму глибокого навчання буде залежати від того, скільки операцій з плаваючою комою в секунду може виконати апаратне забезпечення. Ця інформація зазвичай позначається як FLOPS (FLOPs per Second) і є однією з основних відмінностей у продуктивності між CPU і GPU, що робить GPU більш придатними для обробки операцій глибокого навчання [99]. Крім того, набуло поширення використання нового апаратного забезпечення, відомого як тензорний процесор (Tensor Processing Unit, TPU), розробленого компанією Google для використання у хмарі. Такі процесори спеціально розроблені для високої пропускної здатності при виконанні обчислень для тренування нейронних мереж (Таблиця 3.2).

Таблиця 3.2

Порівняння показників продуктивності CPU, GPU та TPU

Платформа	Версія	Тип пам’яті	Об’єм пам’яті (ГБ)	Пропускна здатність (ГБ\сек)	Показник FLOPS
CPU	Intel Skylake	DDR4	120	16.6	2T
GPU	Nvidia V100	HBM2	16	900	125T
TPU	v2	HBM	8	2400	180T
TPU	v3	HBM	16	3600	420T

Прунінг намагається визначити параметри, які можна вилучити з моделі без шкоди для моделі без суттєвого погіршення точності моделі. Прунінг може надавати прискорення як за рахунок зменшення обчислювальних витрат на

виведення, так і зменшенням кількості параметрів під час навчання. При прунінгу глибокої нейронної мережі ціллю є зменшити кількість параметрів. Стискаючи глибоку нейронну мережу, ми прагнемо зменшити кількість параметрів, а таким чином, обчислювальну складність, зберігаючи при цьому продуктивність вихідної щільної мережі (передбачення початкової щільної мережі (точність передбачення).

Вибір архітектури нейронної мережі відіграє центральну роль у загальній кількості FLOP на модель. Кількість параметрів безпосередньо впливає на кількість математичних операцій, але сучасні архітектури також часто мають блоки, які суттєво збільшують кількість FLOP.

Вимоги до пам’яті моделей глибокого навчання можуть бути дуже високими. Під час фази навчання є потреба зберігати в пам’яті обчислювальний граф, що посилається на значення різних параметрів (вагів) у мережі та їх взаємозв’язок для кроків градієнтного спуску. Крім того, нам також потрібно вмістити в пам’ять пакетів даних для розрахунку градієнтів.

Для запуску моделі після тренування потрібно зберігати лише параметри, оскільки дані зазвичай обробляються по одному екземпляру, і немає необхідності обчислювати градієнти для оновлення параметрів. Хоча потрібно зберігати лише параметри моделі, сучасні моделі можуть налічувати більше 100 мільярдів параметрів, що обмежує використання таких моделей на малопотужних або портативних пристроях.

Навіть коли розмір моделі є меншим - близько 100 мільйонів параметрів, для зберігання моделі все одно потрібно приблизно 500 МБ. Параметри під час навчання зазвичай кодуються у форматі з плаваючою комою 32 біти (повна точність), оскільки він кодує ширший діапазон чисел, що призводить до більш точних обчислень. Можна зменшити точність кодування, щоб зменшити вимоги до пам’яті та прискорити обчислення, наприклад, можна зменшити розрядність

до 16 біт або 8 біт, щоб зменшити витрати пам’яті після тренування без суттєвої втрати продуктивності [100].

За замовчуванням параметри моделей глибокого навчання зберігаються за схемою BitMap (BM), де для зберігання параметрів моделі використовується один біт на параметр. При розрідженій параметризації потрібні лише ненульові параметри. Існує декілька схем для ефективного зберігання ненульових параметрів: серед найпоширеніших - формат стисненого розрідженого рядка (Compressed-Sparse-Row, CSR). Він складається з утримання вказівника на рядок, що вказує на будь-які ненульові значення, пов’язаного з ним індексу стовпця, і самого значення, у вигляді масивів (рисунок 3.11). У такій конфігурації для представлення одного елемента потрібно 3 значення. Щільна мережа повинна бути стиснута мінімум у 3 рази для отримання мінімального прискорення, оскільки існує збільшення накладних витрат при зберіганні ненульових значень .

Рисунок 3.11 – Формат CSR

Це означає, що необхідно досягти певного рівня розрідженості, оскільки більше бітів на елемент для зберігання, зазвичай це означає високий рівень стиснення близько ≈ 90%. Інші методи включають стиснутий розріджений стовпчик (Compressed-Sparse-Column, CSC) - еквівалент CSR, що містить вказівники стовпців, або Координатний зсув (Coordinate Offset, COO) - замість вектора вказівників зберігається індекс стовпця та рядка для кожного

ненульового значення. Попри те що зазначені методи дозволяють ефективно зберігати параметри нейронної мережі, вони все ще мають невеликий вплив на швидкодію.

Під час досліджень в області ефективності використання розріджених матриць на сучасних GPU було розроблено новий формат зберігання даних – розріджена матриця формату 2:4, що надає змогу вирішити проблеми, пов’язані з використанням розріджених матриць для збільшення швидкодії. Формат розрідженості 2:4 передбачає, що для кожної групи з 4 значень, принаймні 2 повинні бути нульовими. Це призводить до 50% розрідженості, що робить підтримання точності без надмірного видалення параметрів набагато практичнішим, ніж наприклад при розрідженості 80%. При застосуванні до матриці, шаблон 2:4 має наступні переваги над альтернативними підходами до розрідженості:

- ефективний доступ до пам’яті;
- стислий формат з низькими накладними витратами;
- збільшення математичної продуктивності в 2 рази на NVIDIA Ampere GPU.

Приклад матриці, яка задовольняє вимогу розрідженості 2:4, показано на рисунку 3.12. У цьому прикладі потрібно зберігати лише 2 ненульових значення у кожній групі з 4 значень. Метадані для декодування стисненого формату зберігаються окремо, використовуючи 2 біти для кодування позиції кожного ненульового значення у групі з 4 значень (індекс позиції в рамках групи, від 0 до 3). Інформація про метадані необхідна для отримання відповідних значень з другої матриці при виконанні множення матриць. Для групи з 4 значень, які мають більше ніж 2 нулі, у форматі стиснений формат все одно зберігатиме 2 значення для збереження послідовного формату.

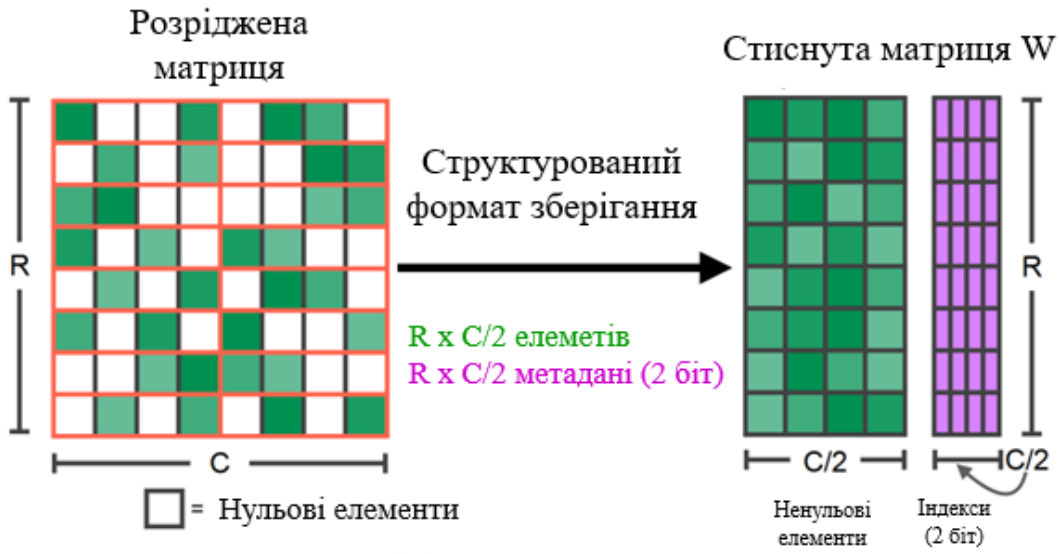

Рисунок 3.12 – Формат зберігання 2:4

Неструктуровані патерни розрідженості неефективно використовують кеш-лінії при доступі до пам’яті, що призводить до недостатнього використання пропускної здатності пам’яті. Крім того, неструктуровані патерни зазвичай використовують формати зберігання CSR/CSC/COO, які призводять до залежного від даних доступу, тим самим збільшуючи затримку при читанні матриць. Розрідженість 2:4 має сталий рівень розрідженості в кожному підблоці більшої матриці, що дозволяє апаратному забезпеченню повністю використовувати переваги прискорення читання великої пам’яті. Аналогічно, оскільки розрідженість є сталою по всій матриці, немає необхідності в опосередкуванні; позиція ненульового значення в пам’яті може бути визначена безпосередньо за ступенем стиснення.

Також, використання формату CSR для неструктурованої розрідженості може призвести до збільшення обсягу пам’яті через метадані до 200%. Для прикладу, 8-бітне квантоване значення ваги: індекс стовпця для значення вимагатиме 16 біт або більше для матриць навіть малого розміру. Завдяки розміру блоку з 4 значень, розріджений формат зберігання 2:4 вимагає лише 2 біти

метаданих на значення, що обмежує накладні витрати на зберігання до 12,5% і 25% для значень 16b і 8b відповідно. Для 16-бітових операндів зберігання розрідженого тензора у стислому форматі призводить до ~44% економії пам'яті: 4 щільних елементи вимагають 4*16 = 64 біт пам'яті, тоді як розрідженість 2:4 призводить до 2*16 біт + 2*2 біт = 36 біт для зберігання двох ненульових елементів. Для 8-бітових операндів зберігання у стислому форматі економить ~38% об'єму пам'яті та пропускної здатності порівняно зі нерозрідженим тензором.

Приклад обробки розріджених матриць графічними процесорами з підтримкою даного функціоналу наведено на рисунку 3.13. Щільна матриця A, розміром M×K, (ліва частина) стає M×K 2 (права частина) після обрізання з розрідженістю 2:4. Апаратне забезпечення Tensor Core вибирає тільки ті елементи з B, які відповідають ненульовим значенням значенням в A, пропускаючи непотрібні множення на нуль. Як у щільних, так і в розріджених GEMM, B і C є щільними матрицями K×N і M×N відповідно

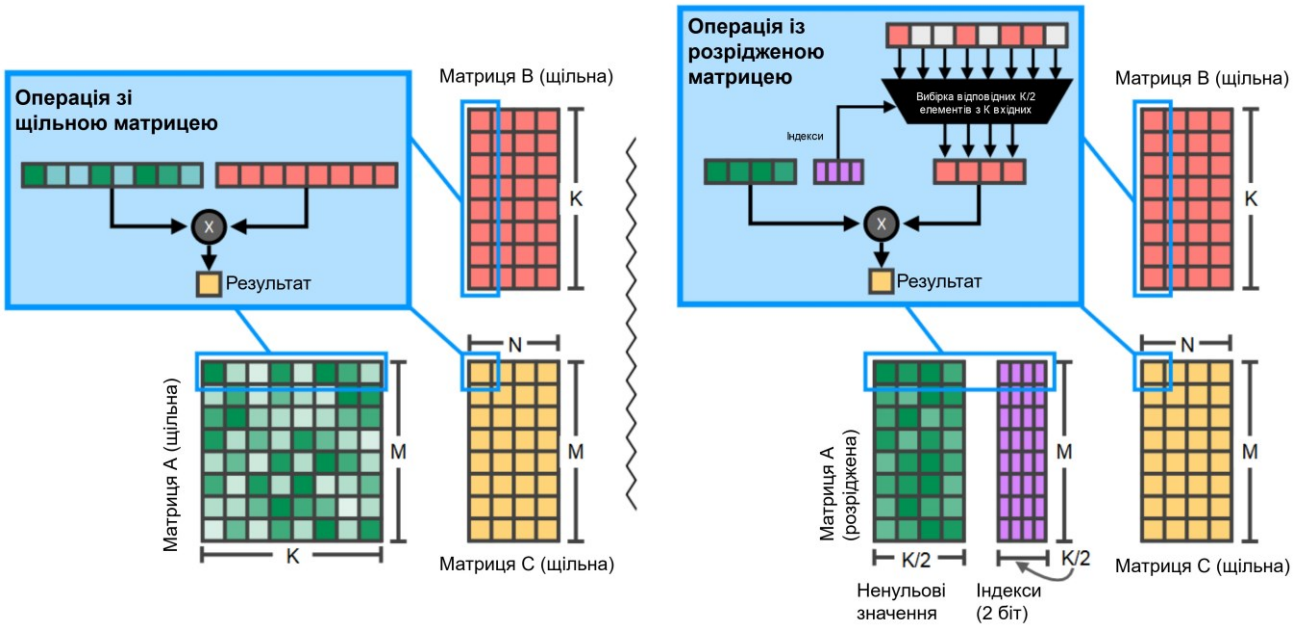

Рисунок 3.13 – Порівняння обчислень з щільними і розрідженими матрицями на тензорних ядрах

Архітектура NVIDIA Ampere GPU розширює тензорні ядра, щоб також обробляти розрідженість 2:4, дозволяючи першому аргументу зберігатися у розрідженому форматі. Таким чином, розріджені тензорні ядра виконують операцію розріджена матриця × щільна матриця = щільна матриця (друга вхідна матриця і вихідна матриця є щільними). На рисунку 3.14 показано, як розріджена операція GEMM (General Matrix Multiplications, Загальне перемноження матриць) 2:4 відображається на тензорні ядра. 50% розрідженість на одному з операндів вдвічі зменшує кількість необхідних операцій множення та додавання, що призводить до 2-кратного збільшення продуктивності порівняно з еквівалентними щільними GEMM.

В результаті тестувань функціоналу бібліотеки PyTorch з розрідженими матрицями, було отримано в середньому прискорення операції множення з розрідженими матрицями на 40.52% при розмірах матриць більше 2048x2048 (рисунок 3.14).

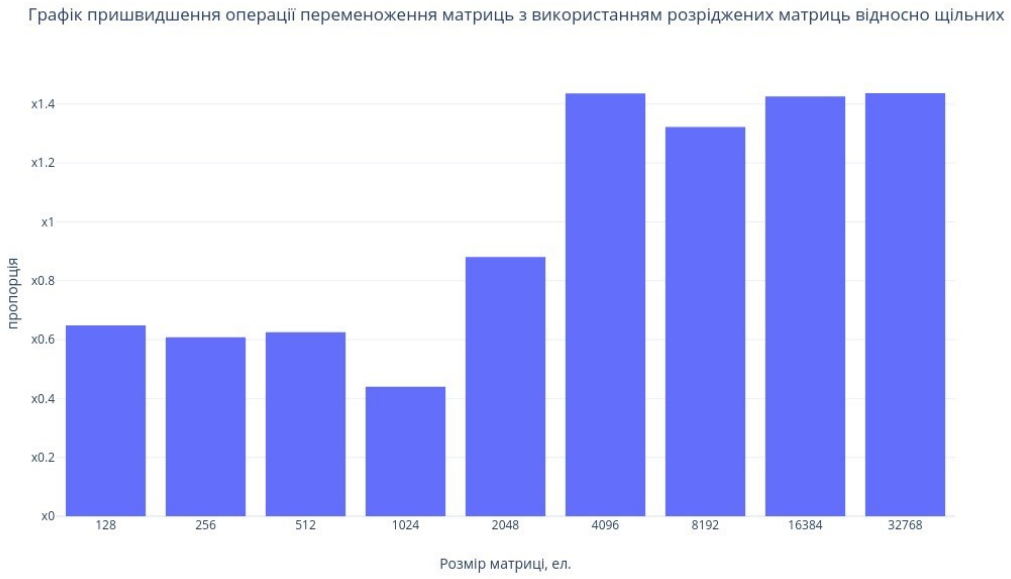

Рисунок 3.14 – Залежність прискорення від розмірів матриць на Nvidia Geforce RTX 3070TI

Інший набір тестів на графічному процесорі, спеціально розробленому для використання при навчачнні нейромереж, показав прискорення до 80% (Рисунок 3.15) [101]

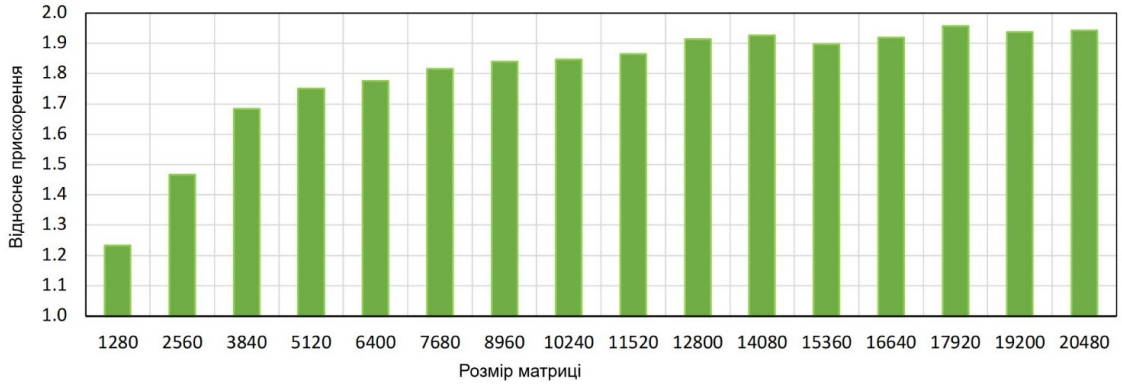

Рисунок 3.15 – Залежність прискорення від розмірів матриць на Nvidia A100

При застосуванні даного методу, стає можливим використання результатів оцінок вагів для застосування гібридного підходу з прунінгу вагів і пришвидшення навчання мережі (рисунок 3.16).

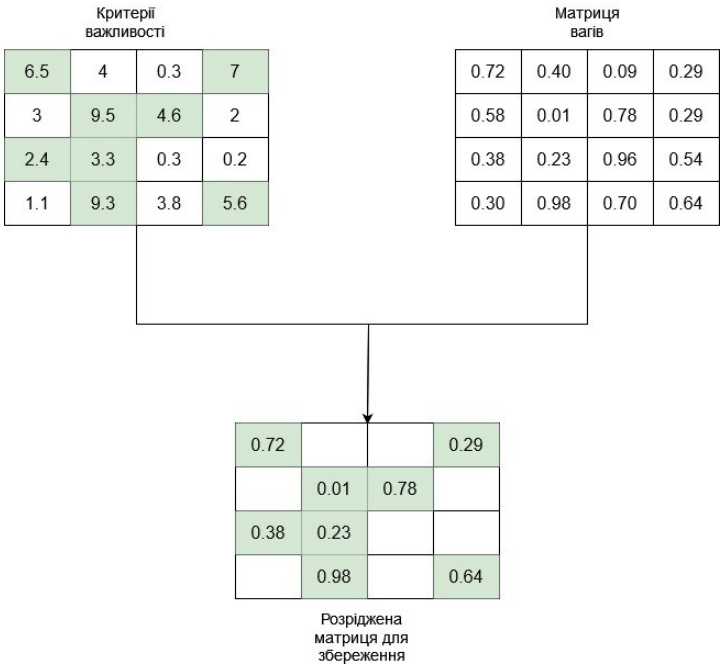

Рисунок 3.16 – Використання критеріїв важливості в поєднанні з розрідженими матрицями 2:4

Оскілки розроблений алгоритм прунінгу, описаний в Розділі 3.6, призначає кожному вазі в мережі критерій важливості, можливо розробити адаптований алгоритм прунінгу, що обиратиме з ваги для розріджених матриць з найбільшими показниками критерію (рисунок 3.16).

Згідно замірів швидкодії окремих елементів і операцій нейронної мережі, зпроектованої в розділі 3.7, 85.80% операцій є перемноженням матриць. Відповідно, для варіацій архітектур трансформерів, використання вказаної модифікації алгоритму прунінгу може призвести до підвищення швидкодії від 20 до 65% в залежності від графічного процесора. (Таблиця 3.3)

Таблиця 3.3

Результати замірів швидкодії окремих операцій під час виконання нейромережі

Відсоток операцій	Витрачений час	Витрати пам’яті (МБ)	Код операції	Назва операції
0.10%	913.980ms	1003.13	aten::linear	Повнозв’язний шар
66.93%	708.760ms	670.13	aten::addmm	Перемноження матриць і додавання вектору (AX + B)
0.03%	200.604ms	333.00	aten::matmul	перемноження матриць
18.87%	199.797ms	333.00	aten::mm	перемноження матриць
0.36%	89.202ms	652.50	aten::einsum	Сумування по нотації Ейнштейна
0.19%	63.316ms	440.00	aten::reshape	Зміна форми матриць
0.08%	59.197ms	440.12	aten::clone	Копіювання
5.41%	57.307ms	0	aten::copy_	Копіювання

Продовження таблиці 2.1

1.97%	20.831ms	326.50	aten::bmm	перемноження матриць (варіація з батчем)
0.01%	14.969ms	20.00	aten::softmax	Активація
1.41%	14.902ms	20.00	aten::_softmax	Активація

В таблиці 3.2 приведені перші 11 позицій з профайлінгу, що становлять 95,36% усіх операцій. Інші позиції, що становлять залишкові 4,64%, опущено з метою кращого представлення результатів.

Висновки до розділу 3

В даному розділі проаналізовано принцип робити нейронних мереж архітектури трансформер, виклики при їх застосуванні і дослідження прискорення швидкодії даних мереж. Мережі архітектури трансформер є одними з найвимогливіших до обчислювальних ресурсів, сучасні застосування налічують сотні мільярдів параметрів, тому питання оптимізації таких мереж є особливо актуальним.

Дослідження методу SNIP на мережах архітектури трансформер показало суттєве падіння точності при використанні даного методу прунінгу перед навчанням. Для експериментів було обрано архітектуру ViT з блоком уваги Linformer, в якості задачі було обрано задачу класифікації зображень хворих та здорових рослин на наборі даних Plant Disease Dataset. В результаті тренування мережі з використанням методу SNIP було отримано мережу, що має в 10 разів менше параметрів, при цьому має точність класифікації 57.9%.

Було описано розроблений алгоритм прунінгу, що включає оцінки уваги до розрахунку критерію важливості. Гіпотеза полягає в тому, що механізм уваги,

ключовий компонент у архітектурі трансформер, може покращити процес обрізки, надаючи більшу важливість вагам, що безпосередньо більше впливають на оцінки вагів. Результати підтверджують, що удосконалений алгоритм має вищу ефективність. Отримана мережа має точність класифікації 94.9%, що на 37% більше, ніж мережа після прунінгу методом SNIP, при цьому має таку саму кількість видалених параметрів.

Було досліджено методи використання розріджених матриць для підвищення швидкодії нейронних мереж після прунінгу. В результаті дослідження було виявлено, що більшість операцій в нейронній мережі включають перемноження матриць, що займає 85% часу при обчисленнях. В результаті аналізу наявних способів використання розріджених матриць було запропоновано підхід до використання технології розріджених матриць формату 2:4 в парі з алгоритмом прунінгу для отримання підвищення швидкодії до 65%.

4. АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ОПТИМІЗАЦІЇ НЕЙРОННИХ МЕРЕЖ ПЕРЕД НАВЧАННЯМ

В даному розділі викладено деталі розробки архітектури програмного забезпечення для оптимізації та вимірювання впливу оптимізації нейронних мереж методами прунінгу перед навчанням, а саме:

- Розроблені компоненти для взаємодії з хмарними обчислювальними ресурсами Azure, які дозволяють проводити тренування та оптимізацію нейронних мереж. Це включає сервіси для аутентифікації користувачів, ініціації тренування та оптимізації, а також збору та відображення метрик;
- Розроблена архітектура додатку, яка включає веб-сервер, ряд сервісів для обробки запитів, бази даних для зберігання інформації про користувачів та метрик, а також сховище для зберігання моделей та контрольних точок;
- Розроблений програмний компонент з реалізаціями розроблених алгоритмів прунінгу перед навчанням, а також компоненти оцінки швидкодії після прунінгу.

4.1 Засоби розробки

Середовищем розробки було обрано PyCharm - інтегроване середовище розробки (IDE) спеціально розроблене для програмування на Python. Середовище PyCharm пропонує ряд функцій, таких як аналіз коду, графічний налагоджувач, інтегрований тестер модулів [102]. Середовище PyCharm є популярним завдяки своїм функціям, що підвищують продуктивність розробки, інтелектуальній підтримці коду та комплексним інструментам розробки. Для реалізації системи

особливо важливим є присутність можливості розробляти та налаштовувати код, що працює на віддалених машинах (наприклад, на GPU-серверах), оскільки розподілена система розгорнута в хмарному середовищі Azure.

Для роботи з базами даних було обрано PostgreSQL - це безкоштовна об’єктно-реляційна система управління базами даних з відкритим вихідним кодом. Дана СУБД зарекомендувала себе своєю надійністю, розширюваністю та відповідністю технічним стандартам. Основні функції PostgreSQL включають підтримку широкого спектру типів даних, надійну індексацію, розширені можливості SQL, мову визначення даних (DDL), значну оптимізацію продуктивності, можливості роботи з JSON, розбиття на розділи та наслідування, а також підтримку представлень та матеріалізованих представлень (materialized view). Крім того, дана СУБД надає широкі можливості безпеки, управління транзакціями та інструменти обслуговування [103].

Для роботи з СУБД було використано середовище розробки DataGrip - крос-платформенне середовище розробки для баз даних та SQL, розроблена компанією JetBrains. Середовище розробки DataGrip підтримує різні бази даних, включаючи PostgreSQL, MySQL, Oracle та інші. Функції DataGrip включають ефективну навігацію за схемами, інтелектуальну консоль запитів та інтеграцію контролю версій.

Для роботи з числовими даними до завантаження на GPU було використано бібліотеку NumPy - фундаментальний пакет для наукових обчислень для Python. Він забезпечує підтримку великих багатовимірних масивів і матриць, а також багату колекцію високорівневих математичних функцій для роботи з ними [104].

Для візуалізації результатів під час розробки і налаштування було використано бібліотеку Matplotlib - комплексна бібліотека для створення статичних, анімованих та інтерактивних візуалізацій для Python. Бібліотеку Matplotlib пропонує широкий спектр функцій побудови графіків та інструментів

для створення різноманітних графіків і діаграм, що робить її основним інструментом для візуалізації даних на Python [105].

Для створення, тренування і збереження моделей нейронних мереж було використано бібліотеку PyToch, основні функції і використання якої описані в наступному підрозділі.

4.2 Використання бібліотеки PyTorch для реалізації нейронних мереж

PyTorch є відкритим програмним засобом для наукових обчислень, який зокрема широко використовується для розробки та тренування нейронних мереж. Бібліотека PyTorch надає потужний інтерфейс для взаємодії з глибокими нейронними мережами, базуючись на динамічному обчислювальному графі, що робить її зручною для наукових досліджень та прототипування.

У сфері машинного навчання та нейронних мереж, обчислювальні графи служать фундаментальною конструкцією для представлення послідовності математичних операцій, які перетворюють вхідні дані в вихідний прогноз. Обчислювальний граф є спрямованим ациклічним графом (DAG), де кожен вузол символізує математичну операцію або змінну, а спрямовані ребра позначають потік даних. Ця формалізація є особливо корисною для кодування складних обчислень та залежностей, які виникають у нейронних мережах.

Обчислювальні графи є дуже практичними при автоматичному диференціюванні, що є основним методом для оптимізації параметрів нейронної мережі. При деякій заданій функції втрат L, яка кількісно визначає розбіжність між прогнозованим та фактичним виходом, мета оптимізації параметрів нейронної мережі полягає в знаходженні набору вагів (параметрів) W, який мінімізує L. Оптимізація вагів досягається за допомогою алгоритмів оптимізації на основі градієнта, таких як стохастичний градієнтний спуск (SGD), де градієнт

∇L обчислюється відносно вагів W. У обчислювальному графі градієнт може бути ефективно обчислений за допомогою техніки зворотнього поширення, яка робить обхід графа в зворотному порядку, застосовуючи ланцюгове правило математичного аналізу для обчислення градієнтів [106].

PyTorch, широко використовувана бібліотека для машинного навчання, використовує динамічні обчислювальні графи. На відміну від статичних графів, де граф визначається до будь-яких обчислень і повторно використовується після цього, динамічні графи будуються в процесі навчання під час кожного прямого проходу через мережу. Ця динамічна структура має високий ступінь гнучкості, дозволяючи використовувати більш складні та адаптивні архітектури мереж, що часто є вимогою в наукових дослідженнях.

Фреймворк PyTorch включає пакет torch.nn, що виступає ключовим компонентом для проектування та реалізації нейронних мереж. Цей пакет охоплює широкий спектр функціональності, надаючи основні будівельні блоки для створення складних моделей машинного навчання. Архітектура torch.nn є модульною, що сприяє складанню складних нейронних архітектур з простіших, повторно використовуваних компонентів. Модульна структура в пакеті torch.nn поєднує зручність інженерії програмного забезпечення з математичними структурами, які є основою досліджень з використанням машинного навчання, тим самим дозволяючи відобразити теоретичні напрацювання на практиці (рисунок 4.1) [107].

Одним з основних елементів у пакеті torch.nn є поняття шару, яке можна розуміти як трансформацію або набір трансформацій, застосованих до його вхідних даних. такі шари як повністю з’єднані шари, згорткові шари та рекурентні шари ініціалізуються як об’єкти і можуть бути скомпоновані для формування нейронної мережі. Кожен об’єкт шару охоплює як пряме обчислення, перетворюючи вхідні дані на вихідні, так і набір параметрів, що навчаються та оптимізуються під час навчання. Абстракції шару в torch.nn

розроблені таким чином, щоб бути легко розширюваними, дозволяючи створювати спеціалізовані шари, які можна інтегрувати в існуючі архітектури.

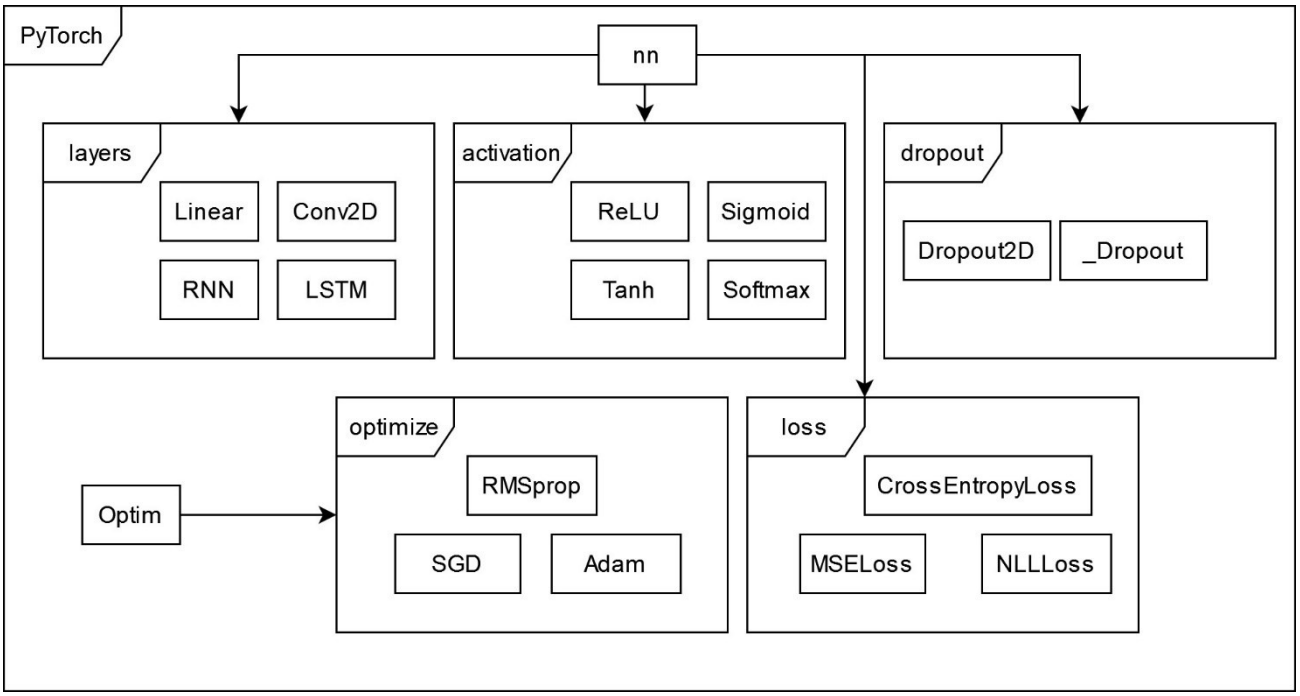

Рисунок 4.1 – Діаграма компонентів пакету torch.nn

Функції активації є ще одним критично важливим компонентом, наданим пакетом torch.nn. Такі функції як ReLU (Rectified Linear Unit), Sigmoid та Tanh використовуються для введення нелінійностей у мережу, дозволяючи їй навчатись та робити складні апроксимації. Функції активації реалізовані як окремі модулі, але вони можуть бути використані як вбудовані функції.

Пакет torch.nn також включає базовий набір функцій втрат, такі як середньоквадратична помилка, перехресна ентропія та дивергенція Кульбака-Лейблера, та інші. Ці функції втрат кількісно визначають наскільки добре нейронна мережа виконує задане завдання і служать функцією об’єкту, яка повинна бути мінімізована під час процесу навчання.

Класи які реалізують алгоритми оптимізації, такі як стохастичний градієнтний спуск (SGD), ADAM та RMSprop, управляють правилами оновлення

для параметрів нейронної мережі під час навчання, і є частинами системи Autograd. Ці оптимізатори працюють у поєднанні з можливостями автоматичного диференціювання обчислювальних графів PyTorch для забезпечення ефективної та точної оптимізації.

Взаємодія між рушієм автоматичного диференціювання Autograd та іншими компонентами PyTorch, такими як оптимізатори в пакеті torch.optim, створює сильний і гнучкий інструментарій для машинного навчання. Оптимізатори використовують градієнти які обчислені за допомогою рушію Autograd для оновлення параметрів моделі відповідно до конкретного алгоритму оптимізації, такого як стохастичний градієнтний спуск (SGD) або ADAM.

Рушій Autograd в PyTorch використовує техніку зворотнього поширення помилки для ефективного обчислення градієнтів. Під час прямого проходу через нейронну мережу, Autograd динамічно будує обчислювальний граф, який відображає всі операції та залежності між ними. Після завершення прямого проходу, система виконує зворотній прохід по цьому графу, використовуючи ланцюгове правило диференціювання для обчислення градієнтів кожного параметра. Алгоритм є фундаментальним застосуванням ланцюгового правила диференціювання для обчислення градієнта функції втрат відносно кожного параметра в нейронній мережі.

Нехай L буде функцією втрат, яка є скаляром, а $W = \{W_1, W_2, \dots, W_n\}$ буде набором всіх ваг в нейронній мережі. Завдання полягає в знаходженні градієнта $\nabla L(W)$, який є вектором, де кожна компонента $\frac{\partial L}{\partial W_i}$ представляє швидкість зміни L відносно W_i.

Нейронна мережа зазвичай структурована як композиція функцій. Нехай $f_1, f_2, \dots, f_m$ будуть цими функціями таким чином, що вихід мережі O задається

$O = f_m(f_{m-1}(\dots f_1(x)\dots))$, де x є входом. Функція втрат L тоді є функцією O, тобто $L = L(O)$.

Для обчислення $\frac{\partial L}{\partial W_i}$, ланцюгове правило застосовується ітеративно від вихідного шару до вхідного шару. Математично, для кожної функції f_j, ми обчислюємо $\frac{\partial L}{\partial f_j}$ за формулою 4.1 :

$$\frac{\partial L}{\partial f_j} = \frac{\partial L}{\partial f_{j+1}} \times \frac{\partial f_{j+1}}{\partial f_j} \tag{4.1}$$

Це робиться рекурсивно, починаючи з $\frac{\partial L}{\partial f_m} = \frac{\partial L}{\partial O}$, яке зазвичай легко обчислити. Як тільки $\frac{\partial L}{\partial f_j}$ відомий для кожного f_j, градієнт $\frac{\partial L}{\partial W_i}$ може бути обчислений за формулою 4.2:

$$\frac{\partial L}{\partial W_i} = \sum_j \frac{\partial L}{\partial f_j} \times \frac{\partial f_j}{\partial W_i} \tag{4.2}$$

У PyTorch цей процес автоматизований рушієм Autograd. Під час прямого проходу Autograd динамічно будує обчислювальний граф, який записує всі операції, виконані на тензорах, що мають атрибут `requires_grad`. Кожен вузол в цьому графі відповідає тензору, а кожен край відповідає операції, яка генерує один тензор з іншого.

Під час зворотного проходу Autograd проходить граф від виходу (функції втрат) тензора назад до кожного з параметричних тензорів, застосовуючи ланцюгове правило для обчислення градієнтів. Градієнти зберігаються в атрибуті `grad` відповідних параметричних тензорів, готові до використання алгоритмами оптимізації для оновлення параметрів.

Прикладом обчислювального графу можна вважати архітектуру ResNet-50, глибоку залишкову мережу, що складається з 50 шарів, яку було використано в якості основи для моделі виявлення облич, що описана в розділі 2. Залишкові

з’єднання в цій архітектурі можна розглядати як скорочення, які обходять один або кілька шарів, полегшуючи навчання глибоких мереж шляхом зменшення проблеми затухаючих/вибухових градієнтів. Обчислювальний граф для такої архітектури включатиме вузли, що представляють згорткові шари, батч-нормалізацію, функції активації та самі залишкові з’єднання.

4.3 Реалізація компоненту для проведення прунінгу перед тренуванням

Для реалізації методів оптимізації перед тренуванням було розроблено ієрархічну структуру класів з імплементаціями відповідних методів. Відповідно до алгоритмів прунінгу, описаних в підрозділах 2.4, 3.4, для розрахунку критеріїв кожен набір вагів має містити маскуючі ваги (маски) зі значеннями 0 або 1. Відповідно для адаптації архітектур нейронних мереж і застосування прунінгу було розроблено функціонал для додавання масок під час створення мережі.

Оскільки фреймворк PyTorch має модульну структуру, нейронні мережі побудовані з сукупності базових блоків. Для додавання масок до таких блоків було реалізовано набір класів, що адаптують найпоширеніші види шарів нейронних мереж, а саме повнозв’язний шар (клас Linear), та згортковий шар (клас Conv2d).

Основний клас, `PruningMethod`, служить абстрактним базовим класом, який окреслює фундаментальну структуру та методи, необхідні для будь-якої стратегії прунінгу. Цей клас розширюється конкретними стратегіями для сприяння повторному використанню коду.

Для ініціалізації абстрактного класу `PruningMethod` використовується конструктор, що приймає список маскованих параметрів, які є параметрами нейронної мережі, що підлягають прунінгу. Він містить кілька методів, кожен з яких служить конкретній меті в процесі прунінгу (рисунок 4.2):

- метод є абстрактним і призначений для перевизначення підкласами для реалізації механізму оцінки, який ранжує важливість кожного параметра;

- метод `calc_masks` оновлює маски на основі обчислених критеріїв важливості та заданого рівня стиснення;

- метод `calc_stats` слугує для розрахунку кількості параметрів мережі, шляхом підрахунку ненульових коефіцієнтів в масках.

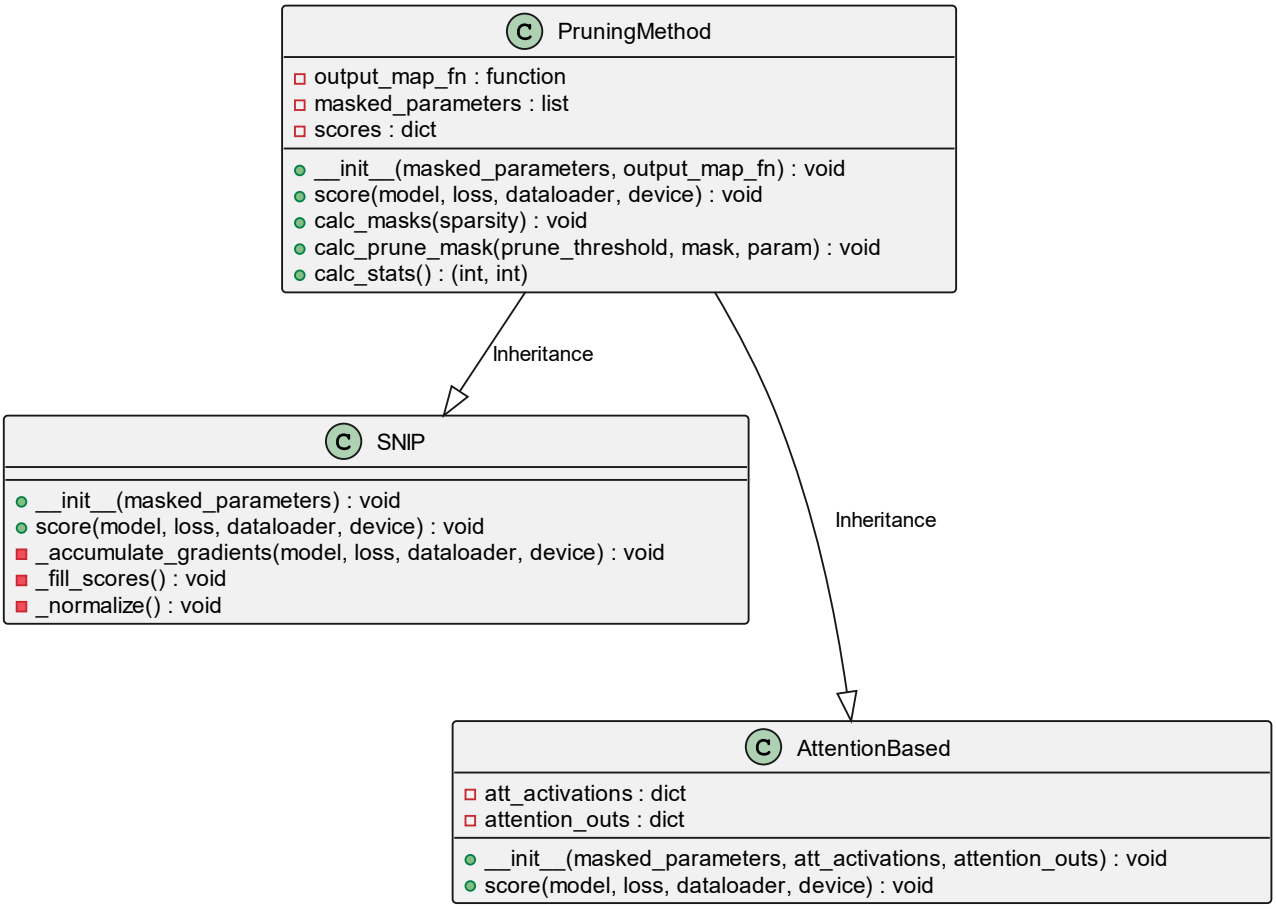

Рисунок 4.2 – Ієрархія класів для прунінгу

Клас `SNIP` розширює `PruningMethod` та надає конкретну реалізацію методу `score` на основі алгоритму SNIP. Він розраховує оцінки, дозволяючи маскам мати градієнти, а потім виконує зворотний прохід для обчислення градієнтів втрат відносно кожного параметра. Оцінки потім нормалізуються, надаючи відносну міру важливості для кожного параметра.

Клас `AttentionBased` також розширює `PruningMethod`, але вводить додаткові параметри конструктору, такі як словник посилань на тензори активації та тензори виходів шарів уваг. Метод `score` в даному класі не тільки враховує втрати, але також включає суму активацій та виходів уваги в зворотний прохід, таким чином імплементуючи метод, описаний у підрозділі 3.4.

Модуль прунінгу реалізовано у вигляді бібліотеки для мови python, що дозволяє використовувати бібліотеку в інших проектах для дослідження і оптимізації моделей розробленими алгоритмами прунінгу. Вхідні і вихідні дані, необхідні для роботи компоненту бібліотеки можна побачити на рисунку 4.3.

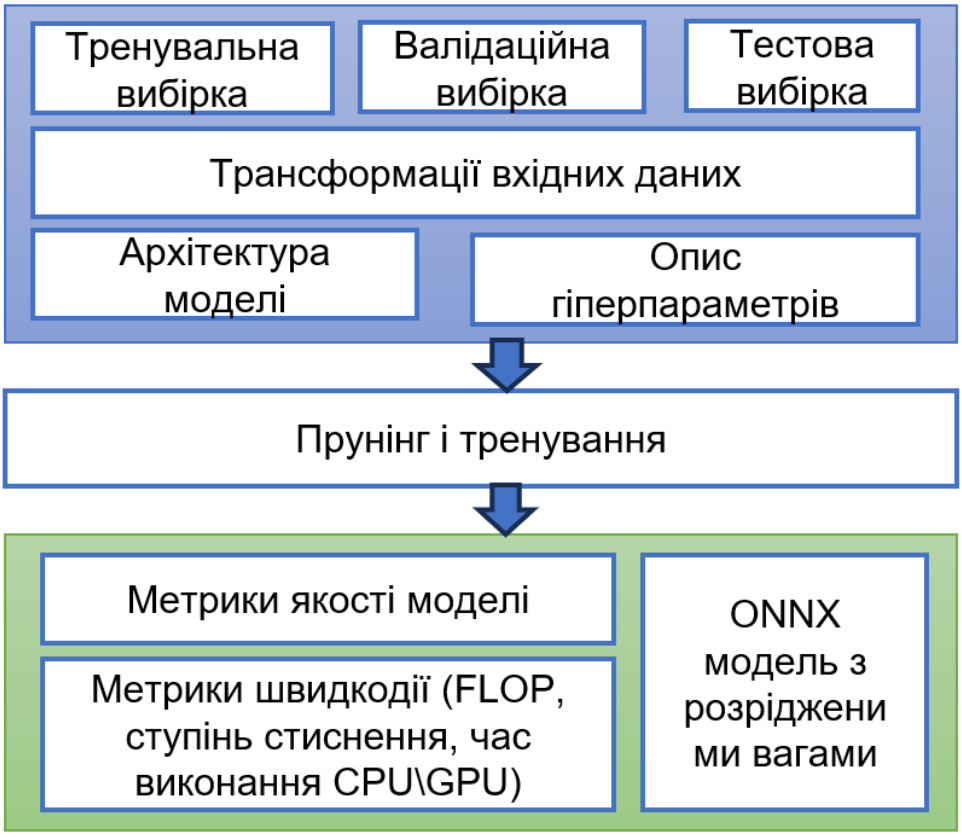

Рисунок 4.3 – Вхідні і вихідні дані для роботи компоненту бібліотеки

Для оцінки швидкодії моделей після прунінгу запропоновано 2 підходи – оцінка часу виконання на GPU та CPU.

- Для оцінки швидкодії на CPU використовується бібліотека deepsparse, що надає рушій для запуску моделей нейронних мереж в форматі ONNX [108].

Бібліотека містить оптимізації обчислень низького рівня, що дозволяють отримати приріст швидкодії на сучасних CPU

- Для оцінки швидкодії на GPU використовуються розріджені матриці формату 2:4, що описаний в підрозділі 3.6, в разі якщо розмірність вагів моделі кратна 4. Для приведення вагів до розрідженого формату використовується функція `to_sparse_semi_structured` модулю `torch.sparse`

4.4 Архітектура програмного забезпечення для оптимізації і тренування нейронних мереж методами прунінгу перед навчанням

Архітектура програмної системи забезпечує виконання вимог до програмної системи для оптимізації і тренування нейронних мереж, які сформульовані в підрозділі 1.4. Архітектуру імплементовано використовуючи мікросервісний підхід, що надає змогу легко розширювати систему. У запропонованій архітектурі системи для оптимізації і тренування нейронних мереж, веб-сервер є вхідною точкою для обробки запитів з клієнтських пристроїв. Розташований в інфраструктурі хмари Azure, веб-сервер є комплексною сутністю, що включає в себе балансувальник навантаження та API-шлюз для забезпечення безперервної роботи та масштабованості системи (рисунок 4.4).

Балансувальник навантаження є початковою точкою контакту для вхідних HTTP-запитів з користувацьких пристроїв та відповідає за розподіл вхідних запитів між кількома серверними застосунками для балансування навантаження та забезпечення високої доступності зменшуючи ризик єдиної точки відмови та оптимізує використання ресурсів, тим самим підвищуючи загальну продуктивність та надійність системи. Використання балансувальника навантаження в комбінації з мікросервісною архітектурою надає можливість будувати гнучкі системи, що доведено попередніми дослідженнями [109].

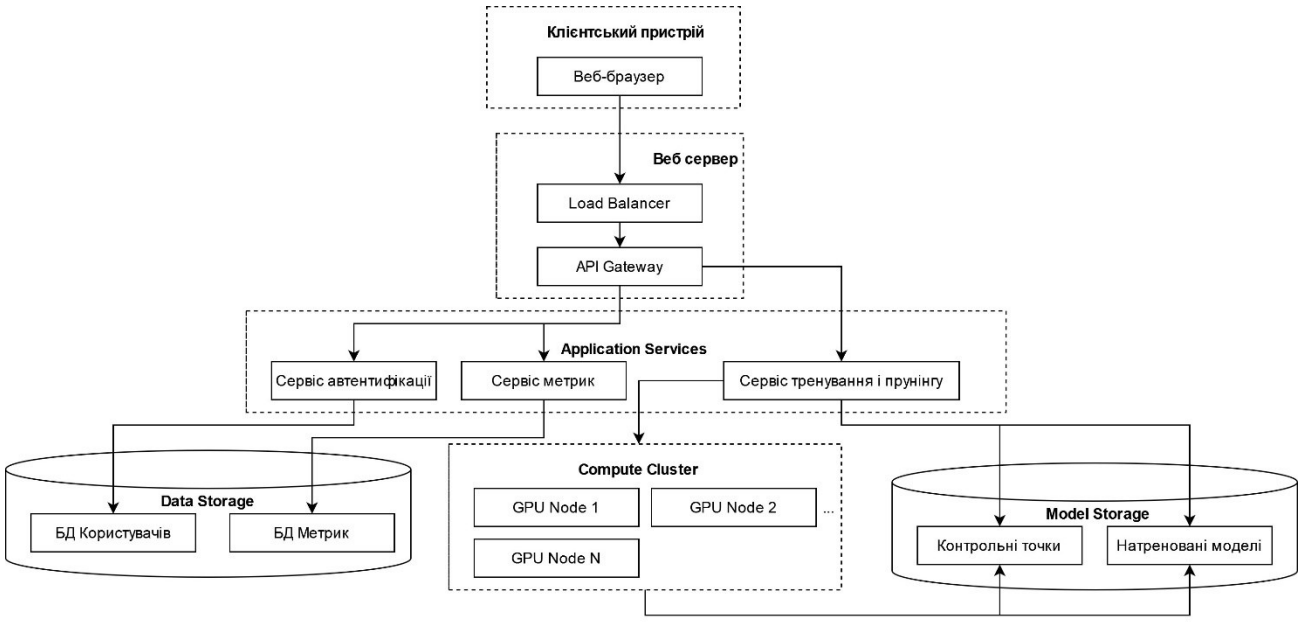

Рисунок 4.4 – Архітектура програмної системи.

API-шлюз, з іншого боку, виступає як оркестратор внутрішніх сервісів системи. Після того, як балансувальник навантаження перенаправляє запит на API-шлюз, обирає потрібну кінцеву точку для перенаправлення цього запиту до відповідного сервісу. Це включає аутентифікацію користувача, запит на навчання нейронної мережі і прунінг або збір метрик. Кожна функціональність інкапсульована в окремому спеціалізованому сервісі. Таким чином, API-шлюз діє як централізований маршрутизатор, що роз’єднує клієнтський запит від виконання на сервері, тим самим сприяючи модульності та зручності обслуговування.

У запропонованій архітектурі шар сервісів акумулює всю програмну логіку застосунку, включаючи функціональні можливості, необхідні для роботи системи навчання та оптимізації нейронних мереж у хмарному середовищі. Цей шар поділено на три основні сервіси: сервіс аутентифікації, сервіс тренування і прунінгу та сервіс метрик, кожен з яких відповідає за окремий набір завдань.

4.5 Програмна реалізація модулю автентифікації

Для реалізації автентифікації клієнтських застосунків у кожній частині додатку використовуються технологія JSON Web Token (JWT). У сфері безпечних та масштабованих розподілених систем JWT виступають як надійний механізм для аутентифікації та авторизації. Перевагами JWT є компактність та самодостатність токенів які включають в себе атрибути користувача такі як унікальний ідентифікатор, роль, доступні операції в системі та інші. На відміну від традиційних систем аутентифікації на основі сесій, де стан сесій зберігається на сервері, використання JWT не потребує збереження сесій. Ця відсутність стану є особливо вигідною для масштабованості, оскільки вона усуває потребу в розподілених рішеннях для зберігання сесій при розгортанні служб на кількох вузлах.

З точки зору клієнтського застосунку, JWT - це рядок, закодований в Base64URL, який складається з трьох частин: заголовка, корисного навантаження та підпису. Заголовок, як правило, оголошує тип токена та криптографічний алгоритм, який використовується для захисту токена. Корисне навантаження містить атрибути користувача. Підпис обчислюється за допомогою секретного ключа, заголовка та корисного навантаження, забезпечуючи цілісність та автентичність токена. Математично підпис S можна представити як $S = \mathrm{Sign}(\mathrm{Base64URL}(\mathrm{Header}) \,||'.'|| \,\mathrm{Base64URL}(\mathrm{Payload}), \mathrm{Secret})$, де Sign - криптографічна функція підпису, а || позначає конкатенацію (рисунок 4.5).

Після успішної аутентифікації сервер генерує JWT, який надсилається назад клієнту. Для наступних запитів клієнт додає JWT в заголовок авторизації HTTP, а сервер потім перевіряє токен, декодуючи його та перевіряючи його підпис. Після перевірки токену на дійсність, сервер продовжує обробку запиту на основі атрибутів користувача, що містяться в корисному навантаженні. Весь цей процес проводиться без необхідності звертання до централізованого сховища даних, зменшуючи таким чином затримку та покращуючи продуктивність.

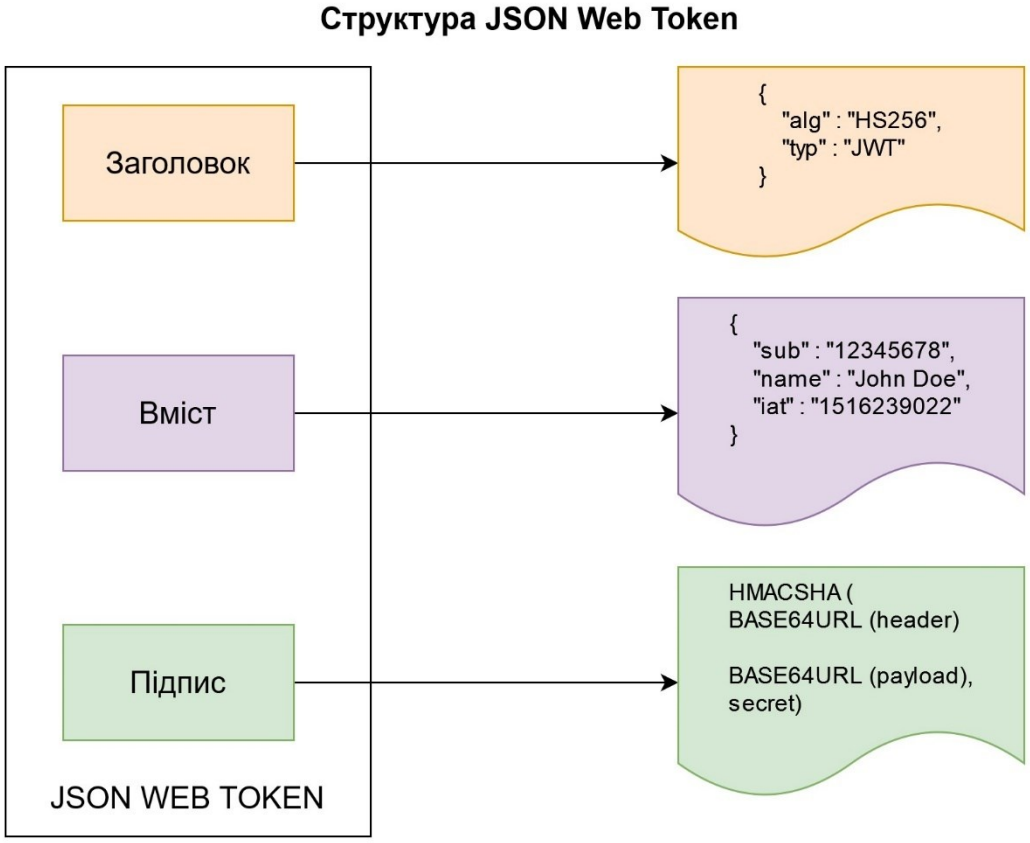

Рисунок 4.5 – Структура JWT

Використання JWT має кілька переваг з точки зору безпеки:

- Оскільки токени є самодостатніми, вони можуть включати всю необхідну інформацію, зменшуючи потребу в запитах до баз даних або систем кешування, тим самим мінімізуючи поверхню для атаки.

- Використання криптографічних підписів забезпечує цілісність даних та підтверджує ідентичність відправника, зменшуючи ризик атак "людина посередині".

- JWT можна легко анулювати або оновити, реалізувавши стратегії чорного списку токенів або використовуючи токени з коротким терміном дії з механізмами оновлення.

Для реалізації аутенифікації за допомогою JWT було обрано технологію з відкритим вихідним кодом Supabase Auth та протокол OAuth 2.0 + PKCE.

Рисунок 4.6. – Діаграма послідовностей протоколу автентифікації OAuth 2.0 + PKCE

Спеціалізована служба Supabase Auth пропонує набір функцій аутентифікації, включаючи входи без пароля, багатофакторну аутентифікацію та підтримку OAuth для входу через сторонні сервіси. Використання Supabase Auth як зовнішньої служби покращує систему з точки зору безпеки та архітектурної модульності.

Ініціюючи запит на аутентифікацію через користувацький пристрій, запит проходить через балансувальник навантаження та API-шлюз у веб-сервері потім досягаючи екземпляра Supabase Auth. Служба Supabase Auth потім виконує необхідні кроки верифікації, які можуть охоплювати діапазон процедур від простої перевірки пароля до більш складних механізмів багатофакторної аутентифікації. Після успішної верифікації генерується JWT, який повертається користувачеві для подальших взаємодій з системою.

4.6 Програмна реалізація сервісу метрик

В окресленій архітектурі для реалізації сервісу метрик використовується утиліта TensorBoard, що слугує компонентом для моніторингу, візуалізації та аналізу метрик продуктивності та обчислювальних графів нейромережевих моделей. TensorBoard, інструмент, спочатку розроблений для екосистеми TensorFlow, але також сумісний з PyTorch, містить веб-інтерфейс, який забезпечує розуміння різних аспектів експериментів з машинного навчання в режимі реального часу. Цей сервіс особливо важливий для ретельного аналізу ефективності процесів навчання та прунінгу, тим самим сприяючи досягненню головних цілей оптимізації та інтерпретації моделі.

Після ініціювання навчання відповідним сервісом, сервіс метрик може бути використаний для візуалізації аспектів нейронної моделі і метрик під час навчання.

Крім того, сервіс фіксує обчислювальні графіки, гістограми вагових розподілів, пропонуючи багатогранний погляд на поведінку та характеристики моделі.

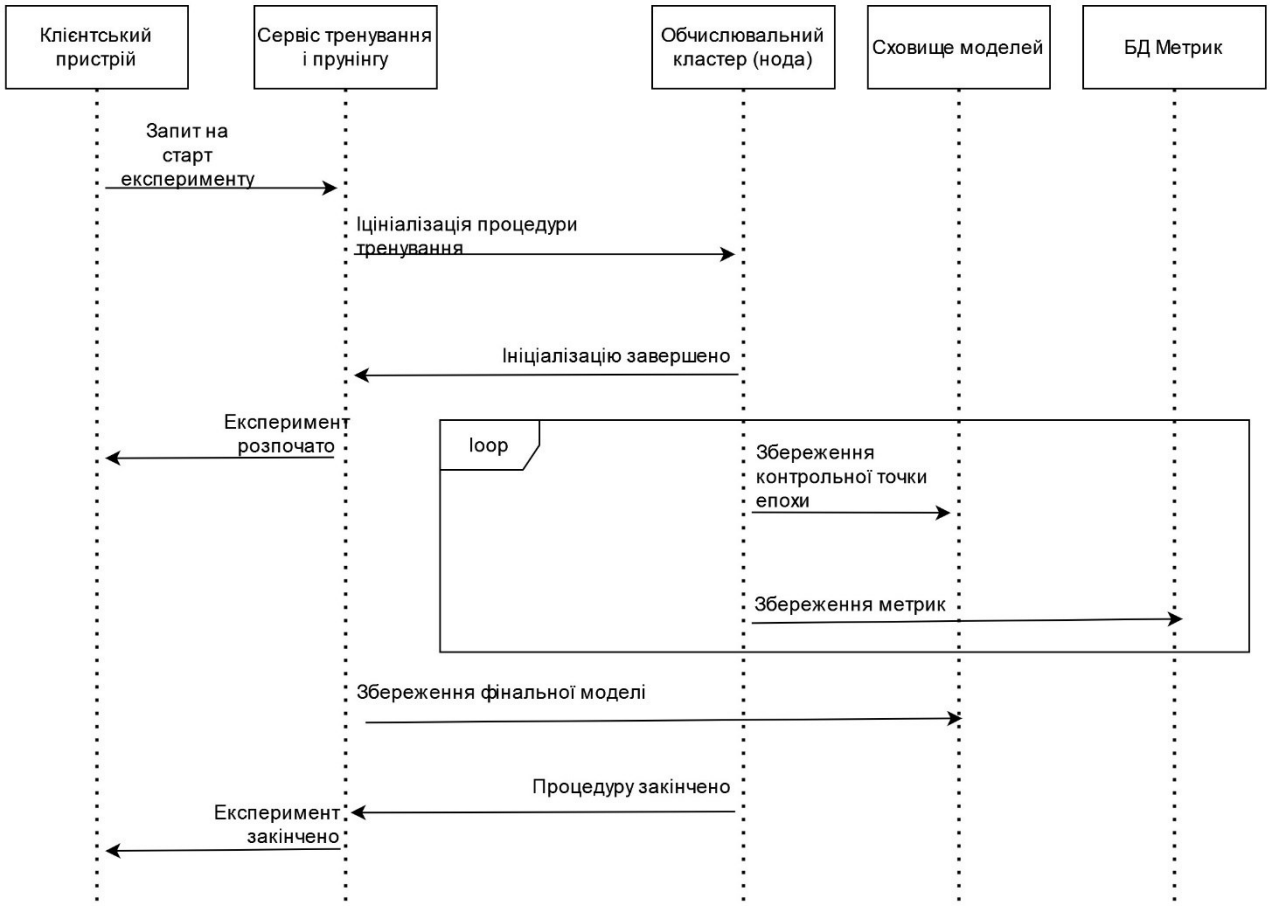

Рисунок 4.7 – Діаграма послідовності фіксації метрик

Однією з найважливіших особливостей TensorBoard є утиліта Graphs, яка надає топологічне представлення архітектури нейронної мережі. Це графічне зображення слугує наочним посібником для розуміння послідовності операцій та ієрархічного розташування шарів у моделі так як кожен вузол на графі відповідає операції, такій як згортка або функція активації, в той час як ребра представляють тензори, що проходять між цими операціями (рисунок 4.8).

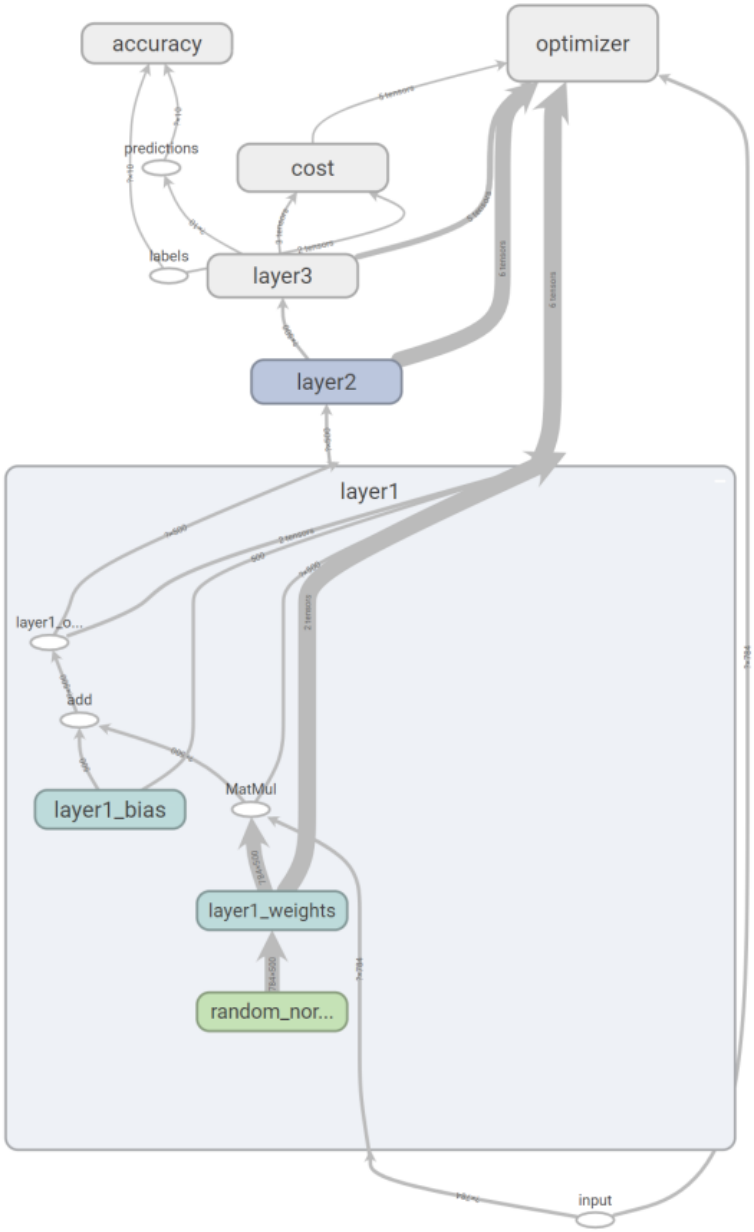

Рисунок 4.8 – Зразок фіксації графу

Це дозволяє інтуїтивно зрозуміти складність і масштаб моделі, тим самим допомагаючи в процесах налагодження та оптимізації. Крім того, утиліта Graphs дозволяє перевіряти форми і типи тензорів, пропонуючи детальний перегляд, що особливо корисно для забезпечення архітектурної узгодженості і налагодження проблем розмірності.

4.7 Програмна реалізація сервісу прунінгу і тренування

В запропонованій архітектурі сервіс прунінгу і тренування використовує можливості Azure Machine Learning Studio для виділення ресурсів для навчання і запуску відповідних процесів [110] (рисунок 4.9).

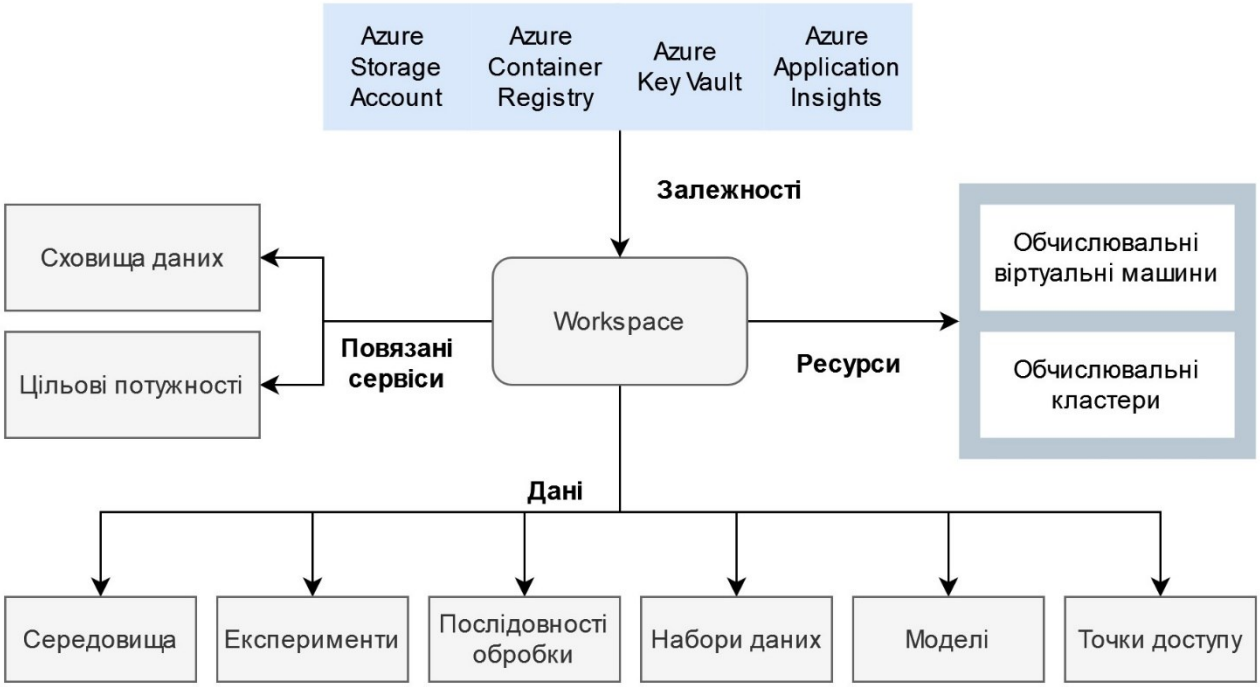

Рисунок 4.9 – Архітектура Azure Machine Learning Studio.

Azure Machine Learning Studio - це інтегроване середовище, яке містить ряд сервісів, що виконують функції від отримання даних до розгортання моделі.

Для управління даними Azure Machine Learning Studio надає спеціалізований компонент "Набори даних" (Datasets). Цей компонент розроблений для роботи з широким спектром типів даних, включаючи структуровані дані у вигляді файлів CSV, TSV і Parquet, неструктуровані дані, такі як зображення і текст, і напівструктуровані дані, такі як JSON і XML. Така універсальність гарантує, що набори даних можуть слугувати єдиним сховищем

для різноманітних завдань машинного навчання - від обробки природної мови до комп’ютерного зору.

Компонент "Моделі" (Models) в Azure Machine Learning Studio підтримує більшість фреймворків машинного навчання, зокрема TensorFlow, PyTorch, Scikit-learn і XGBoost. Архітектура також підтримує ONNX (Open Neural Network Exchange), забезпечуючи формат, який дозволяє передавати моделі між різними фреймворками машинного навчання.

Обчислювальні завдання (Jobs) в Azure Machine Learning Studio виконуються в середовищі, яке забезпечує ізоляцію виконання і відтворюваність. Кожне завдання тісно пов’язане з конкретними версіями наборів даних і моделей, що забезпечує високий ступінь відстежуваності. Завдання можуть бути параметризовані, що дозволяє систематично виконувати завдання з налаштування гіперпараметрів та оптимізації. Вони також можуть бути пакетними або виконуватися в режимі реального часу залежно від вимог конкретного завдання машинного навчання.

Процес створення навчального завдання в Azure Machine Learning Studio в рамках складається з декількох кроків. Спочатку Служба навчання встановлює з’єднання з робочим простором Azure Machine Learning, який слугує базовим середовищем для керування експериментами, моделями та обчислювальними ресурсами. Підключення здійснюється за допомогою класу Workspace.

Після успішного підключення до робочого простору навчальний сервіс переходить до створення експерименту - логічного контейнера для організації окремих прогонів навчального завдання. Для цього використовується клас Experiment. Далі навчальний сервіс налаштовує обчислювальні ресурси, які будуть використовуватися для виконання навчального завдання. Це передбачає визначення типу та конфігурації віртуальної машини, а також середовища Python. Класи Environment та ComputeTarget є важливими на цьому етапі.

Після налаштування обчислювального середовища та ресурсів навчальний сервіс інкапсулює навчальний сценарій та його залежності в об’єкт ScriptRunConfig. Цей об’єкт визначає скрипт Python, який потрібно виконати, обчислювальну ціль та середовище, серед інших параметрів. Навчальне завдання надсилається до робочої області Azure Machine Learning для виконання. Для запуску навчального завдання викликається метод submit класу Experiment. Алгоритм завдання наведено на рисунку 4.10.

Рисунок 4.10 – Алгоритм прунінгу і навчання

На етапі прунінгу, що відбувається безпосередньо перед навчанням нейронної мережі, відбувається прохід по навчальній вибірці і аккумуляція

градієнтів, що потім використовуються для розрахунку критерію важливості вагів і подальшого відсікання, за алгоритмами, описаними у розділах 2.4 та 3.4, використовуючи компонент, описаний в підрозділі 4.3.

Сервіс навчання відстежує хід виконання навчального завдання, фіксуючи метрики і контрольні точки та зберігаючи їх у визначеному сховищі моделей і базі даних метрик. Azure Machine Learning SDK надає функції для реєстрації метрик і завантаження контрольних точок, які легко інтегруються в навчальний сценарій (train_and_prune.py).

4.8 Кластер для тренування моделей нейронних мереж

Azure Machine Learning Studio надає платформу для створення навчальних кластерів з декількома графічними процесорами, пристосованих для глибокого навчання. Ця можливість допомагає задовольнити обчислювальні потреби алгоритмів глибокого навчання, які часто потребують значних паралельних обчислювальних потужностей для роботи зі складними архітектурами нейронних мереж і великими наборами даних.

Процес налаштування навчального кластера на базі GPU в Azure Machine Learning Studio складається з кількох ключових кроків. Спочатку користувач отримує доступ до робочої області Azure Machine Learning - централізованого центру, який полегшує керування різними компонентами машинного навчання після чого у робочому просторі користувач може створити новий обчислювальний кластер або налаштувати існуючий, який буде задовільняти конкретні вимоги завдань глибокого навчання.

Створення навчального кластера на базі графічних процесорів починається із зазначення типу та кількості необхідних віртуальних машин з графічними процесорами. Azure Machine Learning Studio пропонує ряд типів машин з графічними процесорами, кожен з яких оснащений різними конфігураціями

пам’яті, обчислювальною потужністю і типами графічних процесорів. Ці типи віртуальних машин призначені для виконання завдань глибокого навчання різного масштабу - від простих моделей до складних багатошарових нейронних мереж. Користувач обирає відповідний тип GPU на основі обчислювальних вимог своєї моделі глибокого навчання та розміру набору даних.

Після вибору типу машини користувач вказує кількість вузлів у кластері відповідно до масштабу навчальної задачі та бажаного рівня паралелізму. Більша кількість вузлів може значно скоротити час навчання за рахунок розподілу обчислювального навантаження, але це сприяє зростанню витрат (рисунок 4.11).

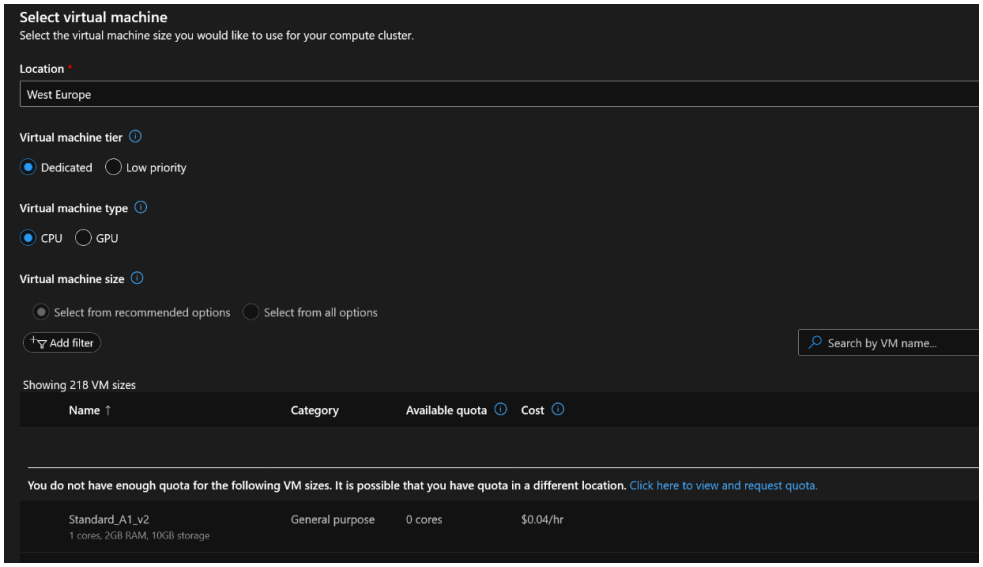

Рисунок 4.11 – Конфігурація віртуальних машин для розгортання кластеру

Процес конфігурації також включає налаштування політик масштабування для кластера. Ці поліси визначають, як кластер масштабується у відповідь на робоче навантаження. Azure Machine Learning Studio дозволяє здійснювати як ручне, так і автоматичне масштабування. Автоматичне масштабування особливо вигідне, оскільки воно динамічно регулює кількість активних вузлів у кластері

залежно від поточного робочого навантаження, оптимізуючи таким чином використання ресурсів і витрати (рисунок 4.12).

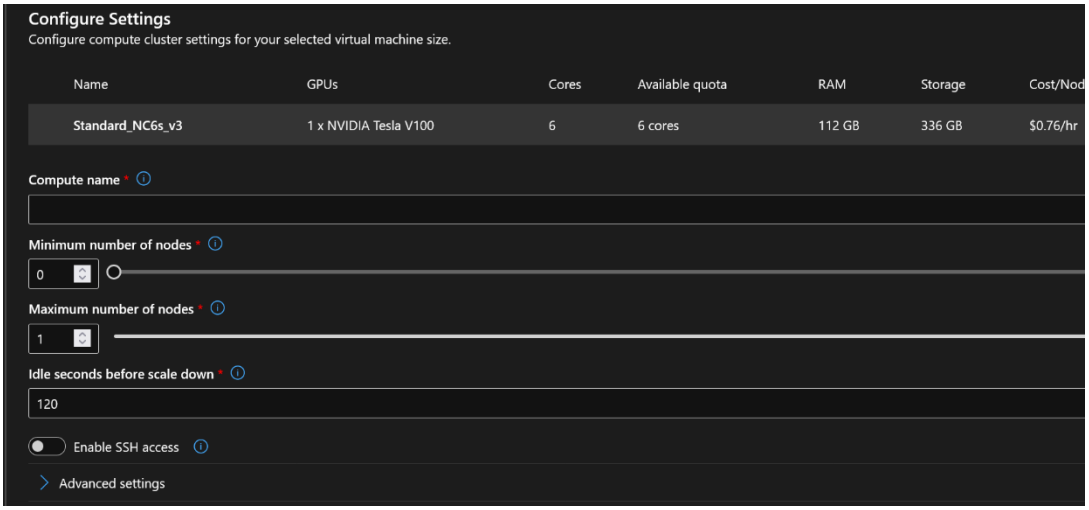

Рисунок 4.12 – Конфігурація політик масштабування для розгортання кластеру

Після завершення конфігурації кластера користувач розгортає кластер у робочому середовищі Azure Machine Learning. Процес розгортання включає в себе забезпечення зазначених машин GPU і налаштування необхідних мережевих конфігурацій і конфігурацій безпеки. Після цих дій кластер готовий до використання для навчання моделей глибокого навчання.

4.9 Збереження даних про експерименти

Для збереження даних про користувачів, автентифікації і авторизації використовується окрема база даних в СУБД PostgreSQL і платформа з відкритим вихідним кодом Supabase.

У запропонованій архітектурі розгортання бази даних PostgreSQL та екземпляра Supabase на окремих віртуальних машинах (VM) у хмарній інфраструктурі Azure окреслює підхід до управління даними користувачів та записами експериментів. Така структура використовує надійне зберігання даних

і можливості управління транзакціями PostgreSQL, одночасно використовуючи Supabase для синхронізації даних у реальному часі та автентифікації користувачів, кожна з яких працює в своєму оптимізованому операційному середовищі.

База даних PostgreSQL розміщена на віртуальній машині Azure, відібраній за високу пропускну здатність вводу/виводу та оптимізацію пам'яті. Встановлення та налаштування PostgreSQL на цій віртуальній машині передбачає налаштування параметрів пулу з'єднань та параметрів використання пам'яті, таких як shared_buffers та work_mem. Конфігурація WAL забезпечує цілісності даних і підвищення відмовостійкості. Для забезпечення мережевої безпеки використовуються групи мережевої безпеки Azure, що надають змогу обмежити доступ VM до діапазонів IP-адрес.

Для екземпляра Supabase створено окрему віртуальну машину Azure. Конфігурація цієї віртуальної машини адаптована до вимог операцій Supabase в режимі реального часу, з акцентом на продуктивність мережі та ефективність роботи процесора. Екземпляр Supabase на цій віртуальній машині встановлює з'єднання з базою даних PostgreSQL, розміщеною на іншій віртуальній машині. Це з'єднання дозволяє Supabase надавати підписки на дані в режимі реального часу та API, використовуючи при цьому надійні можливості зберігання даних PostgreSQL.

Служба автентифікації Supabase використовується для управління процесами автентифікації користувачів, включаючи реєстрацію користувачів, входи та управління токенами. У поєднанні з політиками безпеки PostgreSQL, Supabase забезпечує контроль доступу до даних.

Інтеграція цих систем проявляється в безперервному потоці даних, де взаємодія користувача з клієнтською програмою викликає запити на пошук даних або маніпуляції з ними до екземпляру Supabase. Supabase обробляє ці запити, при

необхідності взаємодіє з базою даних PostgreSQL і передає відповіді клієнтській програмі. Схему бази даних відображено на рисунку 4.13.

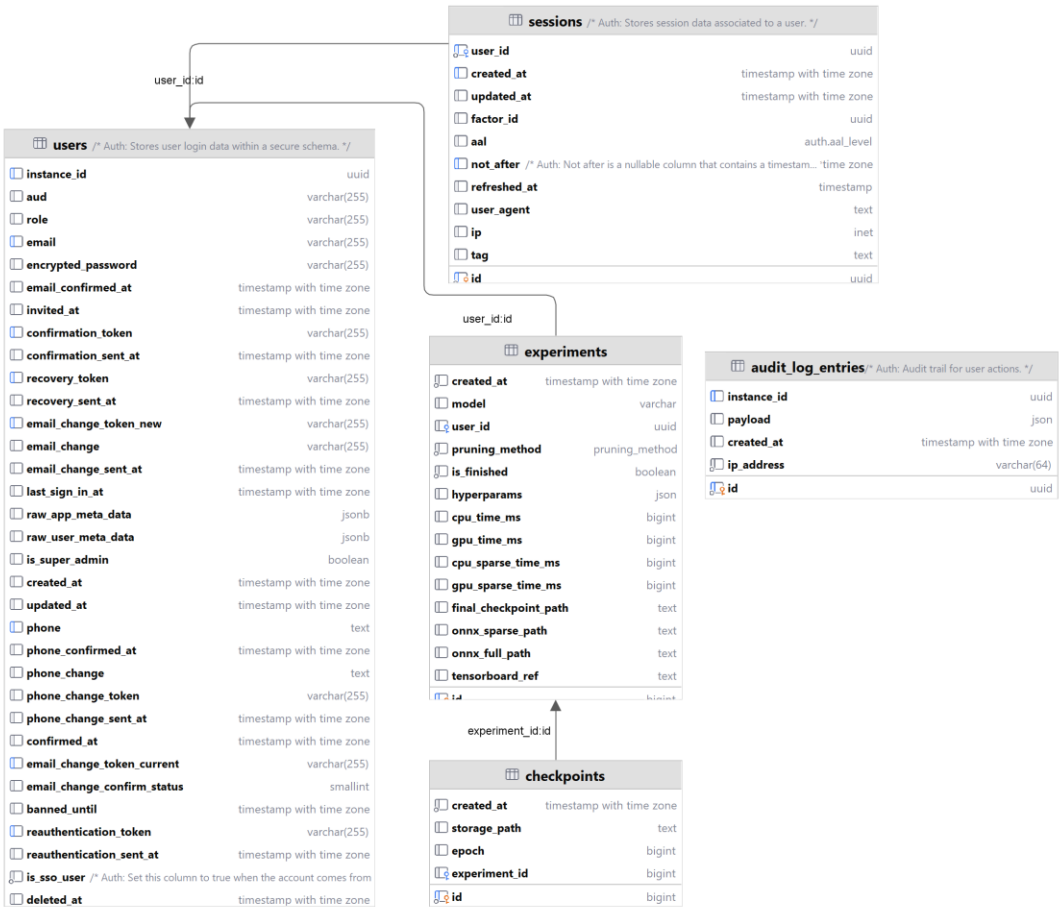

Рисунок 4.13 – Схема БД для збереження даних про користувачів і експерименти

Моніторинг та обслуговування обох віртуальних машин здійснюється за допомогою Azure Monitor та інструментів моніторингу баз даних Azure. Це включає в себе відстеження показників продуктивності, створення сповіщень про аномальні дії та регулярні перевірки стану, щоб забезпечити високу доступність та оптимальну продуктивність систем.

Висновки до розділу 4

Розроблена архітектура програмного забезпечення дозволяє інтегрувати технології хмарних обчислень платформи Azure для проведення експериментів з прунінгу нейронних мереж. Архітектурні рішення дозволяють додавати додаткові модулі до рішення, розширюючи наявний функціонал. Алгоритми прунінгу та нейронні мережі, імплементовані за допомогою таких інструментів, як бібліотека PyTorch, можуть бути розширені для подальших експериментів. Система має модуль автентифікації та базу даних експериментів, що дозволяє використовувати її в безпечному середовищі і мати доступ до даних для аналізу.

ВИСНОВКИ

У результаті дисертаційного дослідження вирішено актуальне наукове завдання розробки методу та програмних засобів для підвищення швидкодії глибоких нейронних мереж для вирішення задач розпізнавання образів. Дане наукове завдання має суттєве значення для теоретичних основ та програмно-алгоритмічного забезпечення в області глибоких нейронних мереж. Відсутність аналогічних рішень у нашій країні та за кордоном робить результати досліджень пріоритетними.

В дисертації одержані такі основні результати:

Аналіз існуючих методів оптимізації нейронних мереж показав нагальну потребу в зменшенні обчислювальних витрат для виконання і тренування нейронних мереж для розпізнавання образів.

Аналіз актуальних алгоритмів тренування і архітектур для вирішення задач розпізнавання образів показав широке використання глибоких нейронних мереж.

В процесі аналізу існуючих методів оптимізації було виявлено перспективні методи прунінгу нейронних мереж, що можуть бути застосовані вже перед початком навчання нейронної мережі

Удосконалено модель нейронної мережі для виявлення облич RetinaFace, яка на відміну від існуючих використовує метод прунінгу SNIP для оптимізації, що дозволяє використовувати розріджені матриці для зберігання і виконання мережі з метою подальшого удосконалення та збільшення швидкодії.

Удосконалено метод прунінгу SNIP для моделі виявлення облич RetinaFace, який на відміну від існуючих передбачає можливість виключення контекстних модулів з процесу прунінгу. Вдосконалений метод дозволяє досягти більшої точності при незмінній кількості виключених параметрів.

Вперше розроблено метод прунінгу перед навчанням для моделей архітектури трансформер, який на відміну від існуючих враховує важливість

механізму «уваги». Використання розробленого методу дозволяє значно збільшити точність класифікації кінцевої моделі в порівнянні з методом SNIP.

Вперше розроблено архітектуру програмного забезпечення для моделювання та дослідження методів прунінгу перед навчанням нейронних мереж, яка на відміну від існуючих дозволяє приводити матриці вагових коефіцієнтів мережі до розрідженого формату, використовуючи запропонований механізм оцінки важливості вагів.

Методика дослідження та отриманні результати можуть також бути використані для створення та оцінки архітектур нейронних мереж для інших доменів, тим самим розширюючи сферу потенційних застосувань. Дослідження може стати основою для оцінки ефективності інших методів оптимізації для моделей комп'ютерного зору та внести вклад у зростаючий обсяг наукової літератури з оптимізації нейронних мереж.

Результати досліджень прийняті до впровадження в Товаристві з обмеженою відповідальністю «ВОТЧЕД» (акт від 20.10.2023р.); в навчальному процесі Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (акт впровадження від 19.01.2024р.) при викладанні дисципліни «Технології розпізнавання образів та машинний зір»» для студентів освітньо-кваліфікаційного рівня «Магістр» спеціальності 122 «Комп'ютерні науки».

Мета досліджень, яка полягала в збільшення ефективності моделей нейронних мереж, а саме зменшення втрати точності при збільшенні швидкодії, після застосування методів оптимізації моделей глибинного навчання, створених для вирішення задач розпізнавання образів, досягнута.

Наукове завдання, що полягало в вдосконалення існуючих методів оптимізації і підвищення швидкодії нейронних мереж з урахуванням архітектурних особливостей мереж, виконано.

Наукові результати досліджень є внеском у розвиток теоретичних і прикладних основ розробки й дослідження науково-методичного і програмного апарату для оптимізації глибоких нейронних мереж з розпізнавання образів.

Наступними перспективними дослідженнями можуть стати дослідження для вдосконалення критеріїв визначення важливості вагів, ітеративне додавання параметрів під час навчання, дослідження інших форматів стиснених матриць та розширення експериментів на інші задачі та архітектури нейронних мереж.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сулема Є.С., Топчієв Б.С. Інтелектуальна колоризація зображень за допомогою генеративних змагальних мереж. *Системні технології*, 2019. Т.5. № 124. С. 94–103.

2. Dong S., Wang P., Abbas K. A survey on deep learning and its applications. *Computer Science Review*, May 2021, Vol. 40, Iss. C.

3. Semenov S., Weilin C., Zhang L., Bulba S. Automated penetration testing method using deep machine learning technology. *Advanced Information Systems*, 2021, Vol. 5, №3, pp. 119–127.

4. Strubell E., Ganesh A., McCallum A. Energy and Policy Considerations for Deep Learning in NLP. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, July 2019, Florence, Italy. pp. 3645–3650

5. Zikria Y.B., Afzal M.K., Kim S.W., Marin A., Guizani, M. Deep learning for intelligent IoT: Opportunities, challenges and solutions. *Computer Communications*, December 2020, Vol. 164, pp. 50-53.

6. Collin A., Siddiqi A., Imanishi Y., Rebentisch E., Tanimichi T., de Weck O. L. Autonomous driving systems hardware and software architecture exploration: optimizing latency and cost under safety constraints. *Systems Engineering*, December 2019, Vol. 23, Iss. 3, pp. 327-337

7. Amodei D. AI and compute. URL: https://openai.com/research/ai-and-compute (режим доступу 30.12.2023).

8. Van der Meulen, R. What Edge Computing Means for Infrastructure and Operations Leaders. URL: https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders (режим доступу 30.12.2023)

9. Li E., Wang B., Yang L., Peng Y., Du Y., Zhang Y., Chiu Y. GPU and CPU cooperative accelaration for face detection on modern processors. *IEEE International Conference on Multimedia and Expo*, 9 July 2012, pp. 769-775.

10. Wang, Y. C., Donyanavard, B., & Cheng, K. T. Energy-aware real-time face recognition system on mobile CPU-GPU platform. *In Trends and Topics in Computer Vision: ECCV 2010 Workshop*s, Heraklion, Greece, 10-11 September 2010, Vol. 11, pp. 411-422).

11. Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Karpathy A., Khosla A., Bernstein M., Berg A. C. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision,* December 2015, Vol. 115, pp. 211-252.

12. Сиротенко І. С., Щербина І. С., Сторчак К. П., Тушич А. М., Фокін В. І. Аналіз ефективності використання нейронних мереж на прикладі багатошарового персептрона та мережі Кохонена., *Зв'язок*, 2020, №5, С. 22-26.

13. Wu X., Sahoo D., & Hoi S. C. Recent advances in deep learning for object detection. Neurocomputing, July 2020, Vol. 396, pp. 39-64.

14. Сторчак К. П., Ткаленко О. М., Полоневич О. В., Тушич А. М., Усик М. Л. Застосування технології розпізнавання облич для запобігання інсайдерським атакам. *Зв'язок*, 2022, № 2, С. 32-39.

15. Cheriet M., Said J. N., Suen C. Y. A Recursive Thresholding Technique for Image Segmentation. *IEEE Transactions on Image Processing*, June 1998, Vol. 7, Iss. 6, pp 918-921.

16. Sappa A. D., Devy M. Fast Range Image Segmentation by an Edge Detection Strategy. In *Proceedings of the Third International Conference on 3-D Digital Imaging and Modeling*, 28 May - 01 June 2001, Quebec, Canada.

17. Fan J., Yau D. K. Y., Elmagarmid A. K., Aref W. G. Automatic image segmentation by integrating color-edge extraction and seeded region growing. IEEE Transactions on Image Processing, vol. 10, no. 10, Oct. 2001, pp. 1454-1466.

18. Long, J., Shelhamer E., Darrell T. Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* April 2017*,* Vol. 39, Iss. 4, pp. 640 - 651.

19. Ronneberger, O., Fischer, P., Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, November 2015, Vol. 9351, pp. 234-241.

20. He, K., Gkioxari, G., Dollar P., Girshick R. Mask R-CNN. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 22-29 October 2017, Venice, Italy, pp 2961-2969.

21. Zhao H., Shi J., Qi X., Wang X., Jia J. Pyramid Scene Parsing Network. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 21-26 July 2017, pp. 2881-2890.

22. Ruvinskaya, V.M. i Timkov, Y.Y.. Deep Learning Technology for Videoframe Processing in Face Segmentation on Mobile Devices. *Herald of Advanced Information Technology*. Publ. Nauka i Tekhnika. Odessa: Ukraine. 2021, Vol.4, No.2, pp. 185–194.

23. Pravitasari A. A., Iriawan N., Almuhayar M., Azmi T., Irhamah I., Fithriasari K., Purnami, S. W., Ferriastuti W. UNet-VGG16 with transfer learning for MRI-based brain tumor segmentation. *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*, June 2020, Vol. 18, Iss. 3, pp. 1310-1318.

24. Foody G. M., Mathur A. Toward intelligent training of supervised image classifications: directing training data acquisition for SVM classification. Remote Sensing of Environment, vol. 93, Issues 1–2, 2004, pp. 107-117.

25. Krizhevsky A, Sutskever I., Hinton G. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, May 2017, Vol. 60, Iss. 6, pp 84–90.

26. Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A. Going deeper with convolutions. *2015 IEEE Conference*

on Computer Vision and Pattern Recognition (CVPR), 07-12 June 2015, Boston, USA, pp. 1-9

27. Wang C., Chen D., Hao L., Liu X., Zeng Y., Chen J., Zhang G. Pulmonary image classification based on inception-v3 transfer learning model. IEEE Access, vol. 7, 2019, pp. 146533-146541.

28. Чопорова О. В., Кривохата А. Г. Оптимізація згорткових нейронних мереж та їх ансамблів. *Computer Science and Applied Mathematics*, № 1, 2020, с. 107-115.

29. Lienhart R., Maydt J. An extended set of Haar-like features for rapid object detection. *International Conference on Image Processing*, Rochester, USA, 2002.

30. Dalal N., Triggs B. Histograms of oriented gradients for human detection. *In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 20 June 2005, pp. 886-893.

31. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2016, pp. 779-788

32. Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C.Y., Berg A.C. SSD: Single shot multibox detector. *Proccedings of Computer Vision–ECCV 2016: 14th European Conference*, 11–14 October 2016, Amsterdam, Netherlands, pp. 21-37

33. Girshick R., Donahue J., Darrell T., Malik J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2014*, 23-28 June 2014, pp. 580-587.

34. Girshick R. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision 2015*, 07-13 December 2015, Santiago, Chile, pp. 1440-1448.

35. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39, June 2017, pp. 1137-1149.

36. Danelljan M., Hager G., Khan F. S., Felsberg M. Convolutional Features for Correlation Filter Based Visual Tracking. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 58-66.

37. Bertinetto L., Valmadre J., Henriques J. F., Vedaldi A., Torr P. H. S. Fully-Convolutional Siamese Networks for Object Tracking. *Computer Vision – ECCV 2016 Workshops*, 03 November 2016, pp. 850-865.

38. Wojke N., Bewley A., Paulus D. Simple Online and Realtime Tracking with a Deep Association Metric. 2017 *IEEE International Conference on Image Processing (ICIP)*, Sep 2017, pp. 3645-3649.

39. Gordon D., Farhadi A., Fox D. Re3: Real-Time Recurrent Regression Networks for Visual Tracking of Generic Objects. *IEEE Robotics and Automation Letters*, Vol. 3, Iss. 2, April 2018, pp. 788-795.

40. Lee J., Wang J., Crandall D., Šabanović S., Fox G. Real-time, cloud-based object detection for unmanned aerial vehicles. *In 2017 First IEEE International Conference on Robotic Computing (IRC),* 10 Apil. 2017, pp. 36-43.

41. Anjum A., Abdullah T., Tariq M. F., Baltaci Y., Antonopoulos N. Video stream analysis in clouds: An object detection and classification framework for high performance video analytics. *IEEE Transactions on Cloud Computing,* Vol. 7, №4, 13 Januray 2016, pp. 1152-1167.

42. Ren J., Guo Y., Zhang D., Liu Q., Zhang Y. Distributed and efficient object detection in edge computing: *Challenges and solutions. IEEE Network*, Vol. 32, №6, 2018, pp. 137-143.

43. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770-778.

44. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *3rd International Conference on Learning Representations (ICLR 2015)*. May 7-9 2015, San Diego, USA.

45. Brown T., Mann B., Ryder N., Subbiah M., Kaplan J. D., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A., Agarwal S. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems 33*, 2020, pp. 1877-1901.

46. Xu X., Ding Y., Hu S. X., Niemier M., Cong J., Hu Y., Shi Y. Scaling for Edge Inference of Deep Neural Networks. *Nature Electronics*, 2018, №4, pp. 216-222.

47. Zhang Y., Ren J., Liu J., Xu C., Guo H., Liu Y. A Survey on Emerging Computing Paradigms for Big Data. *Chinese Journal of Electronics*, 2017, Vol. 29, Iss. 1, pp. 1-12.

48. Kuchuk, N., Mozhaiev, O., Semenov, S., Haichenko, A., Kuchuk, H., Tiulieniev, S., Mozhaiev, M., Davydov, V., Brusakova, O., & Gnusov, Y. (2023). Devising a method for balancing the load on a territorially distributed foggy environment. *Eastern-European Journal of Enterprise Technologies*, Vol. 1, №4 (121), pp. 48–55.

49. Ke R., Zhuang Y., Pu Z., Wang Y. A smart, efficient, and reliable parking surveillance system with edge artificial intelligence on IoT devices. *IEEE Transactions on Intelligent Transportation Systems*, Vol. 22, №8, 8 April 2020, pp. 4962-4974.

50. Li B., Kong Z., Zhang T., Li J., Li Z., Liu H., Ding C. Efficient Transformer-based Large Scale Language Representations Using Hardware-friendly Block Structured Pruning. *Findings of the Association for Computational Linguistics*: *EMNLP 2020*, 2020, pp. 3187-3199.

51. Guo M., Yang Y., Xu R., Liu Z., Lin D. When NAS Meets Robustness: In Search of Robust Architectures Against Adversarial Attacks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2020, pp. 631-640.

52. Liang T., Glossner J., Wang L., Shi S., Zhang X. Pruning and Quantization for Deep Neural Network Acceleration: A Survey. Neurocomputing, 2021, Vol. 461, pp. 370-403.

53. Cheng H., Zhang M., Shi J. Q. A Survey on Deep Neural Network Pruning-Taxonomy, Comparison, Analysis, and Recommendations, 13 Aug 2023, arXiv preprint arXiv:2308.06767.

54. Phan A.-H., Sobolev K., Sozykin K., Ermilov D., Gusak J., Tichavský P., Glukhov V., Oseledets I., Cichocki A. Stable Low-Rank Tensor Decomposition for Compression of Convolutional Neural Network. *Computer Vision – ECCV 2020: 16th European Conference*, August 23–28, 2020, Glasgow, UK, pp. 522-539.

55. Astrid M., Lee S. I. Cp-decomposition with tensor power method for convolutional neural networks compression. *In 2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 13 February 2017, pp. 115-118.

56. Wang Y., Guo W. G., Yue X. Tensor decomposition to compress convolutional layers in deep learning. *IISE Transactions*, Vol. 54, №5, 4 May 2022, pp. 481-495.

57. Winata G. I., Madotto A., Shin J., Barezi E. J., Fung P. On the Effectiveness of Low-Rank Matrix Factorization for LSTM Model Compression. *Proceedings of the 33rd Pacific Asia Conference on Language, Information and Computation*, 2019, pp. 253-262.

58. Zhu C., Han S., Mao H., Dally W. J. Trained Ternary Quantization. *5th International Conference on Learning Representations*, April 24-26, 2017, Toulon, France.

59. Струнін І. В., Прогонов Д. О. Огляд сучасних методів адаптації нейронних мереж для малопотужних пристроїв. *XIX Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики»*, Україна, м. Київ, 13–14 травня 2021 р. С. 166-169.

60. Кушнір Д. О., Парамуд Я. С. Методи пошуку та розпізнавання об’єктів у відеозображеннях на мобільній платформі IOS в реальному часі. *Комп’ютерні системи та мережі*, 2019 р., том 1, № 1, С. 24-34.

61. Hinton G., Vinyals O., Dean J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 9 March 2015.

62. Howard A. G., Zhu M., Chen B., Kalenichenko D., Wang W., Weyand T., Andreetto M., Adam H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 17 April 2017.

63. Jiao X., Yin Y., Shang L., Jiang X., Chen X., Li L., Wang F., Liu Q. TinyBERT: Distilling BERT for Natural Language Understanding. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 4163-4174.

64. Sutaji D., Yildiz O., Gazi University. LEMOXINET: Lite Ensemble MobileNetV2 and Xception Models to Predict Plant Disease. *Ecological Informatics*, Vol. 70, September 2022.

65. Asif U., Tang J., Harrer S. Ensemble Knowledge Distillation for Learning Improved and Efficient Networks. *ECAI 2020*, 2020, Vol. 325, pp. 953-960.

66. Zoph B., Le Q. Neural Architecture Search with Reinforcement Learning. *5th International Conference on Learning Representations*, April 24-26, 2017, Toulon, France.

67. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Proceedings of the 36th International Conference on Machine Learning*, May 2019, pp. 6105-6114.

68. Tan M., Chen B., Pang R., Vasudevan V., Sandler M., Howard A., Le Q. V. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2820-2828.

69. Wang Z., Li C., Wang X. Convolutional Neural Network Pruning With Structural Redundancy Reduction. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 20-25 June 2021, Nashville, USA, pp. 14913-14922.

70. Howard A. G., Zhu M., Chen B., Kalenichenko D., Wang W., Weyand T., Andreetto M., Adam H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, *arXiv preprint, arXiv.1704.04861,* 17 April 2017.

71. Frankle J., Carbin M. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *7th International Conference on Learning Representations*, May 2019, New Orleans, USA.

72. Lee N., Ajanthan T., Torr P. H. S. SNIP: Single-shot Network Pruning Based on Connection Sensitivity. *7th International Conference on Learning Representations*, May 2019, New Orleans, USA.

73. Han S., Pool J., Tran J., Dally W. J. Learning Both Weights and Connections for Efficient Neural Networks. *Proceedings of the 28th International Conference on Neural Information Processing Systems*, December 2015, Vol. 1, pp. 1135-1143.

74. Li H., Kadav A., Durdanovic I., Samet H., Graf H. P. Pruning Filters for Efficient ConvNets. *5th International Conference on Learning Representations,* 24 – 26 April 2017, Toulon, France.

75. Wu B., Ai H., Huang C., Lao S. Fast rotation invariant multi-view face detection based on real adaboost. *In 6th IEEE International Conference on Automatic Face and Gesture Recognition*, 19 May 2004, pp. 79-84.

76. Carro R., Ahuactzin J.-M., Huerta E., Morales-Caporal R., Ramirez F. Face Recognition Using SURF. *Intelligent Computing Theories and Methodologies: 11th International Conference,* August 20-23, 2015, Fuzhou, China, pp. 316-326.

77. Zhang L., Chu R., Xiang S., Liao S., Li S. Z. Face Detection Based on Multi-Block LBP Representation. Advances in Biometrics: Proceedings of the International Conference on Biometrics, 2007, pp. 828–837.

78. Jiang H., Learned-Miller E. Face Detection with the Faster R-CNN. *12th IEEE International Conference on Automatic Face & Gesture Recognition*, May 30, 2017, pp. 650-657.

79. Chen Q., Shen F., Ding Y., Gong P., Tao Y., Wang J. Face Detection Using R-FCN Based Deformable Convolutional Networks. *2018 IEEE International Conference on Systems, Man, and Cybernetics*, Oct 2018, pp. 4165-4170.

80. Zhang K., Zhang Z., Li Z., Qiao Y. Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. *IEEE Signal Processing Letters*, October 2016, Vol. 23, №10, pp. 1499-1503.

81. Najibi M., Samangouei P., Chellappa R., Davis L. S. SSH: Single Stage Headless Face Detector. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 4875-4884.

82. Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C.-Y., Berg A. C. SSD: Single Shot MultiBox Detector. *Computer Vision – ECCV 2016*, 2016, pp. 21-37.

83. Lin T., Goyal P., Girshick R., He K., Dollár P. Focal Loss for Dense Object Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 01 February 2020, Vol. 42, Iss. 2, pp. 318 - 327.

84. Deng J., Guo J., Ververas E., Kotsia I., Zafeiriou S. RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 13-19 June 2020, Seattle, USA.

85. Yang, S., Luo, P., Loy, C. C., & Tang, X. (2016). WIDER FACE: A Face Detection Benchmark. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 27-30 June 2016, Las Vegas, USA.

86. Zitnick C. L., Dollár P. Edge Boxes: Locating Object Proposals from Edges. *European Conference on Computer Vision (ECCV 2014)*, 2014, pp. 391-405.

87. Glorot X., Bengio Y. Understanding the difficulty of training deep feedforward neural networks. *In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 31 Mar. 2010, pp. 249-256.

88. Melnychenko, A., Shaldenko, O. Evaluation of a snip pruning method for a state-of-the-art face detection model. Computational Problems of Electrical Engineering, 2023, Vol. 12, №1, pp. 22-27

89. Hoffer E., Ailon N. Deep Metric Learning Using Triplet Network. *Proceedings of International Workshop on Similarity-Based Pattern Recognition*, October 12-14, 2015, Copenhagen, Denmark, pp. 84-92.

90. Heidari M., Fouladi-Ghaleh K. Using Siamese Networks with Transfer Learning for Face Recognition on Small-Samples. *2020 International Conference on Machine Vision and Image Processing (MVIP)*, 18-20 February 2020

91. Melnychenko, A., Zdor, K. Efficiency of supplementary outputs in siamese neural networks. *Advanced Information Systems*, 2023, Volume 7, №3, pp. 49–53.

92. Liu Y., Lapata M. Text Summarization with Pretrained Encoders. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 3721.

93. Devlin J., Chang M. W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, June 2019, Vol. 1, pp. 4171-4186.

94. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention is All you Need. *Proceedings of the 31st International Conference on Neural Information Processing Systems,* December 2017, pp. 6000–6010.

95. Dosovitskiy A., Beyer L., Kolesnikov A., Weissenborn D., Zhai X., Unterthiner T., Dehghani M., Minderer M., Heigold G., Gelly S., Uszkoreit J., Houlsby N. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *International Conference on Learning Representations*, 04 May 2021, Vienna, Austria.

96. Zhu M., Gupta S. To prune, or not to prune: Exploring the efficacy of pruning for model compression. *6th International Conference on Learning Representations*, 30 April - 3 May 2018, Vancouver, Canada.

97. Wang S., Li B. Z., Khabsa M., Fang H., Ma H. Linformer: Self-Attention with Linear Complexity. *arXiv preprint arXiv:2006.04768,* 8 June 2020.

98. Melnychenko, A., Zdor K. Incorporating attention score to improve foresight pruning on transformer models. Computer Science and Applied Mathematics, 2023, №2, P.18-22

99. Wang Y. E., Wei G. Y., Brooks D. Benchmarking TPU, GPU, and CPU platforms for deep learning. arXiv preprint arXiv:1907.10701, 24 July 2019.

100. Vanhoucke V., Senior A., Mao M. Z. Improving the Speed of Neural Networks on CPUs. *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, January 2011.

101. Mishra A., Latorre J. A., Pool J., Stosic D., Stosic D., Venkatesh G., Yu C., Micikevicius P. Accelerating Sparse Deep Neural Networks. *arXiv preprint arXiv:2104.08378*, 16 Apr 2021.

102. Nguyen Q. Hands-on application development with PyCharm: Accelerate your Python applications using practical coding techniques in PyCharm, 2019. 494 p.

103. Obe R.O., Hsu L.S. PostgreSQL: up and running: a practical guide to the advanced open source database, 2017. 312 p.

104. McKinney W. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython, 2012. 547 p.

105. Johansson R. Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib, 2018. 723 p.

106. Raschka S., Liu Y., Mirjalili V. Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python, 2022. 774 p.

107. torch.nn - PyTorch 2.2 documentation. URL: https://pytorch.org/docs/2.2/nn.html (режим доступу 30.12.2023)

108. Kurtz M., Kopinsky J., Gelashvili R., Matveev A., Carr J., Goin M., Leiserson W., Moore S., Shavit N., Alistarh D. Inducing and exploiting activation sparsity for fast inference on deep neural networks. *Proceedings of the 37th International Conference on Machine Learning, PMLR 119*, 2020, pp. 5533-5543.

109. Kuzminykh V. O., Koval O. V., Svistunov S. Y., Xu Beibei, Zhu Shiwei. Data collection for analytical activities using adaptive micro-service architecture. *Data Recording, Storage & Processing*, 2021, Vol. 23, № 1.

110. Architecture & key concepts (v1) - Azure Machine Learning | Microsoft Learn. URL: https://learn.microsoft.com/en-us/AZURE/machine-learning/concept-azure-machine-learning-architecture (режим доступу 30.12.2023)

111. Мельниченко, А., Шалденко, О. Особливості використання прунінгу перед тренуванням нейронної мережі для детекції обличчя, XX Міжнародна науково-практична конференція молодих вчених і студентів, 25–28 квітня 2023 року, Київ, Україна

112. Melnychenko A. Evaluating SNIP pruning method on the state-of-the-art face detection model. Modern scientific research: achievements, innovations and development prospects, XVI Міжнародна науково-практична конференція, 11-13 вересня 2022 року, Берлін, Німеччина. С. 68-72.

113. Melnychenko, A., Zdor, K. Applying classification and regression supplemetary output in siamese neural network using fashion MNIST and plantvillage datasets, VII Міжнародна науково-практична конференція “Modern problems of science, education and society”, 11-13 вересня 2023 Київ, Україна, С. 126-129.

114. Melnychenko, A., & Zdor, K. Appling classification and regression supplemetary outputs in siamese neural network using plantvillage dataset, I Міжнародна науково-практична конференція “Current challenges of science and education”, 18-20 вересня 2023, Берлін, Німеччина. С. 79-82.

115. Melnychenko A., Zdor K. Appling classification and regression supplemetary output in siamese neural network using fashion MNIST and plantvillage datasets, X Міжнародна науково-практична конференція “Innovations and prospects in modern science”, 25-27 вересня 2023, Стокгольм, Швеція. С. 87-92.

116. Мельниченко А., Здор К. Збільшення ефективності оптимізації моделей архітектури ViT перед навчанням шляхом включення активацій механізму самоуваги, І міжнародна науково–практична конференція “Сучасні аспекти інженерії програмного забезпечення”, 14 грудня 2023, Київ, Україна.

117. Мельниченко А.В., Здор К.А. Врахування механізмів самоуваги при прунінгу моделей нейронних мереж Vision Transformer. Збірник матеріалів ІІІ Міжнародної науково-технічної конференції “Системи і технології зв’язку, інформатизації та кібербезпеки: актуальні питання і тенденції розвитку”, 30 листопада 2023 року, Київ, Україна. С. 214 – 215.

Додаток А

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Melnychenko, A., Shaldenko, O. Evaluation of a snip pruning method for a state-of-the-art face detection model. *Computational Problems of Electrical Engineering*, 2023, Vol. 12, №1, pp. 22-27
2. Melnychenko, A., Zdor K. Incorporating attention score to improve foresight pruning on transformer models. *Computer Science and Applied Mathematics*, 2023, №2, pp.18-22
3. Melnychenko, A., Zdor, K. Efficiency of supplementary outputs in siamese neural networks. *Advanced Information Systems*, 2023,Volume *7*, №3, pp. 49–53.

Наукові праці, які засвідчують апробацію матеріалів дисертації:

1. Мельниченко, А., Шалденко, О. Особливості використання прунінгу перед тренуванням нейронної мережі для детекції обличчя, XX Міжнародна науково-практична конференція молодих вчених і студентів, 25–28 квітня 2023 року, Київ, Україна
2. Melnychenko A. Evaluating SNIP pruning method on the state-of-the-art face detection model. Modern scientific research: achievements, innovations and development prospects, XVI Міжнародна науково-практична конференція, 11-13 вересня 2022 року, Берлін, Німеччина. С. 68-72.
3. Melnychenko, A., Zdor, K. Applying classification and regression supplemetary output in siamese neural network using fashion MNIST and plantvillage datasets, VII Міжнародна науково-практична конференція “Modern problems of science, education and society”, 11-13 вересня 2023 Київ, Україна, С. 126-129.

4. Melnychenko, A., & Zdor, K. Appling classification and regression supplemetary outputs in siamese neural network using plantvillage dataset, I Міжнародна науково-практична конференція “Current challenges of science and education”, 18-20 вересня 2023, Берлін, Німеччина. С. 79-82.

5. Melnychenko A., Zdor K. Appling classification and regression supplemetary output in siamese neural network using fashion MNIST and plantvillage datasets, X Міжнародна науково-практична конференція “Innovations and prospects in modern science”, 25-27 вересня 2023, Стокгольм, Швеція. С. 87-92.

6. Мельниченко А., Здор К. Збільшення ефективності оптимізації моделей архітектури ViT перед навчанням шляхом включення активацій механізму самоуваги, І міжнародна науково–практична конференція “Сучасні аспекти інженерії програмного забезпечення”, 14 грудня 2023, Київ, Україна.

7. Мельниченко А.В., Здор К.А. Врахування механізмів самоуваги при прунінгу моделей нейронних мереж Vision Transformer. Збірник матеріалів ІІІ Міжнародної науково-технічної конференції “Системи і технології зв’язку, інформатизації та кібербезпеки: актуальні питання і тенденції розвитку”, 30 листопада 2023 року, Київ, Україна. С. 214 – 215.

Додаток Б

УКРАЇНА

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

03056, м. Київ, пр-т Берестейський, 37; тел. (+38 044) 204-82-82 тел./факс (+38 044) 204-97-88
http://www.kpi.ua e-mail: mail@kpi.ua ЄДРПОУ 02070921

_____ № _____

"Затверджую"
проректор з навчальної роботи
КПІ ім.Ігоря Сікорського,
~~кандидат~~ філософський наук, професор
_____ Анатолій МЕЛЬНИЧЕНКО
"19" 01 2024 р.

АКТ
про використання в навчальному процесі наукових досліджень
Мельниченка Артема Васильовича - здобувача наукового ступеня доктора філософії

Результати дисертаційної роботи А.В. Мельниченко використовуються в навчальному процесі при викладанні таких навчальних дисциплін:
- "Технології розпізнавання образів та машинний зір" освітньої програми "Цифрові технології в енергетиці" за спеціальністю 122 Комп'ютерні науки за освітнім рівнем підготовки магістра (2022).

Директор навчально-наукового інституту
атомної та теплової енергетики,
доктор технічних наук, професор _____ Євген ПИСЬМЕННИЙ

Завідувач кафедри
цифрових технологій в енергетиці,
доктор технічних наук, професор _____ Наталія АУШЕВА

Заступник завідувача кафедри
цифрових технологій в енергетиці,
кандидат технічних наук, доцент _____ Юлія СИДОРЕНКО

ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ «ВОТЧЕД»
04119, м. Київ, вул. Хохлових Сім'ї, буд. 8, код ЄДРПОУ 43433547

№2010-1
Від 20.10.2023

АКТ

Впровадження результатів і висновків дисертаційної роботи

Мельниченка Артема Васильовича

«Методи та програмні засоби підвищення швидкодії моделей розпізнавання образів на основі машинного навчання»

Даний акт засвідчує, що результати та висновки дисертаційної роботи Мельниченка Артема Васильовича «Методи та програмні засоби підвищення швидкодії моделей розпізнавання образів на основі машинного навчання» використані для оптимізації моделей нейронних мереж з розпізнавання облич в рамках грантового договіру № 957321 на реалізацію проєкту START-UP DRIVEN INNOVATION IN EUROPEAN MEDIA (STADIEM) у межах Рамкової програми Європейської Комісії Горизонт 2020 «Дослідження реакції аудиторії на аудіовізуальний контент за допомогою нейронних мереж».

Керівник відділу досліджень
ТОВ «ВОТЧЕД»

Безсмертний В.Ю.

Додаток В

Лістинги коду класів для визначення коефіцієнтів важливості алгоритмів прунінгу

```
import os
import torch
import numpy as np


class PruningMethod:
    def __init__(self, masked_parameters, output_map_fn=None):
        self.output_map_fn = output_map_fn
        self.masked_parameters = list(masked_parameters)
        self.scores = {}

    def score(self, model, loss, dataloader, device):
        raise NotImplementedError

    def calc_masks(self, sparsity):
        all_scores = torch.cat([torch.flatten(v) for v in
self.scores.values()])
        elements_to_prune = int((1.0 - sparsity) *
all_scores.numel())
        if not elements_to_prune < 1:
            prune_threshold, _ = torch.kthvalue(all_scores,
elements_to_prune)
            for mask, param in self.masked_parameters:
                self.calc_prune_mask(prune_threshold, mask, param)

    def calc_prune_mask(self, prune_threshold, mask, param):
        score = self.scores[id(param)]
        zero = torch.tensor([0.0]).to(mask.device)
```

```
            one = torch.tensor([1.0]).to(mask.device)
            mask.copy_(torch.where(score <= prune_threshold, zero, one))

        def calc_stats(self):
            remaining_params, total_params = 0, 0
            for mask, _ in self.masked_parameters:
                remaining_params += mask.detach().cpu().numpy().sum()
                total_params += mask.numel()
            return remaining_params, total_params


    class SNIP(PruningMethod):
        def __init__(self, masked_parameters):
            super(SNIP, self).__init__(masked_parameters)

        def score(self, model, loss, dataloader, device):
            for m, _ in self.masked_parameters:
                m.requires_grad = True

            self._accumulate_gradients(model, loss, dataloader, device)
            self._fill_scores()
            self._normalize()

        def _accumulate_gradients(self, model, loss, dataloader, device):
            for batch_idx, (data, target) in enumerate(dataloader):

                if os.environ.get("TESTING") and batch_idx > 10:
                    break
                data,         target         =         data.to(device),
torch.tensor([t.to(device) for t in target])
                output = model(data)
                loss(output, target).backward()
```

```
        def _fill_scores(self):
            for mask, param in self.masked_parameters:
                self.scores[id(param)]                                    =
torch.clone(mask.grad).detach().abs_()
                param.grad.data.zero_()
                mask.grad.data.zero_()
                mask.requires_grad = False

        def _normalize(self):
            all_scores   =   torch.cat([torch.flatten(v)   for   v   in
self.scores.values()])
            norm = torch.sum(all_scores)
            for _, p in self.masked_parameters:
                self.scores[id(p)].div_(norm)



    class TransformerV1(PruningMethod):
        def    __init__(self,    masked_parameters,    att_activations,
attention_outs):
            self.att_activations = att_activations
            self.attention_outs = attention_outs
            super(TransformerV1, self).__init__(masked_parameters)

        def score(self, model, loss, dataloader, device):

            for m, _ in self.masked_parameters:
                m.requires_grad = True

            for batch_idx, (data, target) in enumerate(dataloader):
                data, target = data.to(device), target.to(device)
                output = model(data)
```

```
                loss(output, target).backward(retain_graph=True)
                for att in self.att_activations:
                    ones_vector                                          =
torch.ones_like(self.att_activations[att]["output"])
                    self.att_activations[att]["output"].backward(
                        gradient=ones_vector, retain_graph=True
                    )
                output.backward(gradient=torch.ones_like(output))

            for m, p in self.masked_parameters:
                self.scores[id(p)] = torch.clone(m.grad).detach().abs_()
                p.grad.data.zero_()
                m.grad.data.zero_()
                m.requires_grad = False

            all_scores   =   torch.cat([torch.flatten(v)   for   v   in
self.scores.values()])
            norm = torch.sum(all_scores)
            for _, p in self.masked_parameters:
                self.scores[id(p)].div_(norm)



    class TransformerV2(PruningMethod):
        def    __init__(self,    masked_parameters,    att_activations,
attention_outs):
            self.att_activations = att_activations
            self.attention_outs = attention_outs
            super(TransformerV2, self).__init__(masked_parameters)

        def score(self, model, loss, dataloader, device):

            for m, _ in self.masked_parameters:
```

```
                m.requires_grad = True

            for batch_idx, (data, target) in enumerate(dataloader):

                data, target = data.to(device), target.to(device)
                output = model(data)
                loss_m = loss(output, target)
                attentions = tuple(
                    torch.sum(self.att_activations[att]["output"])
                    for att in self.att_activations
                )
                attentions = torch.sum(torch.stack(attentions))
                out_sum = torch.sum(output)
                torch.sum(torch.stack((loss_m,                attentions,
out_sum))).backward()

            for m, p in self.masked_parameters:
                self.scores[id(p)] = torch.clone(m.grad).detach().abs_()
                p.grad.data.zero_()
                m.grad.data.zero_()
                m.requires_grad = False

            all_scores    =    torch.cat([torch.flatten(v)    for    v    in
self.scores.values()])
            norm = torch.sum(all_scores)
            for _, p in self.masked_parameters:
                self.scores[id(p)].div_(norm)


    class TransformerV3(PruningMethod):
        def    __init__(self,    masked_parameters,    att_activations,
attention_outs):
```

```
            self.att_activations = att_activations
            self.attention_outs = attention_outs
            super(TransformerV3, self).__init__(masked_parameters)

        def score(self, model, loss, dataloader, device):

            for m, _ in self.masked_parameters:
                m.requires_grad = True

            for batch_idx, (data, target) in enumerate(dataloader):

                data, target = data.to(device), target.to(device)
                output = model(data)
                loss_m = loss(output, target)
                attentions = tuple(
                    torch.sum(self.att_activations[att]["output"])
                    for att in self.att_activations
                )
                attentions = torch.sum(torch.stack(attentions))
                out_sum = torch.sum(output)
                torch.sum(torch.stack((loss_m,                attentions,
out_sum))).backward()

            for m, p in self.masked_parameters:
                self.scores[id(p)] = torch.clone(m.grad).detach().abs_()
                p.grad.data.zero_()
                m.grad.data.zero_()
                m.requires_grad = False

            all_scores    =    torch.cat([torch.flatten(v)    for    v    in
self.scores.values()])
            norm = torch.sum(all_scores)
```

```
            for _, p in self.masked_parameters:
                self.scores[id(p)].div_(norm)


    def load_pruner(pruner, masked_parameters, **kwargs) ->
PruningMethod:
        if pruner == "snip":
            return SNIP(masked_parameters)
        elif pruner == "transformer":
            return TransformerV3(masked_parameters)
        else:
            raise ValueError(f"Pruner {pruner} not supported")
```

Лістинги коду програмного компоненту для запуску експериментів з впливу прунінгу на мережу

```
from dataclasses import dataclass
from datetime import datetime
import numpy as np
import torch
from tqdm import tqdm
from prunelib.metrics import measure_model_flop, summary

from prunelib.save import save_model
from prunelib.trainer import Trainer
from prunelib.pruners import load_pruner
from prunelib import layers

BEFORE_PRUNING_DIR = "before_pruning"
```

```
    def masks(module):
        for name, buf in module.named_buffers():
            if "mask" in name:
                yield buf


    def is_trainable(module):
        return not isinstance(module, (layers.Identity1d,
layers.Identity2d))


    def is_prunable(module, batchnorm, residual):
        isprunable = isinstance(module, (layers.Linear, layers.Conv2d))
        if batchnorm:
            isprunable |= isinstance(module, (layers.BatchNorm1d,
layers.BatchNorm2d))
        if residual:
            isprunable |= isinstance(module, (layers.Identity1d,
layers.Identity2d))
        return isprunable


    def parameters(model):
        for module in filter(lambda p: is_trainable(p), model.modules()):
            for param in module.parameters(recurse=False):
                yield param


    def get_masked_parameters(model, bias=False, batchnorm=False,
residual=False):
        for module in filter(
```

```
            lambda    p:    is_prunable(p,    batchnorm,    residual),
model.modules()
        ):
            for      mask,      param      in      zip(masks(module),
module.parameters(recurse=False)):
                if param is not module.bias or bias is True:
                    yield mask, param


    @dataclass
    class PruneExperiment:
        pruner: str = "snip"
        device: str = "cuda:0"
        seed = 128
        save_dir = "training_results"
        expid = None
        input_shape = None
        verbose = True
        mask_scope = "global"
        compression = 0.5
        compression_schedule = "linear"
        prune_epochs = 1
        train_epochs = 100
        starting_epoch = 0
        prune_bias = False
        prune_batchnorm = False
        prune_residual = False
        lr_drops = [70, 90]
        lr_drop_rate = 0.1
        level = 1
        on_epoch_ended: callable = lambda *args: None
```

```
        @property
        def expid(self):
            return
f"{self.pruner}_{self.compression}_{datetime.now().strftime('%Y%m%d%H%M%S
')}"

        def run(
            self,
            prune_ds,
            train_ds,
            val_ds,
            test_ds,
            model,
            optimizer,
            loss,
            scheduler,
            input_shape,
        ):
            torch.manual_seed(self.seed)

            save_model(model,   optimizer,   scheduler,   self.save_dir,
BEFORE_PRUNING_DIR)

            pruner = load_pruner(
                self.pruner,
                get_masked_parameters(
                    model,    self.prune_bias,    self.prune_batchnorm,
self.prune_residual
                ),
            )
```

```
            sparsity = (10 ** (-float(self.compression))) ** ((self.level
+ 1) / self.level)

            self.prune_loop(
                model,
                loss,
                pruner,
                prune_ds,
                self.device,
                sparsity,
                self.compression_schedule,
                self.mask_scope,
                self.prune_epochs,
            )

            optimizer.load_state_dict(
                torch.load(
                    "{}/{}/optimizer.pt".format(self.save_dir,
BEFORE_PRUNING_DIR),
                    map_location=self.device,
                )
            )
            scheduler.load_state_dict(
                torch.load(
                    "{}/{}/scheduler.pt".format(self.save_dir,
BEFORE_PRUNING_DIR),
                    map_location=self.device,
                )
            )

            print("Pruning done")
            prune_result = summary(
```

```
                model,
                pruner.scores,
                measure_model_flop(model, input_shape, self.device),
                lambda    p:    is_prunable(p,    self.prune_batchnorm,
self.prune_residual),
            )

            print("Starting training")

            trainer = Trainer(
                model,
                loss,
                optimizer,
                scheduler,
                train_ds,
                val_ds,
                test_ds,
                self.save_dir,
                f"/tb/{self.expid}",
                starting_epoch=self.starting_epoch,
                on_epoch_ended=self.on_epoch_ended,
            )
            trainer.train(self.device, self.train_epochs, self.verbose)
            self._calc_prune_results(prune_result)

        def _calc_prune_results(self, prune_result):
            total_params      =      int((prune_result["sparsity"]      *
prune_result["size"]).sum())
            possible_params = prune_result["size"].sum()
            total_flops      =      int((prune_result["sparsity"]      *
prune_result["flops"]).sum())
            possible_flops = prune_result["flops"].sum()
```

```
            print("Prune results:\n", prune_result)
            print(
                "Parameter Sparsity: {}/{} ({:.4f})".format(
                    total_params,   possible_params,   total_params   /
possible_params
                )
            )
            print(
                "FLOP Sparsity: {}/{} ({:.4f})".format(
                    total_flops,    possible_flops,    total_flops    /
possible_flops
                )
            )

        def prune_loop(
            self, model, loss, pruner, dataloader, device, sparsity,
schedule, scope, epochs
        ):

            model.eval()
            remaining_params, total_params = pruner.calc_stats()
            print(
                f"Before pruning. Total params: {total_params}, Remaining
params: {remaining_params}"
            )

            for epoch in tqdm(range(epochs)):
                pruner.score(model, loss, dataloader, device)
                if schedule == "exponential":
                    sparse = sparsity ** ((epoch + 1) / epochs)
                elif schedule == "linear":
```

```
                    sparse = 1.0 - (1.0 - sparsity) * ((epoch + 1) /
epochs)

                pruner.calc_masks(sparse)

            self.confirm_sparsity(pruner, sparsity)

        def confirm_sparsity(self, pruner, sparsity):
            remaining_params, total_params = pruner.calc_stats()
            print(total_params)
            print(remaining_params)
            print(total_params - remaining_params)
            if np.abs(remaining_params - total_params * sparsity) >= 5:
                raise ValueError(
                    f"ERROR:  {remaining_params}  prunable  parameters
remaining, expected {total_params*sparsity}"
                )



    def reload_model(self, device, model, optimizer, scheduler, subdir):
        model.load_state_dict(
            torch.load("{}/{}/model.pt".format(self.save_dir,  subdir),
map_location=device)
        )
        optimizer.load_state_dict(
            torch.load(
                "{}/{}/optimizer.pt".format(self.save_dir,     subdir),
map_location=device
            )
        )
        scheduler.load_state_dict(
```

```
            torch.load(
                "{}/{}/scheduler.pt".format(self.save_dir,      subdir),
map_location=device
            )
        )
```