КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

Розпізнавання клинописних знаків за допомогою графової мережі згортки

Дипломна робота магістра
студентки II курсу ОР «Магістр»
спеціальності 123 «Комп'ютерна інженерія»
ОП «Комп'ютерні системи та мережі»

Карини ХАТХОХУ

_____

Науковий керівник:
кандидат фізико-математичних наук,
асистент **Андрій КОНОВАЛОВ**

_____

Рецензент:
кандидат фізико-математичних наук,
доцент кафедри нанофізики конденсованих середовищ, Інститут високих технологій,
Іван ІВАНОВ

_____

До захисту допускаю: Завідувач кафедрою, канд. фіз.-мат. наук, доцент **Юрій БОЙКО**

Ухвалено на засіданні кафедри "____" _____ 2022 р., протокол № ____

Київ - 2022

Реферат

Робота присвячена побудові класифікаторів клинописних символів набору даних Cuneiform на основі графових нейронних мереж різних типів архітектур (ChebConv, GMM, GeneralConv, SplineCNN).

Робота складається з трьох частин. Перша частина містить мотивацію до проведення роботи та теоретичні відомості щодо графових мереж згортки. Друга частина описує інструменти та методи проведення досліджень. Третя частина роботи містить результати проведених досліджень, їх аналіз та знайдену оптимальну модель графової мережі з типом шарів SplineCNN з точністю 93.7%, що більш ніж на 6% перевищує точність представленої у літературі моделі класифікатора, що визначалась за аналогічною методикою.

Робота містить 57 сторінок, 18 рисунків, 3 таблиці та 3 додатки.

Ключові слова: ГРАФОВА НЕЙРОННА МЕРЕЖА ЗГОРТКИ, КЛИНОПИС, SPLINECNN, CUNEIFORM.

Зміст

Вступ

Клинопис є найстарішою відомою системою письма у світі, що зберігає безцінні записи ранньої історії людства у всіх сферах життя. Він використовувався на більшій частині Стародавнього Близького Сходу протягом періоду понад три тисячі років до поточної ери [1].

Письмо клинописом полягає у відбитті на шматку вологої глини клиноподібних узорів, за допомогою стилуса гострої, прямокутної форми. Результатом є глиняна табличка, текст на якій — тривимірний, і стає читабельним лише при наявності джерела світла, що утворює чіткі тіні [2].

Шумерська мова — ізолят [3], та не має прямих нащадків. Окрім власне шумерського корпусу текстів світу залишилися тексти народів, що перейняли їхню систему письма.

Існує більш як 100 000 знайдених глиняних табличок, багато з яких ще не були оброблені експертами [2]. Така велика кількість обумовлено особливостями глиняних табличок як засобу зберігання інформації — глина не була дефіцитним матеріалом, на ній робились як побутові замітки, так і державна бюрократія, і літературний матеріал. Після чого, через клімат, таблички висихали на сонці, самі перетворюючись на археологічний артефакт для майбутніх поколінь. Кількість збереженого матеріалу така, якби сучасна людина ламінувала всі, навіть найменш значні свої паперові замітки.

Ручна обробка клинописних текстів це трудомісткий процес, що не тільки вимагає багато часу та зусиль, а й потребує від спеціаліста років професійного навчання [4—7]. Тому автоматичне розпізнавання цих текстів — актуальна задача археології та машинного навчання [2—8]. Форма штриха, обумовлена інструментом письма — стилусом, — в літературі традиційно називається клин (wedge), та по суті являє собою тетраедр [9], [10]. Автоматизована обробка

цифрових клинописних даних може проводитися з використанням різних способів представлення, від фотографічних репродукцій та сканованих 3D моделей до 2D розгортки тривимірного об'єкта [8].

Символи клинопису можна також представити в якості графів. Тоді розпізнавання даних можна виконувати за допомогою машинного навчання графових нейронних мереж.

В рамках роботи було досліджено класифікацію клинописних символів за допомогою графових мереж згортки, та представлено результати дослідження класифікації в залежності від архітектури та гіперпараметрів цих мереж.

1 Огляд літератури

1.1 Цифрова обробка археологічних артефактів

Протягом років були розроблені та випробувані різні способи цифрового представлення клинописних табличок, та інших археологічних артефактів. Одним з перших способів зберігання даних про ці об'єкти стали бази даних, в яких об'єкт зберігається як набір фотографій та текстової інформації. У випадку табличок цією текстовою інформацією зазвичай є: транслітерація (точна передача клинописних знаків спеціальним латинським алфавітом), переклад, та іноді оригінальний текст символами Unicode, які разом з іншими даними про артефакт складають так звані метадані. Однією з найбільших з таких баз даних є CDLI (Cuneiform Digital Library Initiative) [11], спільний проект Каліфорнійського університету в Лос-Анджелесі, Оксфордського університету та Інституту історії науки Макса Планка в Берліні.

Як було зазначено вище, створення таких цифрових записів про об'єкт потребує багато часу та роботи висококваліфікованого співробітника. Відповідно, постає задача автоматизації їх обробки, що може охоплювати, в залежності від вибору дослідника [12]:

- автоматичне зацифровування фізичної інформації про об'єкт;
- автоматичне відновлення артефактів;
- автоматична транслітерація та транскрипція;
- лінгвістична анотація;
- комп'ютерна обробка змісту.

1.1.1 Автоматичне зацифровування фізичної інформації про об'єкт

На сьогодні сюди відноситься, наприклад, створення тривимірного скану об'єкта, на основі якого можна згенерувати, (іноді полегшену), інформацію для

обробки: як двовимірні зображення об'єкту зі стандартних ракурсів, теплові карти глибини тощо.

При створенні набору даних HeiCuBeDa (Heidelberg Cuneiform Benchmark Dataset), було поєднано тривимірні скани табличок (Hilprecht Sammlung, HS [13]) з метаданими (CDLI каталог [11]) [14]. Тривимірні моделі HS не мали асоційованих метаданих, та не були послідовно орієнтовані у просторі. Створення структурованого набору даних потребувало ручного представлення моделей та метаданих у відповідність одне одному, та чищення самих файлів моделей. Автори роботи створили її у 2019 році з метою надати дослідникам можливості працювати з великим, нормалізованим набором даних.

На рис. 1.1 зображено деякі методи обробки тривимірної моделі, з яких можна надалі виділити різні ознаки, при чому не лише й розпізнавати символи, а й, наприклад виявляти період виготовлення артефактів.

Щодо розробок, які намагаються саме автоматизувати створення набору даних, можна зазначити [15], що використовує ресурси GPGPU та паралельні обчислення для виділення ознак, аналізу та візуалізації клинописного тексту на відсканованих 3D моделях фізичних табличок. За їх же словами, «проекти зацифровування стародавніх артефактів створюють величезну кількість даних, яку неможливо проаналізувати за адекватний проміжок часу за допомогою звичних філологічних методів.» Таким чином, [15] не просто зберігає час спеціалістів, а є логічним кроком у розвитку археологічної галузі.

«Автоматизоване недороге фотограмметричне отримання 3D-моделей з артефактів малого формфактора [16]» — проект 2019 року, що розробив систему для сканування фрагментів табличок, яку можна зібрати на місці всього за 150$, що ефективно використовує програвачі, цифрові камери та смартфони.

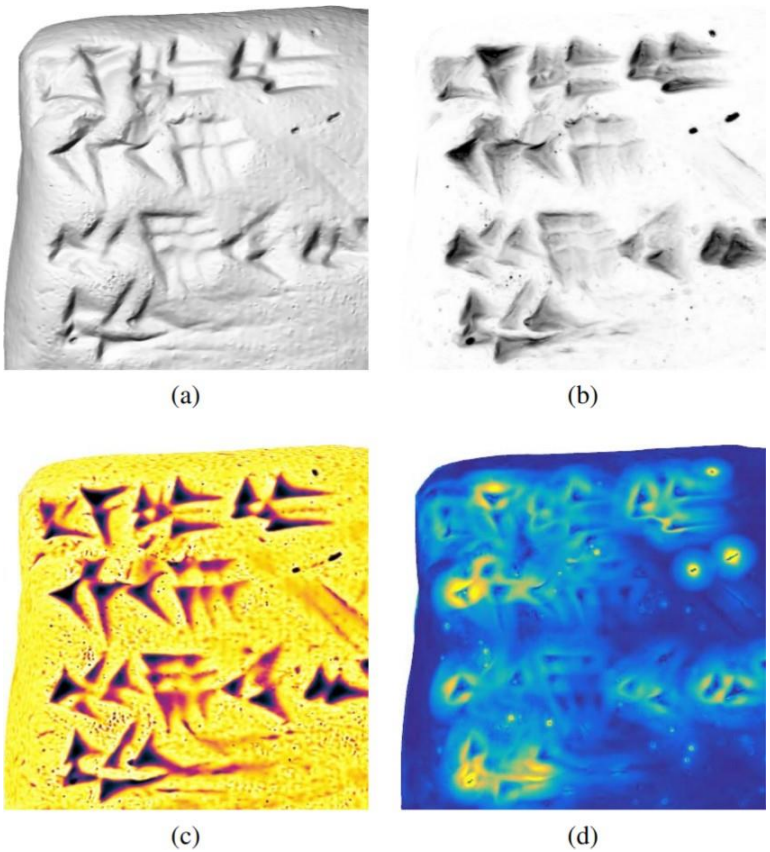

Рис. 1.1. Іллюстрація з [14], приклад генерації двовимірних зображень різними способами. (a) генеація з освітленням, (b) AmbientOcclusion, (c) виділення найбільш об'ємного елементу(d) інтегральний інваріант по поверхні.

1.1.2 Автоматичне відновлення артефактів

Часто археологічні артефакти на момент заходження є пошкодженими. Автоматична реконструкція артефакту полягає у знаходженні найбільш вірогідних з’єднань для цифрового представлення фрагментів об’єкта за допомогою обчислювальних методів. Це робить задачу чимось схожою на розв'язання пазлу, але з додатковими складнощами: наявні частини зазвичай пошкоджені, а кількість частин результату невідома [12].

Окрім відновлення власне фізичних об'єктів, досліджується можливість відновлення власне транслітерованого змісту табличок, що робить задачу відновлення лінгвістичною. Розглядається відновлення за допомогою

класифікатору максимально ентропії [17]; з алгоритмом вирівнювання послідовності [18]; та відносно новий підхід заповнення прогалин на основі прикладів, знайдених в інших подібних текстах, представлений у [19].

1.1.3 Автоматична транслітерація та транскрипція

Досліджуються багато різних підходів до розпізнавання клинописного тексту, серед яких обробка фотографій (освітлених так, щоб показати глибину [4], або оброблених так, щоб спрости зображення до простих контрастних форм [20]), або двовимірних проєкцій згенерованих з тривимірних моделей [14], обробку сіток тривимірних моделей напряму, та, в нові часи, обробку клинописного тексту, представленого графами [8].

Незалежно від способу представлення даних, для розпізнавання використовуються різного роду системи штучного інтелекту, від простих класифікаторів, що розпізнають символи на основі ретельно підготовлених даних, до складніших систем, які мають спочатку локалізувати символ на зображенні, як у [4].

1.1.4 Лінгвістична обробка змісту

Ця частина процесу зацифровування належить до розділу комп'ютерної лінгвістики, і не представляє інтересу в цій роботі. Клинописна писемність використовувались багатьма народами, починаючи з шумерів, хеттів, асирійців, тощо протягом довгого часу. Її використання відрізнялося навіть тим, чи була азбука силабічна чи лого-силабічна. Таким чином, задача машинного перекладу різних мов, що використовують клинопис, має вирішуватися окремо для кожної з мов.

Наразі, дослідження зосереджені на розпізнаванні клинопису загалом, враховуючи мову в першу чергу для класифікації символів.

В результаті об'єднання та технічного вдосконалення цих 3-х кроків, вони могли б утворити єдиний автоматизований конвеєр перетворення археологічного об'єкта (таблички) на перекладений текст. Робота [4], наприклад, демонструє саме філософію уніфікованого конвеєра від зображення до транслітерації. Але наразі дослідження знаходяться на надто ранньому етапі для серйозної розробки холістичної системи.

Штучний інтелект і машинну обробку даних про археологічні артефакти можна застосовувати й в інший спосіб. В роботі [1], наприклад, машинне навчання на 3-вимірних моделях табличок використовуються для їх автоматичної періодизації. Для класифікації використовується геометрична нейронна мережа (див. 1.3), в яку на вхід надходить трикутна сітка точок (3D модель), а не абстрагування, ручне чи автоматичне, ознак табличок.

1.2 Представлення клинопису в форматі графів

Нещодавно стали з'являтися роботи, що використовують для розпізнавання клинописних текстів графові нейронні мережі. Цей метод можна ефективно застосувати до клинопису, через особливості його форми. Штрихом клинописного символу є заглиблення в глині у формі тетраедру. Якщо з іншим об'єктом, наприклад, з єгипетськими ієрогліфами, вирізьбленими в поверхні, довелося б обробляти складну тривимірну сітку, то представлення клинопису можна спростити до цих елементарних тетраедрів.

1.2.1 Набір даних Cuneiform

В цій роботі використовувався набір даних, створений у [8]. Для його створення були використані таблички, створені науковцями-хеттологами. Це дозволило забезпечити збалансованість набору — однакова кількість зразків (9) для кожного символу, написані різними почерками [21]. Розмір набору — 267 зразків. Кожен зразок представлений у вигляді графу.

В наборі 30 класів, тобто 30 символів хеттської складової азбуки, а кожен з

символів представлений приблизно 9-ма зразками. Як видно на рисунку 1.2, кожен символ складається з декількох клинів — відмітин, що являють собою заглиблення, зроблені стилусом у глиняній дошці, коли глина була ще вологою.

Символи для набору даних були обрані таким чином, щоб включати й сильно відмінні одне від одного сузір'я клинів, так і візуально подібні. Як наслідок, набір даних має забезпечити як генералізацію, так і дискримінацію при навчанні на ньому мережі.

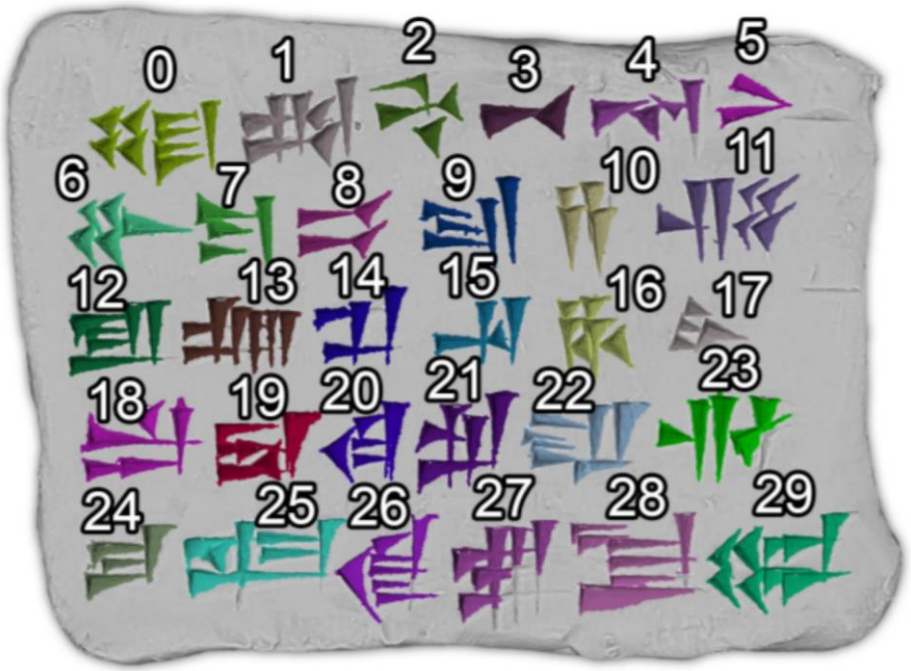

Рис. 1.2. Приклад таблички з 30 символами хеттського алфавіту [8].

Існує декілька типів відмітин-клинів в залежності від кута, під яким знаходився стилус. Символ азбуки визначається взаємним розташуванням клинів та їх типами.

Хвостова вершина завжди є слідом від довгої частини, тобто руків'я стилуса. Ліва та права вершини відмічені та називаються так відносно хвостової. Вершиною глибини називається вершина, заглиблена у поверхню таблички (рис. 1.3).

При формуванні набору даних Cuneiform було додано ребра, проведені між вершинами глибини одного символу (рис. 1.3), що не існують фізично, і названі ***ребрами аранжування***. Це зроблено, щоб мережа навчалася на інформації про відносне розташування цих вершин, адже деякі символи дуже схожі між собою без цієї інформації, наприклад символи 0 та 29. Різниця між позиціями окремих клинів недосконало відображається їх розташуванням у глобальній системі координат. Хоча ця різниця легко розпізнається людиною, один знак можна перетворити на інший без помітного зміщення у просторі. Ребра аранжування були додані для розв'язання цієї проблеми.

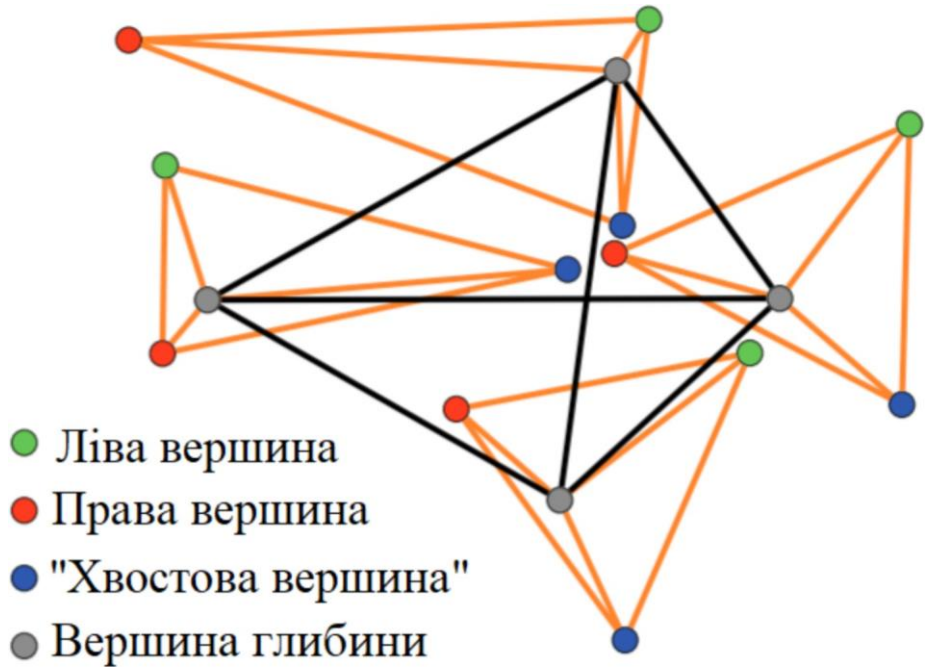

Рис. 1.3. Символ номер 2 з рис. 1 з відміченими типами вершин [8].

1.3 Графові нейронні мережі

1.3.1 Геометричне глибоке навчання

Ідея геометричного глибокого навчання (geometric deep learning) не нова, і почала розвиватися ще в двадцятому сторіччі. В загальному сенсі вона полягає у машинному навчанні на геометричних структурах. [22] виділяє 5 типів структур, з якими працюють в геометричному навчанні:

- сітки та евклідові простори;
- групи та однорідні простори;
- **графи та множини**;
- геодезії та різноманіття;
- калібровки та векторні розшарування.

Робота [22] також визначає геометричні графи як окремий підтип графів. **Геометричні графи** визначаються як графи, що можуть бути представлені в Евклідовому просторі. Слід зазначити, що сама ідея генералізації геометричного навчання полягає у створенні можливості працювати з неевклідовими структурами [23]. Геометричні графи ж можуть служити відправною точкою для розуміння переходу від широко відомих нейронних мереж згортки, в яких вхідними даними є зображення, до більш загальних **геометричних нейронних мереж.**

Геометрична нейронна мережа — нейронна мережа, що приймає в якості вхідних даних загальну геометричну структуру.

1.3.2 Графові нейронні мережі (GNN)

Графом називається множина вузлів та множина ребер між парами вузлів. У вузлів є деякі ознаки, кількість ознак називається їх розмірністю. **Сусідством вузла** називається множина N_u така, що $N_u=\{v : (u, v) \in E\}$, де E множина ребер, (u, v) — пари вузлів, що належать множині ребер. Сусідство також називають вузлами на відстані одного стрибка (**1-hop**) [22], або сусідствами 1 стрибка.

В роботі [24] у 1997 році вперше застосували нейронні мережі до орієнтованих ациклічних графів, що поклало початок раннім дослідженням GNN. Двома основними типами графових нейронних мереж є **рекурентні GNN** (RecGNNs), та **графові мережі згортки** (ConvGNN, вони ж **GCN**).

Ітеративна природа рекурентних GNN робить їх ресурсозатратними з точки зору обчислень. Графові мережі згортки є як кроком до розв'язання цієї проблеми, так і одночасно результатом розширення поняття згорткової мережі на нерегулярні геометричні структури [23]. Перші GCN були розроблені на основі спектральної теорії графів, і тому називаються спекратльними (spectral-based) [23, 25]; так звані просторові (spatial-based) GCN [26] з'явились раніше за спектральні, але залишалися на периферії досліджень до недавнього часу [23].

1.3.3 Мережа SplineCNN

MoNet — мережа змішаної моделі (Mixture Model Network) — це загальний фреймворк, що дозволяє проектувати згорткові мережі глибокого навчання у неевклідових просторах, таких як графи та різноманіття [27].

MoNet використовує псевдокоординати вузлів для визначення відносного «положення» між вузлами та їх сусідами. Коли відомим стає відносне положення вузлів, функція ваг ставить у відповідність ці позиції відносним вагам між цими вузлами. Таким чином, параметри фільтра графу використовуються в різних місцях мережі.

Мережа **SplineCNN** побудована на фреймворку **MoNet** [23]. В процесі досліджень, особливо в останні декілька років, було створено безліч графових нейронних мереж для найрізноманітніших цілей [23], і в таблиці 1.1 представлено місце SplineCNN в їх типології.

Таблиця 1.1. Систематика нейронної мережі SplineCNN як підвиду графових нейронних мереж.

GNN				
GraphCNN				...
Spatial-based ConvGNN			...	
MoNet		...		
SplineCNN	...			

Мережа SplineCNN визначається як клас глибоких нейронних мереж, що побудована із шарів згортки заснованих на методі сплайна. Такий шар отримує нерегулярно структуровані дані на вході, та ставить їх у відповідність напрямленому графу. В цьому просторовому шарі згортки, ознаки вузлів та ребер агрегуються з використанням неперервної функції ядра з тренованими параметрами [28].

Функція ядра згортки позначається $g_l : [a_1, b_1] \times ... \times [a_d, b_d] \to \mathbb{R}$, де d — розмірність псевдокоординат, тобто кількість ознак ребер. Оператор згортки для функції ядра g_l та вхідних ознак f вузлу i визначено так [28]:

$$(f * g)(i) = \frac{1}{|N(v_i)|} \sum_{l=1}^{M_{in}} \sum_{j \in N(i)} f_l(j) \cdot g_l(u(i,j))$$

де $N(v_i)$ — сусідство вузла i, M_{in} — кількість вхідних ознак вузла.

SplineCNN надає швидкий алгоритм, що робить припущення про клас графу [8]. Це робить його гарним кандидатом для використання в розпізнаванні клинописних символів, представлених у вигляді графів. Іншою мотивацією до використання цієї мережі — її можливість швидко навчатися на відносному невеликому наборі даних [8].

В роботі [8] досліджено можливість графової мережі згортки на основі SplineCNN розпізнавати індвивідуальні символи клипопису на наборі даних Cuneiform. Точність було отримано за допомогою 10-кратної перехресної перевірки з випадковим розділенням на частини. Після повторення експерименту перехресної перевірки десять разів отримано середнє значення точності **87,8**% без *аугментації* (штучного розширення даних за рахунок додвавання випадкових спотровень вхідних зразків) та **93,54%** з аугментацією.

Метою дипломної роботи магістра є підбір архітектури графової мережі і оптимізація її гіперпараметрів для максимізації точності розпізнавання клинописних символів з набору даних Cuneiform.

Для досягнення поставленої мети виконувались такі задачі:

1. Побудова моделей класифікаторів клинописних символів Cuneiform на основі графових мереж різних типів архітектур (ChebConv, GMM, GeneralConv, SplineCNN) з використанням ранньої зупинки навчання.
2. Оптимізація гіперпараметрів побудованих моделей на основі методу перехресної перевірки.
3. Аналіз отриманих результатів та їх порівняння з літературними даними.

2 Методологія дослідження

2.1 Інструментарій

2.1.1 Середовище Google Coalbaratory

Розробка проекту відбувалась в середовищі Goole Colabaratory, що дозволяє виконувати Python код прямо у вікні браузера.

Код зберігається в так званих блокнотах — .ipyb файлах.

Виконується код на віртуальній машині Linux, на яку автоматично встановлено Python та деякі бібліотеки. Google виділяє необхідні ресурси для нового віртуального середовища, та надає можливість безкоштовно використовувати графічні або тензорні прискорювачі, але обмежений проміжок часу.

Графічні карти, що надає Google відрізняються, в залежності від типу послуги, яку надає Google Colaboratory та можуть змінюватися у випадку змін на серверах Google. В основному це **NVIDIA Tesla K80** (12 Гб пам'яті, 562 МГц тактової частоти) за безкоштовною програмою Colab для студентів та дослідників, та **NVIDIA Tesla P100** (16 Гб пам'яті, 1190 МГц тактової частоти) за платною програмою передплати Colab Pro.

Специфікацію виданого GPGPU можна переглянути за допомогою відповідної команди, як показано на рисунку 2.1. Прискорювач видається один.

2.1.2 Екосистема PyTorch

PyTorch — бібліотека машинного навчання з відкритим кодом, розроблена Лабораторією Фейсбук (FAIR) на основі бібліотеки Torch під мову програмування Python. Написана на Python, C++, та розширена мові C для CUDA.

Основним функціоналом, що надає PyTorch, є:

- Швидкий розрахунок тензорів, що до того ж розпаралелюється на графічних процесорах GPU;
- Глибоке навчання нейронних мереж за допомогою систем алгоритмічної диференціації.

```
[1]   1 !nvidia-smi

Mon Nov  8 23:21:26 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 495.44       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   56C    P8    31W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Рис. 2.1. Параметри графічного прискорювача CUDA NVIDIA Tesla K80, що надає Google Colab.

PyTorch також являє собою екосистему суміжних проектів, розроблених на її основі [29]. З багатьох проектів, що належать до екосистеми Pytorch, були використані:

- **Pytorch Geometric [30]** — бібліотека геометричного глибокого навчання [22] на нерегулярних структурах, таких як графи, хмари точок та многовиди. Вона є ключовою в даній роботі, оскільки саме вона охоплює методи навчання на графах, інструменти для побудови графової нейронної мережі, методи зберігання та обробки графів, тощо.
- **skorch [31]** — високорівнева бібліотека для PyTorch, що забезпечує повну

сумісність з бібліотекою scikit-learn, та дозволяє використовувати такі її інструменти як k-кратна перехресна перевірка (k-fold cross validation).

- **Torch Metrics [32]** — бібліотека з понад 50 реалізованих метрик (таких як точність, значення функції втрат тощо) в **PyTorch** і простий у використанні API для створення користувацьких метрик.

Pytorch Lightning являє собою обгортку для PyTorch моделей. Бібліотека абстрагує такі функції як тренування моделі, її тестування, валідація, збереження тощо, зазвичай до декількох рядків коду. Принципами **Pytorch Lightning** є максимальна гнучкість, самодостатність коду моделі, усунення шаблонного (boilerplate) коду та забезпечення модульності [33]. Добре працює з бібліотекою **Torch Metrics**, у взаємодії з якою дозволяє автоматично логувати метрики, зменшуючи кількість шаблонного коду.

Модулями, тобто частинами коду, що абстрагуються, є:

- Код дослідження (об'єкт LightningModule, що успадковується від стандартного модуля Pytorch; як наслідок, код моделі можна перетворити на LightningModule зміною одного слова).
- Інженерний код (Тренування, тестування, etc. реалізовані одним рядком коду)
- Вторинний код дослідження (логування, та інші функції, що реалізуються через Callbacks)
- Робота з даними (LightningDataModule)

Бібліотека також полегшує роботу з прискорювачами, абстрагуючи необхідність згадувати та змінювати код під різні прискорювачі до однієї змінної у функціях тренування/тестування. Lightning серйозно полегшує роботу з PyTorch кодом, прискорюючи процес розробки та відлагоджування.

2.1.3 Особливості Pytorch Geometric

Pytorch Geometric має свій набір пов'язаних з його використанням труднощів. Через те, що тип даних **Pytorch Geometric** не ідентичний типу даних стандартного **Pytorch** [30], а бібліотека **Pytorch Lightning** розроблена як обгортка саме для **Pytorch**. Як наслідок, Pytorch Lightning не може абстрагувати завантажувачі даних (DataLoaders), що не дає максимально ефективно її використовувати й лишає шаблонний код.

2.2 Робота з даними

Набір даних зберігається на сервері [34] і може бути завантажений однією командою, як показано на рисунку 2.2. Подробиці щодо формату зберігання даних можна переглянути у файлі README.txt.

```
[ ]   1 dataset = TUDataset(root='data/TUDataset', name='Cuneiform')

      Downloading https://www.chrsmrrs.com/graphkerneldatasets/Cuneiform.zip
      Extracting data/TUDataset/Cuneiform/Cuneiform.zip
      Processing...
      Done!
```

Рис. 2.2. завантаження набору даних Cuneiform в серидовищі Google Colab.

2.2.1 Нормалізація

Архітектура шару SplineCNN, використаного в моделі [8], потребує, щоб ознаки ребер були представлені як псевдокоординати в інтервалі [0,1] [28].

Ознаки ребер уже представлені як псевдокоординати в наборі Cuneiform. Щоб нормалізувати їх до інтервалу [0,1] до значень було застосовано наступну формулу:

$$x_1 = \frac{(x_0 - min)}{(max - min)}$$

де x_1 нормалізоване значення псевдокоординат, x_0 не нормалізоване значення псевдокоординат, min та max — мінімальне та максимальне значення

псевдокоординат в одному об'єкту даних (символу).

2.3 Перехресна перевірка

Для отримання статистично значущих результатів використовувалась 10-тикратна перехресна перевірка. Набір даних випадковим чином розділявся на 10 частин, кожна з яких використовувалась як тестова вибірка для моделі, навчаною на іншиї дев'яти. Середнє значення ***точності*** (долі правильно класифікованих зразків у тестовій вибірці) на цих 10 частинах є *точністю перехресної перевірки*, або *точністю кроссвалідації* і може вважатися статистично значимими даними [35].

Розраховувалась середня величина обраної точності та стандартне відхилення за відповідною формулою:

$$STD=\sqrt{D}=\sqrt{\frac{\sum(x_i-M_x)^2}{n-1}}$$

Перехресна перевірка займає багато часу та ресурсів, через що застосовувалася лише для моделей, де попереднє дослідження на одній з частин тренувального набору надавала найкращі результати відповідно до обраної метрики.

Було розроблено варіанти експерименту з випадковим перемішуванням. Перемішування проводилось:

1) всередині виділеної частини, таким чином на кожній епосі подаючи зразки з тренувального набору у випадковому порядку,
2) на етапі розбиття набору на частини для 10-кратної перехресної перевірки, випадковим чином формуючи набори для навчання та перевірки **один раз і на всі епохи навчання**.

2.4 Дослідження архітектури та параметрів мережі

Досліджувалась точність на 10-кратній перехресній перевірці в залежності від архітектури та параметрів мережі.

За основу архітектури було взято приклад мережі класифікатора графів, що надає документація Pytorch Geometric [30], зі SplineCNN [28] в якості шарів.

Варіювалися такі параметри:

- типи шарів та їх кількість,
- параметри шарів,
- параметр розмір пакету,
- функції активації,
- кількість шарів Dropout та їх коефіцієнти.

2.4.1 Типи шарів

В процесі пошуку оптимальної архітектури, було досліджено значення точності в залежності від типів шарів, їх кількості та параметрів, наявності Dropout та його коефіцієнта.

Були досліджені наступні шари:

- ChebConv [36-37]
- GMMConv [36, 27]
- GeneralConv [36, 37]
- SplineConv [36, 28]

Підбір архітектури та типів шарів відбувався за допомогою десятикратної перехресної перевірки. Схеми досліджених архітектур надано в додатках А та Б.

2.4.2 Параметри шарів згортки

Досліджувався вплив параметрів шарів на точність на 10-кратній перехресній перевірці.

SplineCNN згортка, яка в якості формули фільтра використовує так званий B-слайн. ЇЇ параметри *розмір ядра* (kernel size) та *ступінь* (degree). Розмір ядра визначає максимальну відстань від вузла до його сусідів при агрегації значень з сусідства вузла, інакше кажучи скільки буде зроблено «стрибків» (англ. hops) [28]. Ступінь сплайна — це параметр B-спалйна, або базисного сплайна що рекурсивно визначає формулу за якою буде розраховуватися його значення. Базисний сплайн першого порядку визначається так:

$$B_{i,1}(x) = \begin{cases} 1 & якщо\ t_i \le x < t_{i+1} \\ 0 & інакше \end{cases}$$

де t — вузли, i — індекс вузла. Тоді для ступеня m, значення функції B-слайну B визначається через рекурсивну формулу:

$$B_{i,k+1}(x) = \omega_{i,k}(x)\, B_{i,k}(x) + [1 - \omega_{i+1,k}(x)]\, B_{i+1,k}(x)$$

де

$$\omega_{i,k}(x) = \begin{cases} \dfrac{x - t_i}{t_{i+k} - t_i}, & t_{i+k} \neq t \\ 0 & інакше \end{cases}$$

Згортка Чебишева заснована на розрахунку полінома Чебишева першого роду за формулою:

$$X' = \sum_{k=1}^{K} Z^{(k)} * \Theta^{(k)}$$

де параметр **K** — ступінь поліному, тобто розмір фільтру, побудованого на його основі [39].

Модель гаусової суміші, або **GMM** (gaussian mixture model) це

імовірнісне поняття, яке використовується для моделювання наборів даних. GMM є узагальненням гаусових розподілів і можуть використовуватися для представлення будь-якого набору даних, який можна об’єднати в кілька гаусових розподілів. Шар нейронної мережі на основі GMM реалізовано на фреймворку MoNet [27], оператор згортки визначено:

$$x_i' = \frac{1}{|N(i)|} \sum_{j \in N(i)} \frac{1}{K} \sum_{k=1}^{K} w_k(e_{i,j}) \odot \theta_k x_j$$

де

$$w_k(e) = \exp(-1/2(e-\mu_k) \sum_k^{-1} (e-\mu_k)) .$$

Загальний шар графової мережі згортки **GeneralConv** є реалізацією [38] в Pytorch Geometric.

2.4.3 Параметр шару Dropout

Використано один із методів регуляризації — Dropout — що полягає у виключенні деякого відсотка випадково обраних нейронів на різних епохах навчання, таким чином зменшуючи перенавчання. Метод Dropout використовувався між шарами згортки, та перед лінійним шаром (див. рисунок 3.13.)

2.4.4 Функції активації

- Досліджено залежність точності кроссвалідації від використаних функцій активації, серед яких [36]:
- RELU ($max(0, x)$);
- ELU ($max(0, x) + min(0, \alpha * (\exp(x) - 1))$);
- GELU ($x * \Phi(x)$, де *Φ(x)* це сукупна функція для Гаусівського розподілу);
- TANH ($\frac{\exp(x) + \exp(-x)}{\exp(x) - \exp(-x)}$);

- SILU ($x * \sigma(x)$, де *σ(x)* — логістична сигмовидна функція);
- MISH ($x * \tanh(Softplus(x))$).

2.4.5 Параметр розміру пакету зразків під час навчання мережі

Розмір пакету при навчанні графових нейронних мереж — це кількість зразків, що подаються на вхід мережі за один крок. Чим більший розмір пакету — тим більше оперативної пам'яті потребує навчання мережі.

Для традиційних нейронних мереж прийнято вважати, що менший розмір пакету дає кращий результат. Цей принцип не можна так само застосувати до графових нейронних мереж [40].

На практиці опублікованого коду можна побачити, що використовується або найбільший доступний розмір пакету як у [41], або за методологією традиційних нейронних мереж обирається невеликий розмір пакету, як у [42] та [43].

Таким чином найбільш ефективний розмір пакету не є очевидним. Для вибору оптимального розміру пакету за такими метриками як швидкість навчання мережі та точність було проведено десятикратну перехресну перевірку на одній і тій самій мережі з фіксованою кількістю епох — 300 — для визначення оптимального значення цього параметру.

2.5 Метод ранньої зупинки

Метод ранньої зупинки — це метод регулярізації, що використовується для запобігання перенавчання та покращення результатів. Для цього частина набору даних виділяється як набір для валідації. В кінці кожної епохи відбувається перевірка заданої метрики на валідаційній вибірці, і коли значення цієї метрики не покращується задану кількість епох (параметр *терпіння*) навчання зупиняється, і повертається модель, що давала найкращі результати на валідаційній вибірці [44].

В якості валідаційної вибірки взято частину тренувальної вибірки. При використанні методу ранньої зупинки в якості валідаційного набору використано одну п'ятнадцяту тренувальної вибірки (приблизно 16 зразків).

В якості метрики, за значенням якої спостерігали на валідаційній вибірці використано точність класифікації, параметр терпіння — 200 епох.

3 Результати дослідження

3.1 Дослідження архітектури та параметрів мережі

3.1.1 Тип шару

За методикою, описаною у 2.4 було досліджено точність при використанні різних типів шарів графових нейронних мереж.

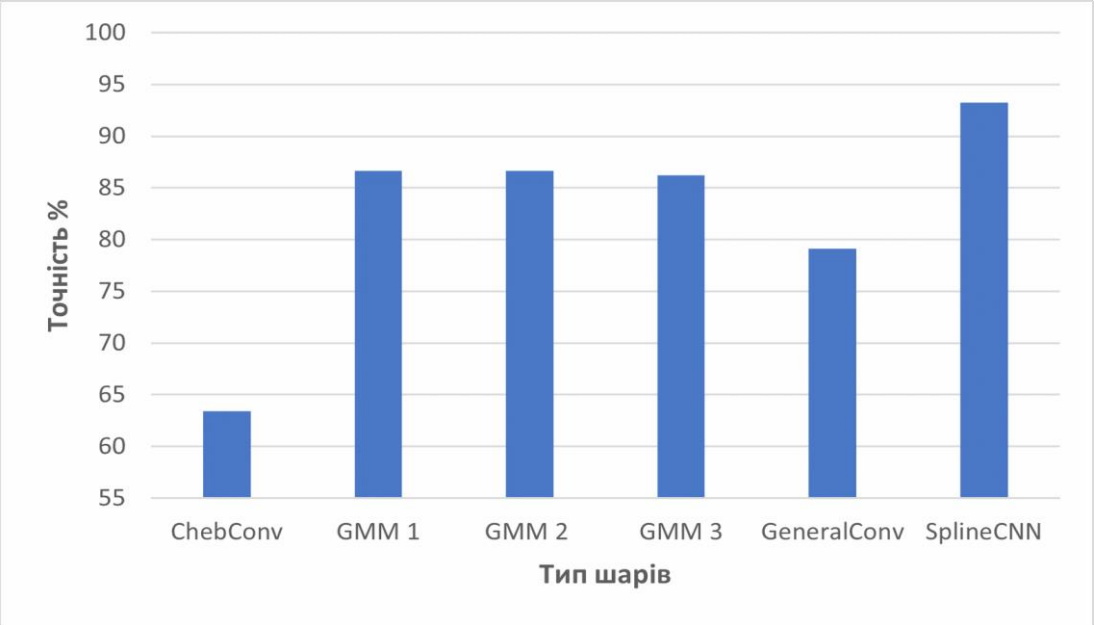

Рис. 3.1. Точність кроссвалідації отримана в результаті 10-кратної перехресної перевірки в залежності від використаної архітектури. Досліджені архітектури представлені в додатку А.

Як показано на графіку 3.1, найкращий результат з досліджених надала мережа SplineCNN.

3.1.2 Дослідження варіантів архітектури

Було досліджено точність та час навчання мережі в залежності від архітектури з використанням шару SplineCNN. Досліджені архітектури представлено в додатку Б.

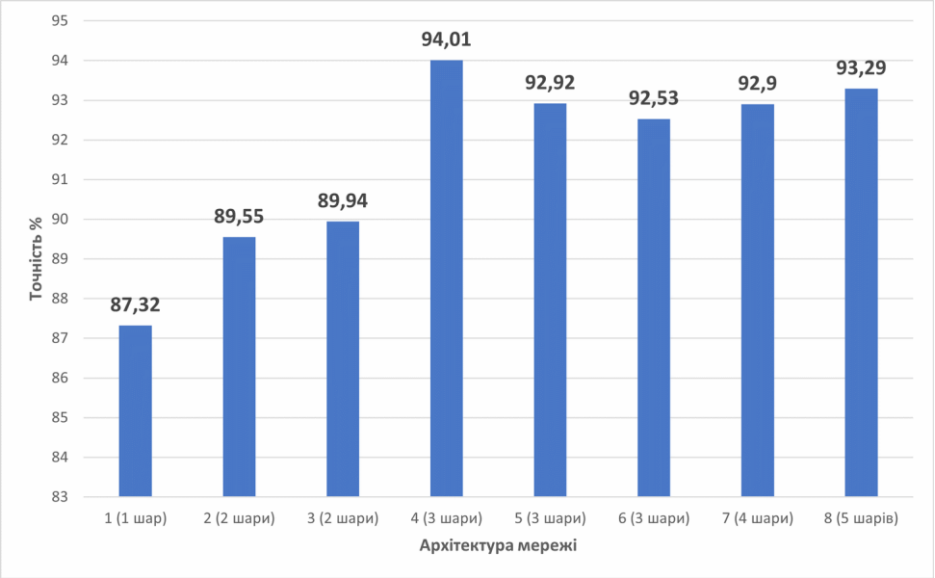

Рис. 3.2. Точність кроссвалідації в залежності від архітектури з використанням шару SplineCNN. Досліджені архітектури на основі SplineCNN представлено в додатку Б.

Архітектура, при якій була отримана найвища точність — номер 4 з 3 шарами як видно на рисунку 3.2. Вона представлена на рисунку 3.12.

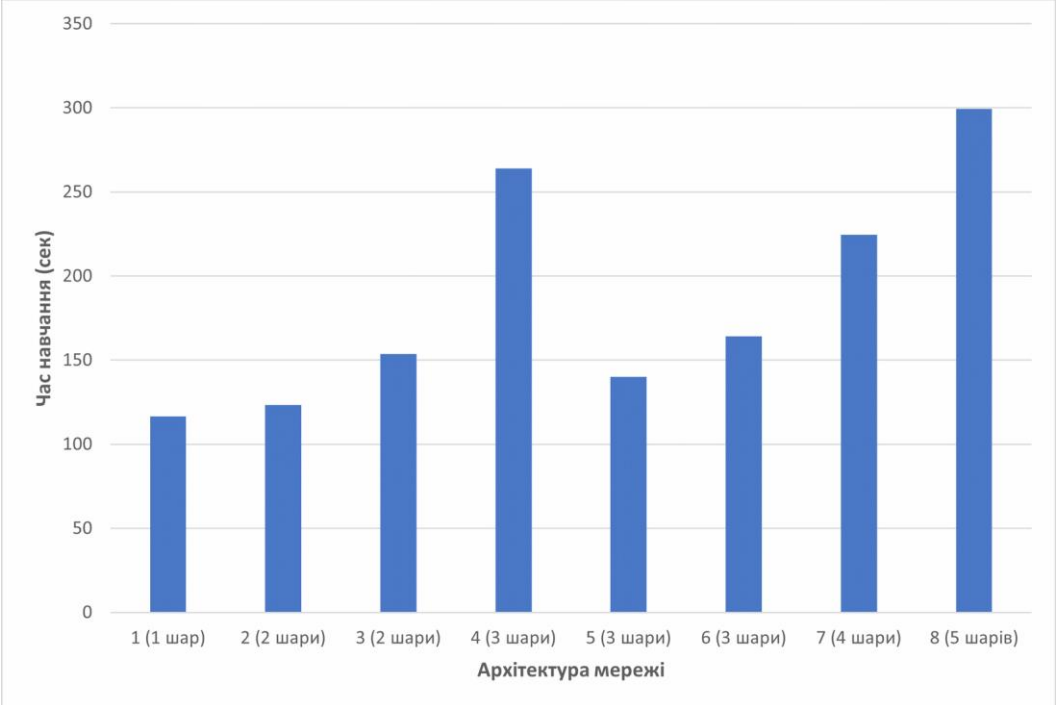

Рис. 3.3. Час навчання в залежності від архітектури з використанням шару SplineCNN. Архітектури представлено в додатку Б.

Середній час навчання мережі на архітектурі номер 4 становить приблизно 247 секунд, а 10-кратна перехресна перевірка на ній займає приблизно 40 хвилин. Як видно на рисунку 3.3, навчання мережі на архітектурі номер 4 займає 2 найдовший час з досліджених архітектур.

3.1.3 Гіперпараметри шару SplineCNN

Досліджено залежність точності мережі на основі шарів SplineCNN від її параметрів, відповідно описаних у 2.4.2 — ступеню базисного сплайна та розміру ядра.

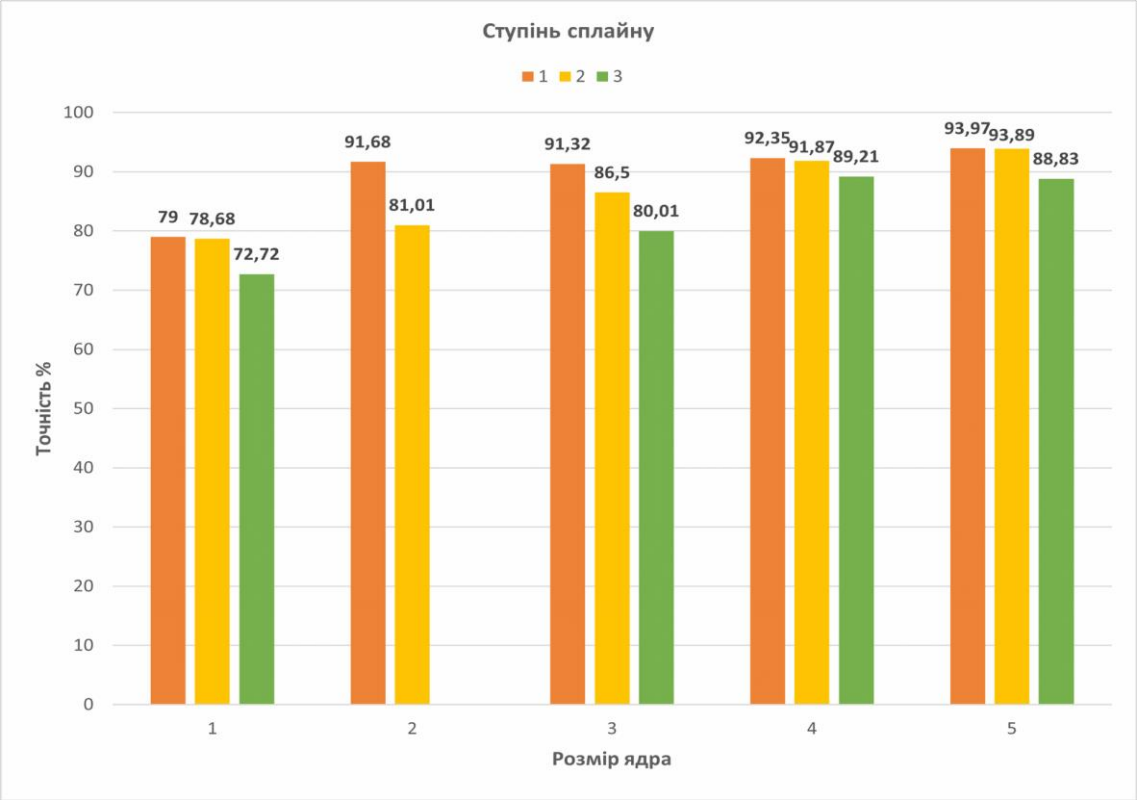

Рис. 3.4. Точність в результаті 10-кратної перехресної перевірки в залежності від параметрів шару SplineCNN.

На рисунку 3.4 можна побачити тенденцію до зменшення точності зі збільшенням ступеню сплайна, та збільшення точності зі збільшенням розміру ядра. Найвища точність отримана при параметрах розмір ядра **5** та ступінь сплайна **1**.

3.1.4 Розмір пакету

Відповідно до методології, описаної в 2.4 було досліджено вплив розміру пакету на цільову метрику та час навчання моделі. На рисунку 3.6 видно, збільшення розміру пакету призводить до зменшення часу навчання.

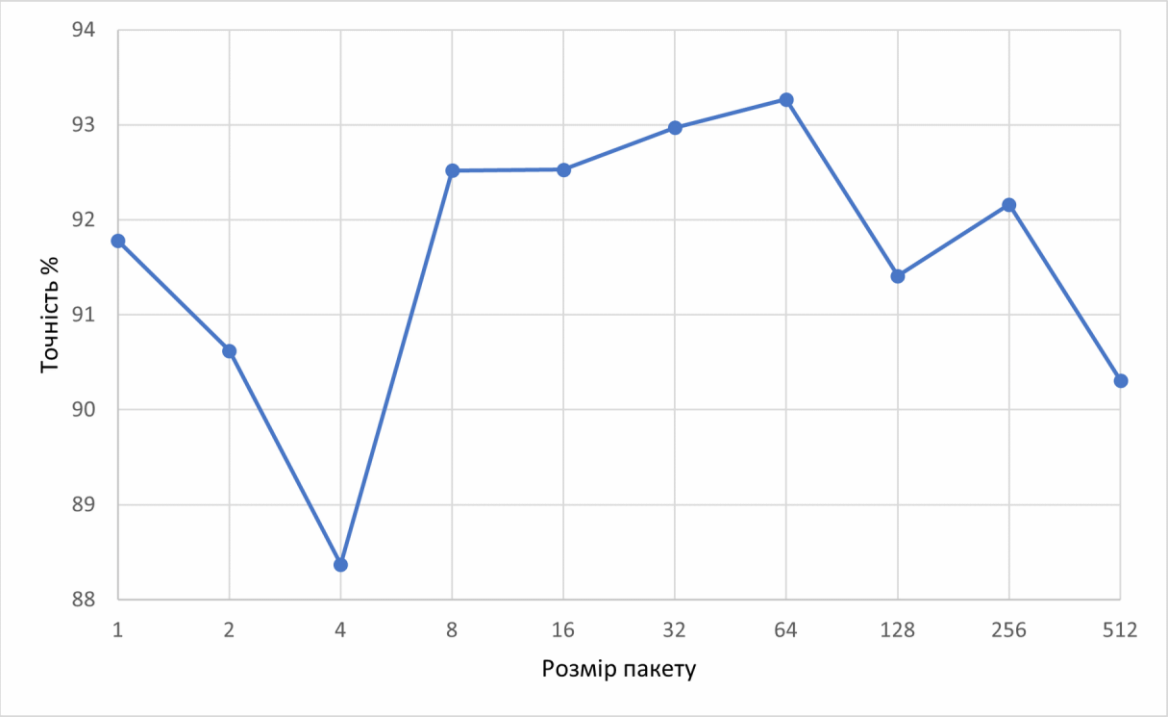

Рис. 3.5. Точність кроссвалідації в залежності від параметру розмір пакету.

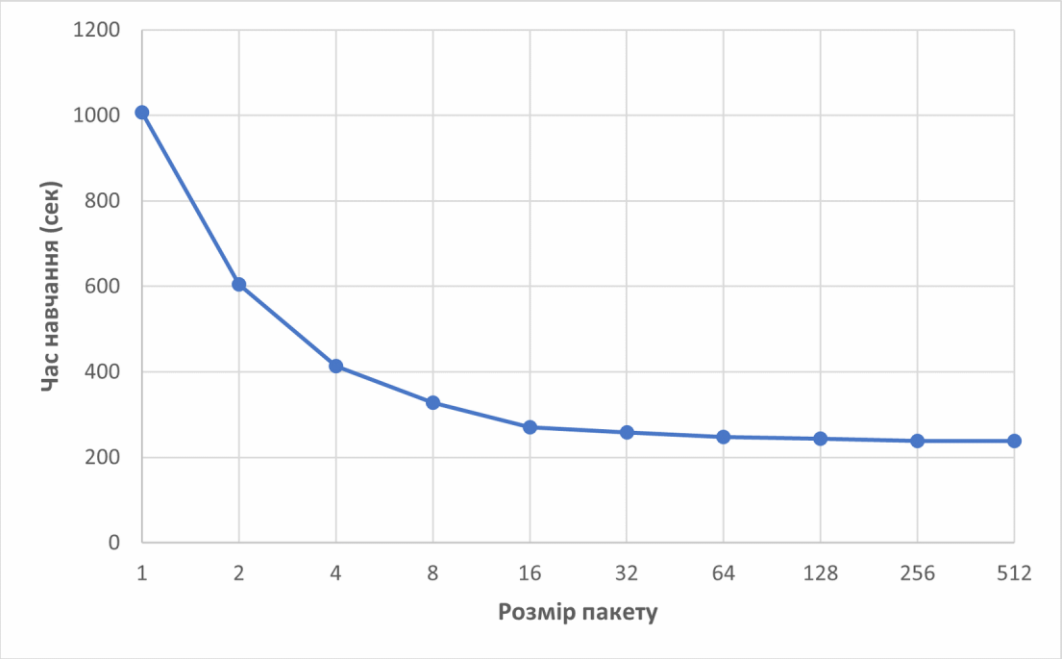

Рис. 3.6. Час навчання в залежності від розміру пакету.

Як видно з рисунків 3.5 та 3.6, після значення в 64 досягається поріг, після якого збільшення пакету перестає помітно зменшувати час навчання, а точність кроссвалідації починає зменшуватися. Тому в якості оптимального значення параметру розмір пакету обрано **64**.

3.1.5 Функція активації

Досліджено залежність точності корссвалідації залежності від використовуваної функції активації. Найкращу точність отримано при використанні функції ELU, як видно на рисунку 3.7.

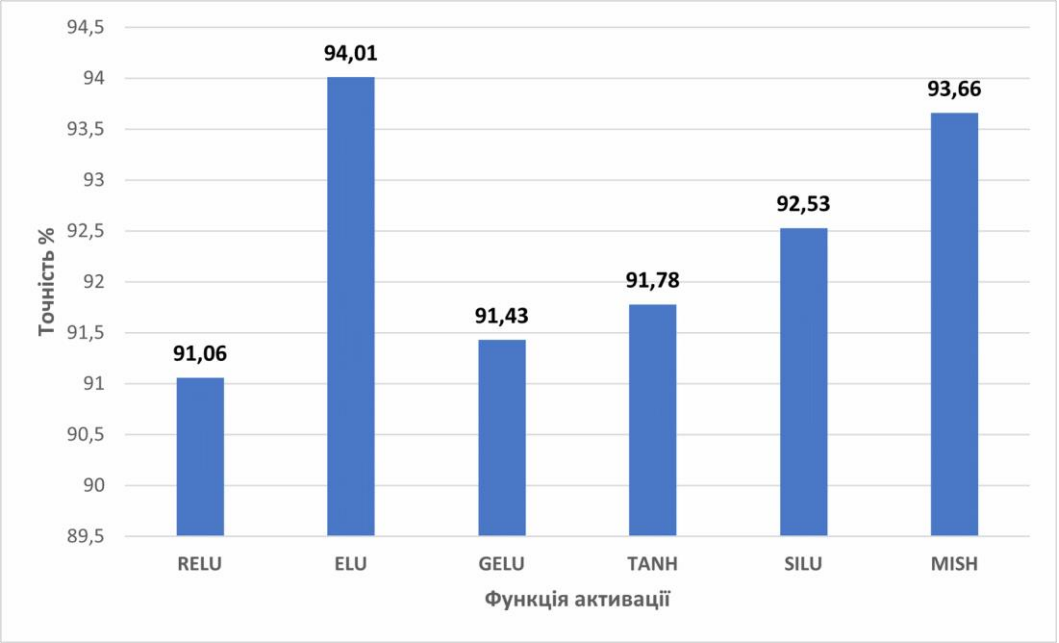

Рис. 3.7. Точність кроссвалідації в залежності від використаної функції активації.

3.1.6 Шари Dropout

Досліджено точність в залежності від коефіцієнтів шарів Dropout між шарами згортки (**внутрішній шар Dropout**) та перед лінійним шаром (**останній шар Dropout**) відповідно до методології, описаної у розділі 2.4.

Як видно з рисунку 3.8 та таблиці 3.1, найбільша точність отримана за таких значень параметрів: p внутрішнього шару Dropout **0,5**, p останнього шару Dropout **0,2**.

*Таблиця 3.1: Точність кроссвалідації у відсотках, отримана при 10-кратній перехресній перевірці в залежності від коефіцієнта **p** шарів Dropout.*

внутрішній шар Dropout	останній шар Dropout			
	0,2	**0,5**	**0,7**	**0,9**
0,2	94,04	93,98	91,76	89,54
0,5	94,38	92,5	92,13	88,36
0,7	94,04	94,01	92,17	86,52
0,9	92,17	89,88	89,51	72,63

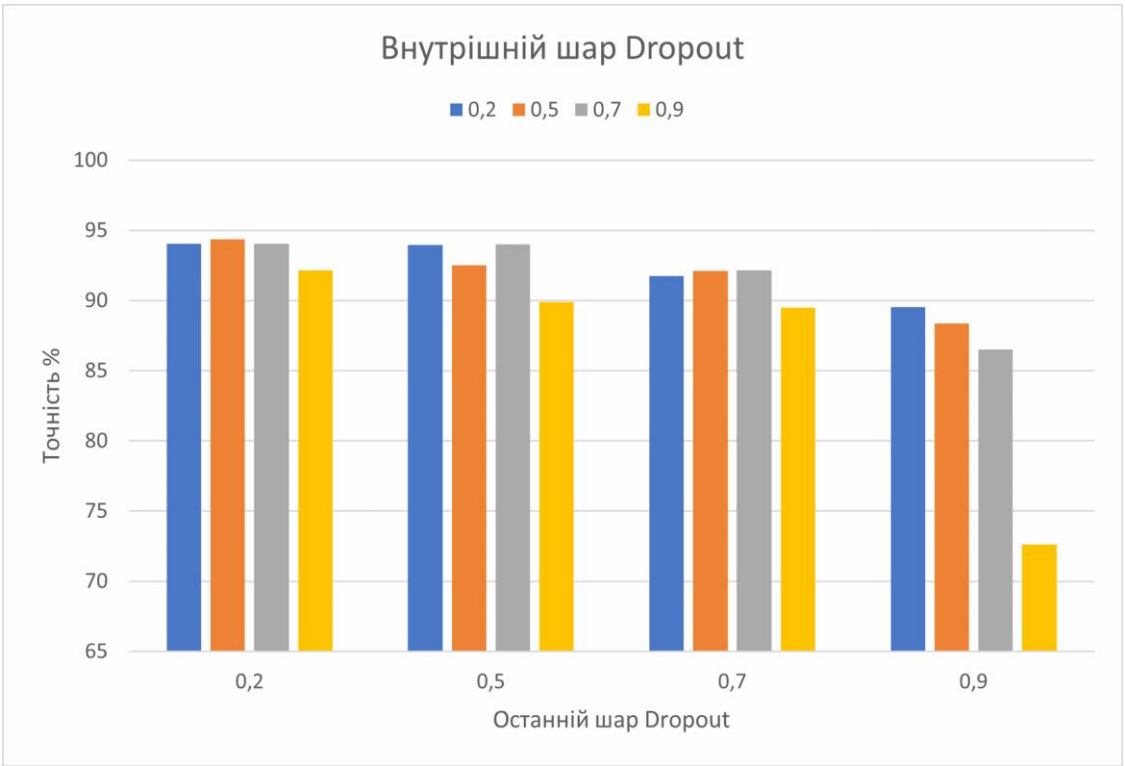

Рис. 3.8. Залежність точності кроссвалідації, отримана в результаті десятикратної перехресної перевірки, від параметрів шарів Dropout.

3.2 Дослідження впливу ранньої зупинки на точність

Як зазначено в [8], однією з сильних сторін шару SplineCNN є його можливість швидко навчатися на невеликій вибірці. За спроби покращити результат використовуючи ранню зупинку навчання (див. 2.5) дійсно було виявлено, що SplineCNN загалом успішно досягає максимуму своїх можливостей за 300 епох.

Криві навчання отримані за допомогою інструмента TensorBoard [45]. Через особливості TensorBoard метрики логуються з залежністю від кроку навчання (кожний раз при розрахунку пакету зразків), і при відображенні метрик в якості вісі абсцих відображає саме кроки, а не епохи. Всюди в цьому розділі розмір пакету дорівнює 64, одна епоха на тренувальній вибірці відповідає 4 крокам.

Рис. 3.9. Крива навчання для навчання 300 епох без ранньої зупинки. Графік отримано в TensorBoard.

Як видно на рисунках 3.10, 3.11 точність на валідаційній вибірці починає зменшуватися, в той час, як тренувальна досягає екстремуму і починає зменшуватися.

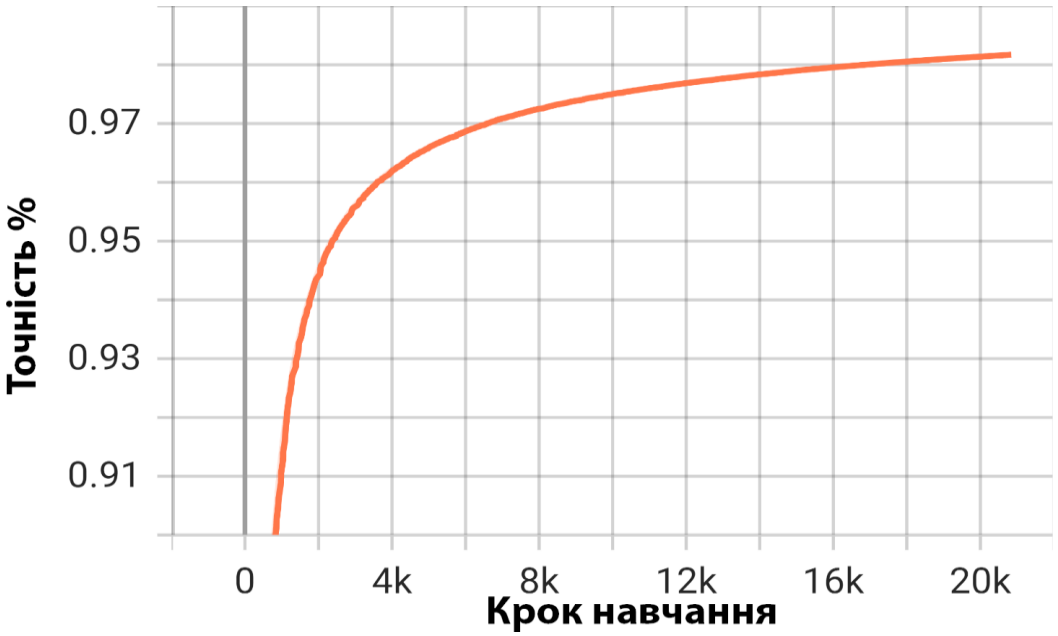

Рис. 3.10. Змніна тренувальної точності в залежності від кроку, графік отримано в TensorBoard.

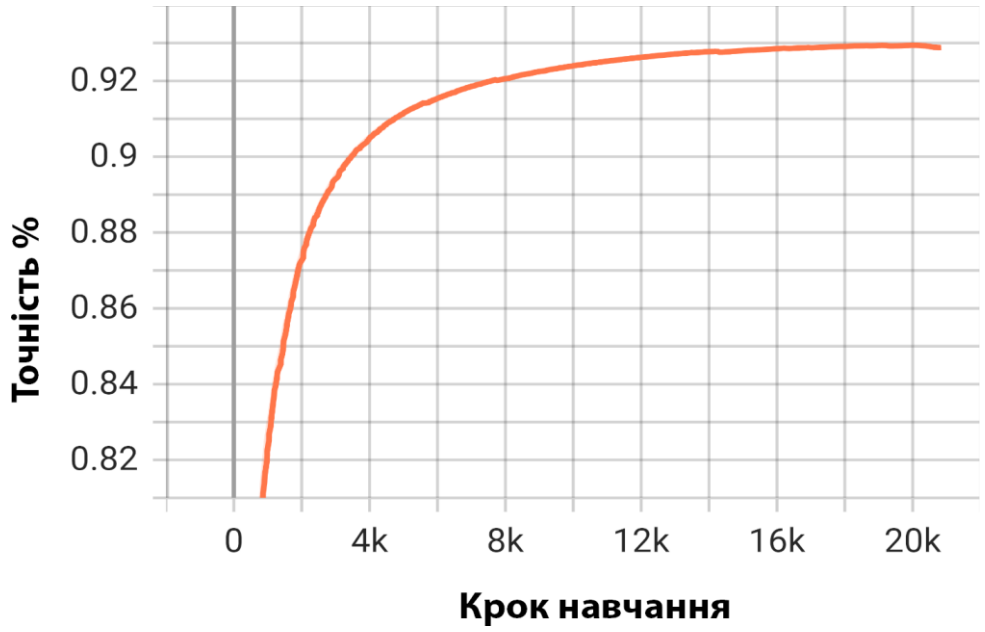

Рис. 3.11. Точність на валідаційній вибірці в залежності від кроку, графік отримано в TensorBoard.

Таблиця 3.2: Точність 10-кратної перехресної перевірки, розписана по частинам при використанні методу ранньої зупинки з параметром терпіння рівним 200 епох. Колонка епох позначає епоху, на якій навчання було зупинено.

частина	**точність %**	**епохи**
0	81,48	538
1	85,18	4505
2	92,59	7122
3	100	1291
4	85,18	4338
5	96,29	4502
6	100	2856
7	100	8808
8	92,3	4819
9	92,3	1507
середня	**92,532**	**4028,6**

Як видно з таблиці 3.2, точність отримана з використанням методу ранньої зупинки не відрізняються суттєво від точності отриманої при навчанні на 300 епохах і становить **92,53%**. Хоча з рисунків 3.10 та 3.11 видно, що незначні покращення можуть продовжуватися упродовж багатьох епох, не призводячи при цьому до значущого покращення узагальнюючої здатності мережі.

3.3 Аналіз результатів

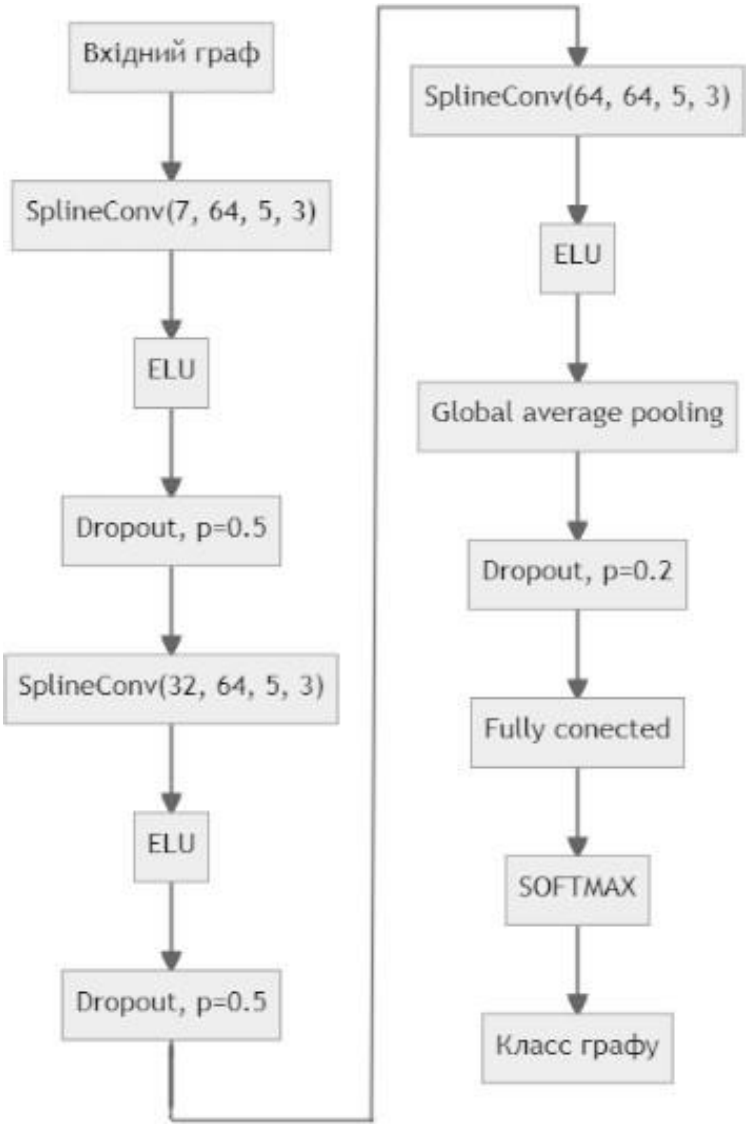

Рис. 3.12. Підібрана архітектура з використанням шару SplineCNN. SplineConv (вхідна розмірність, вихідна рзмірність, розмір ядра, ступінь сплайна).

Для визначення відтворюваності значень точності класифікатора були проведені 10 оцінок точностей за допомогою 10-кратної перехресної перевірки. Середня точність для оптимальної архітктури (рис. 3.12) і значень

гіперпараметрів склала 93,74% при стандартному відхиленні 1,44%.

Як видно з графіку на рисунку 3.13 точність на підібраній архітектурі суттєво вища за результат в [8] що отриманий за такою ж процедурою, — 10 десятикратних перевірок, — та є близьким до результатів, що в літературі отримано за допомогою аугментації навчальних даних.

Можна очікувати, що аугментація даних дозволить ще більше покращити узагальнюючу здатність мережі.

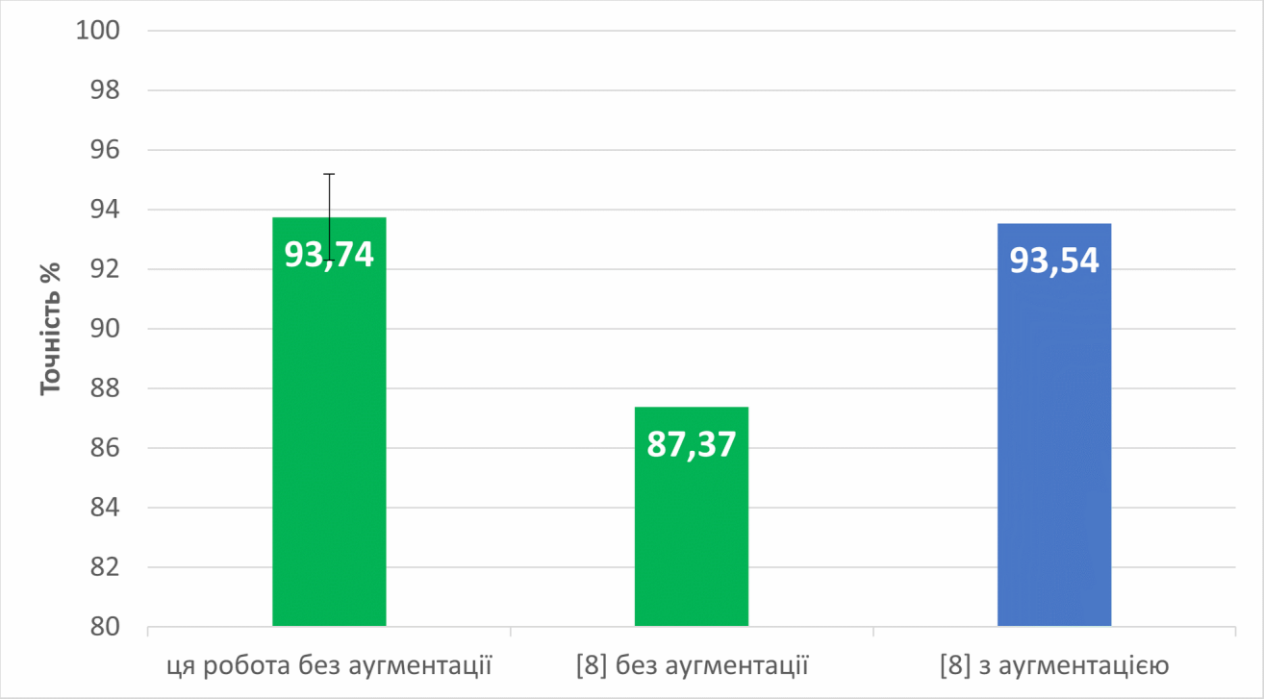

Рис. 3.13. Точність отримана в результаті 10 10-кратних перевірок, разом зі стандартним відхиленням, отримана при параметрах, що давали найкращі результати.

Висновки

У ході виконання дипломної роботи магістра:

1) Побудовані моделі класифікаторів клинописних символів набору даних Cuneiform на основі графових нейронних мереж різних типів архітектур (ChebConv, GMM, GeneralConv, SplineCNN).

2) Визначені залежності точностей побудованих моделей, що оцінювались методом 10-кратної перехресної перевірки, від архітектури графових нейронних мереж та їх різних гіперпараметрів (розмір ядра згортки, степінь сплайну, розмір пакету навчальних прикладів, коефіцієнт шару Dropout).

3) Точність отриманої оптимальної моделі графової мережі з типом шарів SplineCNN, функцією активації ELU, розміром ядра згортки 5, ступенем сплайну 1 досягнула **93.7%**, що більш ніж на **6%** перевищує точність представленої у літературі моделі класифікатора, що визначалась за аналогічною методикою (без використання штучного розширення даних).

Список використаних джерел

1. B. Bogacz, M. Gertz, and H. Mara, “Character retrieval of vectorized cuneiform script,” in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, 2015, pp. 326–330. doi: 10.1109/ICDAR.2015.7333777.
2. H. Mara, “HeiCuBeDa Hilprecht - Heidelberg Cuneiform Benchmark Dataset for the Hilprecht Collection.” heiDATA, 2019. doi: 10.11588/DATA/IE8CCN.
3. I. J. Gelb, “Sumerian language,” *Encyclopedia Britannica*. Mar. 20, 2019. [Online]. Available: https://www.britannica.com/topic/Sumerian-language
4. T. Dencker, P. Klinkisch, S. M. Maul, and B. Ommer, “Deep learning of cuneiform sign detection with weak supervision using transliteration alignment,” *PLoS ONE*, vol. 15, no. 12, p. e0243039, Dec. 2020, doi: 10.1371/journal.pone.0243039.
5. D. Charpin, *Reading and Writing in Babylon*. Harvard University Press, 2010.
6. K. Radner and E. Robson, *The Oxford Handbook of Cuneiform Culture*. Oxford University Press, 2011.
7. I. L. Finkel and J. Taylor, *Cuneiform*. J. Paul Getty Museum, 2015.
8. N. M. Kriege, M. Fey, D. Fisseler, P. Mutzel, and F. Weichert, “Recognizing Cuneiform Signs Using Graph Based Methods,” in *Proceedings of The International Workshop on Cost-Sensitive Learning*, Aug. 2018, pp. 31–44. Accessed: Mar. 20, 2022. [Online]. Available: https://proceedings.mlr.press/v88/kriege18a.html
9. I. L. Finkel, “REPORT ON THE SIDON CUNEIFORM TABLET, ARCHAEOLOGY & HISTORY IN THE LEBANON ISSUE TWENTY FOUR: WINTER,” 2006.
10. C. B. F. Walker, *Cuneiform*. University of California Press, 1987.
11. “CDLI - Cuneiform Digital Library Initiative.” https://cdli.ucla.edu/ (accessed Mar. 20, 2022).
12. A. Sahala, “Contributions to Computational Assyriology,” Aug. 2021, Accessed: Apr. 01, 2022. [Online]. Available: https://helda.helsinki.fi/handle/10138/332924
13. “Hilprech Archive Online.” https://hilprecht.mpiwg-berlin.mpg.de/object3d/30696 (accessed Mar. 20, 2022).
14. H. Mara and B. Bogacz, “Breaking the Code on Broken Tablets: The Learning Challenge for Annotated Cuneiform Script in Normalized 2D and 3D Datasets,” in *2019 International Conference on Document Analysis and Recognition (ICDAR)*, Sydney, Australia, Sep. 2019, pp. 148–153. doi: 10.1109/ICDAR.2019.00032.
15. D. Fisseler and F. Weichert, “Towards an interactive and automated script feature analysis of 3D scanned cuneiform tablets,” p. 10, 2013.
16. T. Collins, S. I. Woolley, E. Gehlken, and E. Ch’ng, “Automated Low-Cost Photogrammetric Acquisition of 3D Models from Small Form-Factor Artefacts,” *Electronics*, vol. 8, no. 12, p. 1441, Dec. 2019, doi: 10.3390/electronics8121441.

17. S. Tyndall, “Toward Automatically Assembling Hittite-Language Cuneiform Tablet Fragments into Larger Texts,” in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Jeju Island, Korea, Jul. 2012, pp. 243–247. [Online]. Available: https://aclanthology.org/P12-2048
18. на, “The ‘Electronic Babylonian Literature’ (eBL).” https://www.ebl.lmu.de/ (accessed Apr. 01, 2022).
19. E. Fetaya, Y. Lifshitz, E. Aaron, and S. Gordin, “Restoration of fragmentary Babylonian texts using recurrent neural networks,” *Proc. Natl. Acad. Sci. U.S.A.*, vol. 117, no. 37, pp. 22743–22751, Sep. 2020, doi: 10.1073/pnas.2003794117.
20. A. H. S. Hamdany, R. R. Omar-Nima, and L. H. Albak, “Translating cuneiform symbols using artificial neural network,” *TELKOMNIKA*, vol. 19, no. 2, p. 438, Apr. 2021, doi: 10.12928/telkomnika.v19i2.16134.
21. G. G. W. Müller and D. Schwemer, “13 Hethitologie-Portal Mainz (HPM). A Digital Infrastructure for Hittitology and Related Fields in Ancient Near Eastern Studies,” in *Crossing Experiences in Digital Epigraphy*, A. De Santis and I. Rossi, Eds. De Gruyter, 2018, pp. 167–179. doi: 10.1515/9783110607208-014.
22. M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, “Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges,” 2021, doi: 10.48550/ARXIV.2104.13478.
23. Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A Comprehensive Survey on Graph Neural Networks,” *IEEE Trans. Neural Netw. Learning Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021, doi: 10.1109/TNNLS.2020.2978386.
24. A. Sperduti and A. Starita, “Supervised neural networks for the classification of structures,” *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 714–735, May 1997, doi: 10.1109/72.572108.
25. J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral Networks and Locally Connected Networks on Graphs,” 2013, doi: 10.48550/ARXIV.1312.6203.
26. A. Micheli, “Neural Network for Graphs: A Contextual Constructive Approach,” *IEEE Trans. Neural Netw.*, vol. 20, no. 3, pp. 498–511, Mar. 2009, doi: 10.1109/TNN.2008.2010350.
27. F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, “Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, Jul. 2017, pp. 5425–5434. doi: 10.1109/CVPR.2017.576.
28. M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, “SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels,” 2017, doi: 10.48550/ARXIV.1711.08920.
29. “PyTorch.” https://www.pytorch.org (accessed Mar. 20, 2022).
30. “torch_geometric.data — pytorch_geometric 2.0.5 documentation.” https://pytorch-geometric.readthedocs.io/en/latest/modules/data.html (accessed Mar. 20, 2022).

31. *skorch-dev/skorch*. skorch-dev, 2022. Accessed: Mar. 20, 2022. [Online]. Available: https://github.com/skorch-dev/skorch
32. "torchmetrics · PyPI." https://pypi.org/project/torchmetrics/ (accessed Mar. 20, 2022).
33. W. Falcon and The PyTorch Lightning team, *PyTorch Lightning*. 2019. doi: 10.5281/zenodo.3828935.
34. "TUDataset," *TUDataset*. https://chrsmrrs.github.io/datasets/datasets/ (accessed Apr. 11, 2022).
35. J. Brownlee, *Statistical Methods for Machine Learning: Discover how to Transform Data into Knowledge with Python*. Machine Learning Mastery, 2018.
36. "torch_geometric.nn — pytorch_geometric 2.0.5 documentation."https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html (accessedApr. 18, 2022).
37. M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," 2016, doi: 10.48550/ARXIV.1606.09375.
38. J. You, R. Ying, and J. Leskovec, "Design Space for Graph Neural Networks," 2020, doi: 10.48550/ARXIV.2011.08843.
39. Z. Liu and J. Zhou, *Introduction to graph neural networks*. San Rafael, California: Morgan & Claypool Publishers, 2020. doi: 10.2200/S00980ED1V01Y202001AIM045.
40. Y. Hu, A. Levi, I. Kumar, Y. Zhang, and M. Coates, "On Batch-size Selection for Stochastic Training for Graph Neural Networks," 2020.
41. G. Li, C. Xiong, A. Thabet, and B. Ghanem, "Deepergcn: All you need to train deeper gcns," *arXiv preprint arXiv:2006.07739*, 2020.
42. J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.
43. D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," *Advances in neural information processing systems*, vol. 32, 2019.
44. L. Prechelt, "Early Stopping — But When?," in *Neural Networks: Tricks of the Trade*, vol. 7700, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67. doi: 10.1007/978-3-642-35289-8_5.
45. "TensorBoard | TensorFlow." https://www.tensorflow.org/tensorboard (accessed May 07, 2022).

Додатки

Додаток А. Архітектури різних шарів

Таблиця А.1. Архітектура ***ChebConv.*** *ChebConv (розмірність вхідного зразка, розмірність вихідного зразка, розмір фільтру Чебишева).*

Вхідний граф
ChebConv(7, 32, K=1)
ELU
Dropout, p=0.5
ChebConv(32, 64, K=1)
ELU
Dropout, p=0.5
ChebConv(64, 64, K=1)
ELU
Global mean pooling
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

Таблиця А.2. Архітектура GMM 1. GMM (розмірність вхідного зразка, розмірність вихідного зразка, розмірність псевокоординат, кількість ядер).

Вхідний граф
GMMConv(7, 32, dim=2, 1)
ELU
Dropout, p=0.5
GMMConv(32, 64, dim=2, 1)
ELU
Dropout, p=0.5
GMMConv(32, 64, dim=2, 1)
ELU
Global mean pooling
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

Таблиця А.3. Архітектура GMM 2. GMM (розмірність вхідного зразка, розмірність вихідного зразка, розмірність псевокоординат, кількість ядер).

Вхідний граф
GMMConv(7, 32, dim=2, 5)
ELU
Dropout, p=0.5
GMMConv(32, 64, dim=2, 5)
ELU
Dropout, p=0.5
GMMConv(32, 64, dim=2, 5)
ELU
Global mean pooling
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

Таблиця А.4. Архітектура GMM 3. GMM (розмірність вхідного зразка, розмірність вихідного зразка, розмірність псевокоординат, кількість ядер)

Вхідний граф
GMMConv(7, 32, dim=2, 10)
ELU
Dropout, p=0.5
GMMConv(32, 64, dim=2, 10)
ELU
Dropout, p=0.5
GMMConv(32, 64, dim=2, 10)
ELU
Global mean pooling
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

Таблиця А.5. Архітектура GeneralConv. GeneralConv (розмірність вхідного зразка, розмірність вихідного зразка).

Вхідний граф
GeneralConv(7, 32)
ELU
Dropout, p=0.5
GeneralConv(32, 64)
ELU
Dropout, p=0.5
GeneralConv(64, 64)
ELU
Global mean pooling
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

Таблиця А.5. Архітектура SplineConv. *SplineConv (розмірність вхідного зразка, розмірність вихідного зразка, розмірність псевдокоординат, розмір ядра, ступінь сплайна).*

Вхідний граф
SplineConv(7, 32, dim=2, 5, 3)
ELU
Dropout, p=0.5
SplineConv(32, 64, dim=2, 5, 3)
ELU
Dropout, p=0.5
SplineConv(64, 64, dim=2, 5, 3)
ELU
Global mean pooling
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

Додаток Б. Архітектури з використанням SplineCNN

*Таблиця Б.1. Архітектура **1 (1 шар).***

Вхідний граф
SplineConv(7, 64, 5, 3)
ELU
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

*Таблиця Б.2. Архітектура **2 (2 шари)**.*

Вхідний граф
SplineConv(7, 8, 5, 3)
ELU
Dropout, p=0.5
SplineConv(8, 64, 5, 3)
ELU
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

*Таблиця Б.3. Архітектура **3 (2 шари)**.*

Вхідний граф
SplineConv(7, 16, 5, 3)
ELU
Dropout, p=0.5
SplineConv(16, 64, 5, 3)
ELU
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

*Таблиця Б.4. Архітектура **4 (3 шари)**.*

Вхідний граф
SplineConv(7, 32, dim=2, 5, 3)
ELU
Dropout, p=0.5
SplineConv(32, 64, dim=2, 5, 3)
ELU
Dropout, p=0.5
SplineConv(64, 64, dim=2, 5, 3)
ELU
Global mean pooling
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

*Таблиця Б.5. Архітектура **5 (3 шари)**.*

Вхідний граф
SplineConv(7, 8, 5, 3)
ELU
Dropout, p=0.5
SplineConv(8, 16, 5, 3)
ELU
Dropout, p=0.5
SplineConv(16, 64, 5, 3)
ELU
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

*Таблиця Б.6. Архітектура **6 (3 шари)**.*

Вхідний граф
SplineConv(7, 16, 5, 3)
ELU
Dropout, p=0.5
SplineConv(16, 32, 5, 3)
ELU
Dropout, p=0.5
SplineConv(32, 64, 5, 3)
ELU
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

*Таблиця Б.7. Архітектура **7 (4 шари)**.*

Вхідний граф
SplineConv(7, 8, 5, 3)
ELU
Dropout, p=0.5
SplineConv(8, 16, 5, 3)
ELU
Dropout, p=0.5
SplineConv(16, 64, 5, 3)
ELU
Dropout, p=0.5
SplineConv(64, 64, 5, 3)
ELU
Global mean pooling
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

*Таблиця Б.8. Архітектура **8 (5 шари)**.*

Вхідний граф
SplineConv(7, 8, 5, 3)
ELU
Dropout, p=0.5
SplineConv(8, 16, 5, 3)
ELU
Dropout, p=0.5
SplineConv(16, 32, 5, 3)
ELU
Dropout, p=0.5
SplineConv(32, 64, 5, 3)
ELU
Dropout, p=0.5
SplineConv(64, 64, 5, 3)
ELU
Dropout, p=0.2
Fully Conected
Softmax
Класс вхідного графу

Додаток В. Лістинг коду

```
# -*- coding: utf-8 -*-
"""08_05_22_GPU_CUDA11.1_codelisting.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1yjPASZCBmNZDQpfC9KYq13jO05GO6YSk
"""

!pip install torch==1.10.0+cu111 torchvision==0.11.0+cu111 torchaudio==0.10.0 -f https://download.pytorch.org/whl/torch_stable.html

# Install required packages.
!pip install -q torch-scatter torch-sparse torch-cluster torch-spline-conv torch-geometric -f https://data.pyg.org/whl/torch-1.10.0+cu111.html
!pip install -q pytorch-lightning
!pip install torchmetrics

import os
os.environ['CUDA_LAUNCH_BLOCKING'] = "1"

import torch
import torch.nn.functional  as  F
import torch_geometric.transforms as T
import torch.optim as optim
import torchmetrics
from torch.optim import lr_scheduler
from torch_geometric.datasets import TUDataset
from torch_geometric.data import DataLoader
import torch_geometric.nn
from torch_geometric.nn import GraphConv, TopKPooling, SplineConv
from torch_geometric.nn import global_mean_pool as gap, global_max_pool as gmp

from sklearn.model_selection import KFold

import sys
import numpy as np
import math

import pytorch_lightning as pl
from pytorch_lightning.callbacks import EarlyStopping, ModelCheckpoint
from pytorch_lightning.loggers import TensorBoardLogger
```

```
from math import sin, cos
import copy
import random

def rotation(graph):
  theta = np.random.uniform(-Theta, Theta)
  a = np.array([[cos(theta), -sin(theta)],
                [sin(theta), cos(theta)]])

  g = graph.edge_attr[:,:2]
  p = np.apply_along_axis(np.dot, 1, g, a)
  graph.edge_attr[:,:2] = torch.from_numpy(p[:,:2])
  return graph

def scaling(graph):
  s = np.random.uniform(1/S, S, size=2)
  a = np.array([[s[0], 0],
                [0, s[1]]])

  g = graph.edge_attr[:,:2]
  p = np.apply_along_axis(np.dot, 1, g, a)
  graph.edge_attr[:,:2] = torch.from_numpy(p[:,:2])
  return graph

def translation(p):
  if p[0]==0 and p[1]==0:
    return np.asarray(p)
  else:
    return np.asarray(p) + np.random.uniform(-TT, TT, size=2)

# TT = 0.1;
# Theta = 0.6;
# S = 1.4;

TT = 0;
Theta = 0;
S = 1;

class my_affine_transforms(T.BaseTransform):
  def___init__(self, args = []):
    pass

  def___call__(self, data: torch_geometric.data.data.Data):
    data = rotation(data)
    data = scaling(data)

    data.edge_attr[:, :2] =
torch.from_numpy(np.apply_along_axis(translation, 1,
data.edge_attr[:, :2]))
```

```
    return data

  def__repr__(self) -> str:
    return (f'{self.__class__.__name__}')

class my_normalization(T.BaseTransform):
  def__init__(self, args = []):
    pass

  def my_norm(x, a, b):
    return (x-a)/(b-a)

  def__call__(self, data: torch_geometric.data.data.Data):
    my_norm_vect  =  np.vectorize(my_normalization.my_norm)
    x = np.array(data.edge_attr)
    x[:,0] = my_norm_vect(x[:,0],  min(x[:,0]), max(x[:,0]))
    x[:,1] = my_norm_vect(x[:,1], min(x[:,1]), max(x[:,1]))
    data.edge_attr[:, 0] = torch.from_numpy(x[:, 0])
    data.edge_attr[:, 1] = torch.from_numpy(x[:, 1])
    return data

  def__repr__(self) -> str:
    return (f'{self.__class__.__name__},')

dataset = TUDataset(root='data/TUDataset', name='Cuneiform',
transform=T.Compose([my_normalization()]))

class Net(pl.LightningModule):
    def__init__(self):
        super(Net, self).__init__()

        degree = 1
        kernel_size = 5

        dim=dataset.num_edge_features


        self.conv1 = SplineConv(dataset.num_features, 32, dim=dim,
kernel_size=kernel_size, degree=degree)
        self.conv2 = SplineConv(32, 64, dim=dim,
kernel_size=kernel_size, degree=degree)
        self.conv3 = SplineConv(64, 64, dim=dim,
kernel_size=kernel_size, degree=degree)

        self.lin1 = torch.nn.Linear(64, dataset.num_classes)

        self.dropout = 0.2

        self.train_accuracy = torchmetrics.Accuracy()
```

```
        self.val_accuracy = torchmetrics.Accuracy()
        self.test_accuracy = torchmetrics.Accuracy()

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.batch
        pseudo = data.edge_attr

        in_dropout = 0.2

        x = F.elu(self.conv1(x, edge_index, pseudo))
        x = F.dropout(x, in_dropout, training = self.training)
        x = F.elu(self.conv2(x, edge_index, pseudo))
        x = F.dropout(x, in_dropout, training = self.training)
        x = F.elu(self.conv3(x, edge_index, pseudo))

        x = gap(x, batch)

        x = F.dropout(x, self.dropout, training = self.training)

        x = F.log_softmax(self.lin1(x), dim=1)
        return x

    def training_step(self, batch, batch_idx):
      output = self(batch)
      loss = F.cross_entropy(output, batch.y)
      self.log('training_loss', loss, on_epoch=True,
on_step=False)
      self.log('train_acc_step', self.train_accuracy(output,
batch.y))
      return loss

    def on_train_epoch_end(self):
      self.log('train_acc_epoch', self.train_accuracy.compute())

    def configure_optimizers(self):
      optimizer = torch.optim.Adam(self.parameters(), lr=0.01)
      # "an initial learning rate of 0.01 and learning rate decay
to 0.001 after 200 epochs"
      scheduler = lr_scheduler.MultiStepLR(optimizer=optimizer,
milestones=[200], gamma = 0.1)
      return {
        "optimizer": optimizer,
        "lr_scheduler": scheduler
      }

    def validation_step(self, batch, batch_idx):
      output = self(batch)
      loss =  F.cross_entropy(output,  batch.y)
      self.log('val_loss', loss, on_epoch=True, on_step=False)
      self.log('val_acc_step', self.val_accuracy(output, batch.y))
```

```
      return loss

    def on_validation_epoch_end(self):
      self.log('val_acc_epoch', self.val_accuracy.compute())

    def test_step(self, batch, batch_idx):
      output = self(batch)
      loss = F.cross_entropy(output, batch.y)
      self.log('test_loss', loss, on_epoch=True, on_step=False)
      self.log('test_acc_step', self.test_accuracy(output,
batch.y))
      return loss

    def on_test_epoch_end(self):
      self.log('test_acc_epoch', self.test_accuracy.compute())

gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)

import numpy as np
from datetime import datetime
p = '/content/drive/My Drive/results/' + str(datetime.now())
+'.txt'

from google.colab import drive
drive.mount('/content/drive', force_remount=True)

!rm -r /content/checkpoints/
k_folds = 10
max_epochs = 300
results = []
tstamps = []
batch_size = 64

TT = 0.1;
Theta = 0.06;
S = 2;

# TT = 0.1;
# Theta = 0.6;
# S = 1.04;


early_stopping = False
np_random = True
kfold_random = True
```

```
# kfold = KFold(n_splits=k_folds, shuffle=kfold_random,
random_state=0)
kfold = KFold(n_splits=k_folds, shuffle=kfold_random)

dataset = TUDataset(root='data/TUDataset', name='Cuneiform',
transform=T.Compose([my_normalization()]))
augmentation_dataset = TUDataset(root='data/TUDataset',
name='Cuneiform', transform=T.Compose([my_affine_transforms(),
my_normalization()]))

# np.random.seed(12345)
model = None

for i in range(1,11):
  results.append({})
  tstamps.append([])
for fold, (train_ids, test_ids) in
enumerate(kfold.split(dataset)):
  # if fold != 0:
  #   continue
  # Print
  print(f'FOLD {fold}')
  print('--------------------------------------------------')

  tstamps[fold].append(datetime.now())

  del model
  model = Net()

  if early_stopping:
    if np_random:
      np.random.shuffle(train_ids)

    val_size = len(train_ids) // 15
    val_ids = list(train_ids[:val_size])
    train_ids = list(train_ids[val_size:])
    val_loader = DataLoader(dataset[val_ids],
batch_size=batch_size)

    dataset = TUDataset(root='data/TUDataset', name='Cuneiform',
transform=T.Compose([my_normalization()]))
    # augmentation_dataset = TUDataset(root='data/TUDataset',
name='Cuneiform', transform=T.Compose([my_affine_transforms(),
my_normalization()]))

    test_loader = DataLoader(dataset[test_ids],
batch_size=batch_size)
    train_loader = DataLoader(dataset[train_ids],
batch_size=batch_size)
```

```
    augmented_train_loader =
DataLoader(augmentation_dataset[train_ids], batch_size=batch_size)


    early_stop_callback = EarlyStopping(monitor="val_acc_epoch",
                                        min_delta=0.00,
                                        patience=1000,
                                        verbose=False,
                                        mode="max")
    checkpoint_callback = ModelCheckpoint(dirpath="checkpoints",
                                          filename="best-
checkpoint-fold-"+str(fold),
                                          save_top_k=1,
                                          verbose=False,
                                          monitor="val_acc_epoch",
                                          mode="max",
                                          every_n_epochs = 1,
                                          save_last=True)

    logger = TensorBoardLogger("lightning_logs", name="model",
default_hp_metric= False)
    trainer = pl.Trainer(gpus=1,
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                         logger=logger,
                         callbacks=[early_stop_callback,
checkpoint_callback],
                         # callbacks=[checkpoint_callback],
                         check_val_every_n_epoch=1,
                         max_epochs=max_epochs)
    trainer.fit(model=model, train_dataloaders=train_loader,
val_dataloaders=val_loader)

    checkpoint = torch.load("/content/checkpoints/best-checkpoint-
fold-"+str(fold)+".ckpt")
    model.load_state_dict(checkpoint['state_dict'])
  else:
    if np_random:
      np.random.shuffle(train_ids)

    test_loader = DataLoader(dataset[test_ids],
batch_size=batch_size)
    train_loader = DataLoader(dataset[train_ids],
batch_size=batch_size)
    augmented_train_loader = DataLoader(augmentation_dataset,
batch_size=batch_size)



    model.apply(reset_weights)

    logger = TensorBoardLogger("lightning_logs", name="model",
```

```
default_hp_metric= False)
    trainer =
pl.Trainer(gpus=1,# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!11
                         logger=logger,
                         check_val_every_n_epoch=1,
                         max_epochs=max_epochs,
                         replace_sampler_ddp=True,
                         )

    trainer.fit(model=model, train_dataloaders=train_loader)

    # trainer.fit(model=model,
train_dataloader=augmented_train_loader)
  #________________________#

  # Завантажує кращу модель у фолді, і проводить тест на ній.

  tstamps[fold].append(datetime.now())
  # results.append({})
  results[fold]['epoch'] = model.current_epoch
  results[fold]['accuracy'] = trainer.test(model=model,
dataloaders=test_loader)
  print(results[fold])
  with open(p, 'w') as f:
    f.write(str(results))
    f.write("\n\n")
    f.write(str(tstamps))
  # if fold == 0:
  #   break;

!cp -r /content/checkpoints/
/content/drive/MyDrive/Masters/checkpoints+$(date +"%Y-%m-%d-%T")
!cp -r lightning_logs/model/
/content/drive/MyDrive/Masters/lightning_logs+$(date +"%Y-%m-%d-
%T")

!nvidia-smi -L

!nvidia-smi
```