Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра Комп'ютерних інтелектуальних технологій та систем
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Інтелектуальна система побудови нейромережевих моделей

Виконав:
студент II курсу, групи КІТм-19-1
Васильєв С.О

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми Освітньо-професійна
Освітня програма
Комп'ютерні інтелектуальні технології
(повна назва освітньої програми)

Керівник: проф. Безсонов О.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри КІТС _____ Руденко О.Г.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра Комп’ютерних інтелектуальних технологій та систем

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп’ютерна інженерія

Тип програми Освітньо-професійна

Освітня програма Комп'ютерні інтелектуальні технології

(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“ ____ ” _____ 20 ___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Васильєву Сергію Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Інтелектуальна система побудови нейромережевих моделей

затверджена наказом по університету від “ 11 ” листопада 2020 р. № 1582 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 10 грудня 2020 р.

3. Вхідні дані до роботи IDE PyCharm Community Edition
Python – мова програмування

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз проблемної області і постановка задачі
Конструювання оптимізаційного алгоритму
Реалізація та тестування розробленої системи
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	11.11 – 12.11	
2	Огляд існуючих рішень	13.11 – 17.11	
2	Вибір інструментів для реалізації	18.11 – 19.11	
3	Реалізація основного функціоналу	20.11 – 02.12	
5	Конструювання та реалізація оптимізаційного	03.12 – 8.12	
	алгоритму		
6	Оформлення пояснювальної записки	9.12 – 10.12	

Дата видачі завдання 11 листопада 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Безсонов О.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 66 с., 18 рис., 3 дод., 12 джерел.

НЕЙРОННА МЕРЕЖА, ГІПЕРПАРАМЕТРИ, СЕРЕДОВИЩЕ РОЗРОБКИ, ГЕНЕТИЧНИЙ АЛГОРИТМ, AUTOML, TENSORFLOW, KERAS, .

Метою атестаційної роботи є реалізація інтелектуальної системи побудови нейромережевих моделей у вигляді середовища розробки.

У ході виконання атестаційної роботи був розроблений основний функціонал середовища розробки, що дозволяє системі бути автономною і дозволяє отримати оптимальну згорткову модель для розпізнавання зображень. Реалізує гіперпараметричу оптимізацію. Результати навчання представляються програмним продуктом у текстовому і графічному вигляді.

ABSTRACT

Master’s thesis: 66 pages, 18 figures, 3 appendices, 12 sources.

NEURAL NETWORK, HIPERPARAMETERS, DEVELOPMENT ENVIRONMENT, GENETIC ALGORITHM, AUTOML, TENSORFLOW, KERAS

The major goal of this thesis is implementation of an intelligent system for neural network models building in a development environment form.

During the certification work, the main functionality of the development environment was developed, which allows the system to be autonomous and allows to obtain an optimal convolutional model for image recognition. Implements hyperparametric optimization. Learning outcomes are presented by this software product in text and graphics.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 12
ВСТУП 13
1 РОЗГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ, ОГЛЯД ІСНУЮЧИХ ІНСТРУМЕНТІВ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ 14
1.1 Необхідний функціонал 14
1.2 Аналіз існуючих рішень і підходів 16
1.2.1 Neuroph Studio 16
1.2.2 Lobe 18
1.3 Концепція AutoML 19
1.3.1 Пошук решіткою 21
1.3.2 Випадковий пошук 23
1.3.3 Байєсівська оптимізація 24
1.3.4 Еволюційна оптимізація 26
1.3.5 Оптимізація гіперпараметрів по Івахненко 28
1.4 Вибір цільової платформи та інших технічних засобів 29
1.5 Постановка задач дослідження 31
2 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ 33
2.1 Менеджер проектів 34
2.2 Обчислювальний блок 35
2.2.1 Оптимізація гіперпараметрів мережі 37
2.2.2 Структура хромосомного відображення 39
2.2.3 Реалізація основного циклу алгоритму 45
2.2.4 Реалізація фітнес-функції для оцінки якості мережі 46
2.3 Графічний інтерфейс користувача 49
ВИСНОВКИ 55
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 56

ДОДАТОК А Використані протоколи 57
ДОДАТОК Б Реалізація основного циклу генетичного пошуку..... 59
ДОДАТОК В Перелік графічного матеріалу..... 61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГА – генетичний алгоритм

ЗНМ – згорткова нейронна мережа

GUI – graphical user interface

ВСТУП

Системи, створені людиною, які б задачі вони не виконували, стрімко розвиваються і, в цьому плані, вони йдуть шляхом ускладнення власної структури, беруть нові висоти, домагаючись все нових і нових рівнів абстракції. Те ж стосується і інструментів, за допомогою яких подібні системи розробляються.

Вірним помічником програмного розробника, не залежно від мови програмування або використовуваних технологій, завжди буде якесь середовище розробки, яке з радістю надасть кошти для пошуку програмних помилок або заощадить час розробника, надавши функціонал, подібний IntelliSense – технології компанії Microsoft для середовища розробки Microsoft Visual Studio. Згадане середовище розробки також має інструменти більш високого рівня абстракції, які дозволяють проектувати графічний користувальницький інтерфейс за допомогою інтерактивного компонування його елементів, а не замість написання програмного коду. Ще більш високий рівень абстракції має візуальне подіє-орієнтоване середовище програмування Scratch, призначене для 3D-моделювання.

Процес побудови якісної нейронної моделі часто зводиться до вельми довготривалих емпіричних експериментів по налаштуванню її гіперпараметрів, але час є найціннішим ресурсом, особливо в IT індустрії, яка відома своїм стрімким розвитком, за яким не всі в змозі встигнути. Тому, IT спільнота перебуває в постійному пошуку інструментів щодо спрощення проектувального процесу, інструментів, які дозволили б перейти від питання «як зробити?» до питання «що зробити?».

Метою даного проекту є розробка програмного продукту - середовища розробки, яка б дозволила абстрагуватися від безлічі питань у дусі «як реалізувати модель?» до питання «яку модель потрібно ралізувати?». Само собою, таке середовище розробки повинне бути зручним у використанні і мати при собі інструменти автоматичного підбору гіперпараметрів.

1 РОЗГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ, ОГЛЯД ІСНУЮЧИХ ІНСТРУМЕНТІВ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Необхідний функціонал

Середовище розробки - безсумнівно складна система, а складність її реалізації виходить далеко за рамки, які можуть бути розглянуті в межах дипломного проекту. У зв'язку з цим, прийнято рішення обмежитися реалізацією базового функціоналу, який може забезпечити автономність даної системи.

Розглянемо необхідний функціонал для майбутнього середовища розробки:

1) можливість оперувати поняттям «проект» в межах програмного продукту. Такий функціонал дозволяє об'єднати поставлену задачу і рішення воєдино, зберегти їх і, в разі потреби, повернутися до них для внесення модифікацій. Файл проекту має містити перелік вихідних файлів і інших ресурсів, і з яких система може відновити рішення. Відносно явно наукової спрямованості сфери Data Science, буде непогано, якщо проект зможе також зберігати історію експериментів і модифікацій, що само по собі може бути корисно для подальшого статистичного аналізу;

2) можливість подання моделі у текстовому вигляді мови програмування. Візуальна складова середовища розробки хоч і полегшує процес розробки, але в такому випадку отримані моделі можуть бути використані тільки в межах самого середовища розробки, що означає неможливість їх вбудовування в інші системи. Тому має місце реалізація концепції «round tripping» («туди і назад»), де система може бути змодельована візуально, програмний код буде генеруватися з отриманих моделей, а будь-які зміни коду можуть бути повернуті назад в модель;

3) середовище розробки повинне мати легко підтримувану структуру.

Подібний програмний продукт повинен мати модульну структуру для того, щоб міг бути адаптований до мінливих запитів сучасного Data Science. Більшою мірою це стосується можливості вбудовування нових типів моделей, які може реалізовувати проект середовища розробки;

4) інтерфейс користувача. Візуальне середовище програмування без графічного інтерфейсу - оксюморон, і єдине що можна було б сказати з цього пункту, це те, що він повинен бути, але виходячи з пункту 3, необхідно реалізувати можливість легко додавати нові форми для розширення функціоналу середовища. Виходячи зі специфіки застосування програмного продукту, його інтерфейс особливо не претендує на інтуїтивну зрозумілість і правило «якщо потрібно пояснювати - інтерфейс поганий» можна порушити;

5) інструменти розробки. Інструменти середовища повинні надавати можливість вирішення поставленого завдання.

Короткий список інструментів для реалізації у сфері Data Science, зокрема для нейромережевих моделей:

– сумісність з однієї або кількома бібліотеками для реалізації моделювання нижчого рівня абстракції (scikit-learn, tensorflow, scipy та ін.). Можливість абстрактно оперувати сутностями з сумісної бібліотеки;

– інструменти концепції AutoML, котрі дозволяють автоматизувати велику частину проектування аж до самостійного знаходження потрібної структури моделі для поставленого завдання;

– інструменти для перевірки і використання отриманих рішень «на ходу», що дозволить оцінити якість такого рішення;

– планувальник розрахунку проектів. Навчання моделей може займати досить тривалий час, тому хорошим інструментом може стати можливість додавати проекти в чергу на розрахунок, що дозволить зробити систему ще більш автономною і гнучкою;

– візуалізація процесу розрахунку та отриманих результатів. Залежно від поставленого завдання, тип графіків і їх кількість може варіюватися, але ясно одне - вони повинні бути. Вони дозволять зробити оцінку якості

рішення більш об'єктивною та інтерпретованою;

– логування процесу розрахунку. У ряді сценаріїв, графіки можуть володіти недостатньою інформативністю, що можна компенсувати збереженням метаданих розрахунку;

– можливість побудови довільних ансамблевих систем з наявних моделей;

– можливість розподіленого розрахунку проектів. Маючи в наявності кілька обчислювальних пристроїв має сенс розгортка клієнт-серверної архітектури для прискорення процесу розрахунку.

На даний список ми будемо спиратися в процесі аналізу існуючих рішень і власної реалізації.

1.2 Аналіз існуючих рішень і підходів

Існуючі на даний момент рішення мають далеко не весь бажаний функціонал, описаний в попередньому розділі, але в сукупності майже повністю покривають його, що зайвий раз підтверджує актуальність поставленої задачі.

1.2.1 Neuroph Studio [8]

Найбільш близькою виявилася випущена під ліцензією Apache середовище візуального програмування Neuroph Studio, розроблена на мові Java. Серед достоїнств можна відзначити:

– наявність графічного інтерфейсу;

– реалізовано безліч простих моделей;

– присутнє поняття проекту;

– відкрите джерело.

Список реалізованих моделей:

– Adaline (англ. Adaptive Linear Element) - адаптивний лінійний

елемент, що по суті є одношаровою нейронною мережею для вирішення завдання регресії.

– Perceptron - одношаровий перцептрон Розенблатта з пороговою функцією активації, який в змозі вирішити прості завдання класифікації.

– Multi Layer Perceptron with Backpropagation Momentum on Resilient Propagation – багатошаровий перцептрон (по Румельхарту);

– Hopfield network - детермінована версія класифікатора, який в процесі навчання замість послідовного наближення до потрібного стану з обчисленням помилок, всі коефіцієнти матриці розраховує по одній формулі, за одну ітерацію, після чого мережа відразу готова до роботи;

– Bidirectional Associative Memory (Двунаправлена асоціативна пам'ять або нейронна мережа Коско) – модель нейронної мережі зі зворотними зв'язками, що виділяє з зашумленних примірників еталонний образ;

– Kohonen network - нейронна мережа з навчанням без вчителя, що виконує завдання візуалізації і кластеризації;

– Hebbian network;

– Maxnet - конкурентна нейронна мережа, нейрони якої крім зв'язку «кожен з кожним» має зв'язок «сам з собою». Конкурентність полягає в тому, що в процесі навчання ми прагнемо досягти ситуації, коли в кожній фіксованій ситуації, активізується тільки один нейрон, а решта нейронів перебувають в стані спокою;

– Competittive network – конкурентна нейронна мережа для задач кластерізації, в котрій вузли змагаються за право відповідати на підмножину вхідних даних;

– Instar and outstar – сімейство мереж, заснованих на instar та outstar правилах;

– RBF network - мережа радіально-базисних функцій, що добре підходить для прогнозування часових рядів;

– Neuro Fuzzy Reasoner – гібридна інтелектуальна система, що є комбінацією штучних нейронних мереж і нечіткої логіки.

Як відзначають самі автори, Neuroph Studio - відмінне рішення для тих, хто тільки почав свій шлях у вивченні нейронних мереж, яке дозволяє продемонструвати функціонал мереж, не вдаючись у подробиці реалізації та складну теорію.

1.2.2 Lobe [9]

Лише місяць тому компанія Microsoft випустила продукт під назвою Lobe, що дозволяє отримати класифікаційну модель мало не на рівному місці, потрібно всього лише завантажити набір розмічених зображень, трохи почекати і насолоджуватися захмарною точністю розпізнавання зображень з вебкамери. На перший погляд може здатися, що цей додаток підбирає структуру моделі, грунтуючись на вхідних даних, але це не так. Насправді, Lobe має в своєму розпорядженні всього 2 моделі: ResNet-50V2 , якщо нас цікавить висока точність і MobileNetV2 для прискорення процесу розпізнавання. Не спортивне, але вельми ефективне рішення задачі полягає в тому, що ми беремо напереднавчену мережу на наборі даних Image Net (1000 класів) і замість останнього шару додаємо свій, кількість виходів якого буде дорівнювати кількості наявних в нашому завданні класів і будемо навчати тільки його. Таке рішення має досить низький час збіжності.

Серед переваг Lobe можна відзначити:

– відмінно пророблений одновіконний інтерфейс;

– проектно-орієнтована система;

– реалізована можливість експорту рішення для вбудовування в інші системи;

– реалізовано тестування отриманої моделі в двох режимах - в режимі реального часу використовуючи периферійний пристрій (вебкамеру) і за допомогою завантаження окремих зображень.

Недоліками Lobe можна вважати:

– на даний момент Lobe може впоратися тільки з завданням класифікації зображень, але за інформацією з сайту додатку, розробники планують розширити функціонал для вирішення завдань object detection і data classification;

– відмовилися від реалізації вельми корисного інструменту підбору структури моделі;

– відсутні інструменти візуалізації процесу навчання і структури самої моделі.

Опираючись на вищесказане, Lobe більш усього підійде тим, кому важливий тільки результат, без усіляких подробиць.

1.3 Концепція AutoML

Перш ніж перейти до існуючим рішенням концепції AutoML, необхідно визначити її.

Автоматичне машинне навчання (eng. AutoML) - процес автоматизації наскрізного процесу застосування машинного навчання до завдань реального світу.

AutoML можна застосувати до різних стадій вищезгаданого процесу:

– підготовка данних. Ця стадія може включати автоматичні визначення поставленого завдання (класифікація, регресія, кластеризація, ранжування і т.д), перетворення сирих даних до потрібного формату, їх очищення, а також збереження в певному форматі;

– конструювання ознак. До даного етапу крім виділення (або вибору) ознак і виявлення викидів можна також віднести метанавчання, в якому алгоритми автоматичного навчання застосовуються до метаданих про експерименти з машинним навчанням, а також перенесення навчання - можливості використання суміжної за змістом моделі для вирішення завдання;

– вибір моделі або їх сімейства. Залежно від поставленого завдання,

вихідних даних, їх якості, можна використовувати різні існуючі моделі:

а) за підходом до процесу навчання:

1) з вчителем (Supervised);

2) без вчителя (Unsupervised);

3) з частковим залученням вчителя (Semi-Supervised);

4) з підкріпленням (Reinforcement).

б) за поставленною задачею:

1) регресія (Regression) – OLS, Adaline, Regression tree;

2) класифікація (Classification) – Logistic regression, CNN, DNN, Classification tree;

3) кластерізація (Clustering) – K-Means, hierarchy, DBSCAN;

4) ранжування (Ranking) – SVM, RankNet;

5) перенос стилю - GAN;

6) виявлення викидів (Outliers detection) – Isolation Forest, KNN;

7) зниження розмірності даних та їх візуалізація (Dimension reduction) – PCA, LDA, autoencoders, t-SNE;

8) обробка природної мови (Nature language processing, NLP) – LSTM, SVM.

– оптимізація гіперпараметрів алгоритму. Гіперпараметри - емпірично підбираємі параметри системи, впливають на якість і швидкість збіжності моделі. Гіперпараметри можна розділити на два класи - пов'язані зі структурою моделі (наприклад , кількість шарів нейронної мережі) і керуючі процесом навчання (наприклад, кількість епох навчання);

– вибір метрик оцінки та процедур валідації. Існує безліч метрик, за якими можна судити про якість побудованої моделі (accuracy, f1-score, akaike, etc.), кожна з яких може бути об'єктивніше інших в певних сценаріях, тому процес вибору оціночної метрики необхідно автоматизувати. Також слід залучити процедури валідації (validation approach, cross-validation, leave-one-out) для того, щоб зробити оцінку незалежної від конкретного набору даних.

Майже всі сучасні рішення в області AutoML спрямовані в бік оптимізації гіперпараметрів мережі, які змінюються від моделі до моделі, тому для кожної моделі доводиться реалізовувати свій власний алгоритм. Однак, існують концепції оптимізаційного пошуку, загальні правила яких можна адаптувати під будь-яку модель. Розглянуто наступні найбільш популярні підходи:

– пошук по решітці;
– випадковий пошук;
– байєсівська оптимізація;
– еволюційна оптимізація;
– підхід Івахненко.

1.3.1 Пошук решіткою

Традиційним і найбільш повільним рішенням вважається так званий пошук по решітці, що по суті є повним перебором заданої підмножини кожного з варійованих гіперпараметрів і перевірці (зазвичай перехресної) на вхідному наборі даних, в результаті чого, ми можемо отримати деяку величину якості для оцінки кожного набору параметрів. На зображенні нижче (рисунок 1.1) зображено схему абстрактного пошуку решіткою.

Навіть для двох змінних (для двох гіперпараметрів) вже досить багато перевірок якості, а зі збільшенням змінних параметрів спостерігаємо експоненційне зростання кількості перевірок, що є так званим прокляттям розмірності і зводиться до вирішення комбінаторної задачі в багатовимірному просторі.

З іншого боку, пошук по решітці часто доволі легко можна розпаралелити. Це пояснюється незалежністю процесів перевірок між собою після визначення самої решітки.

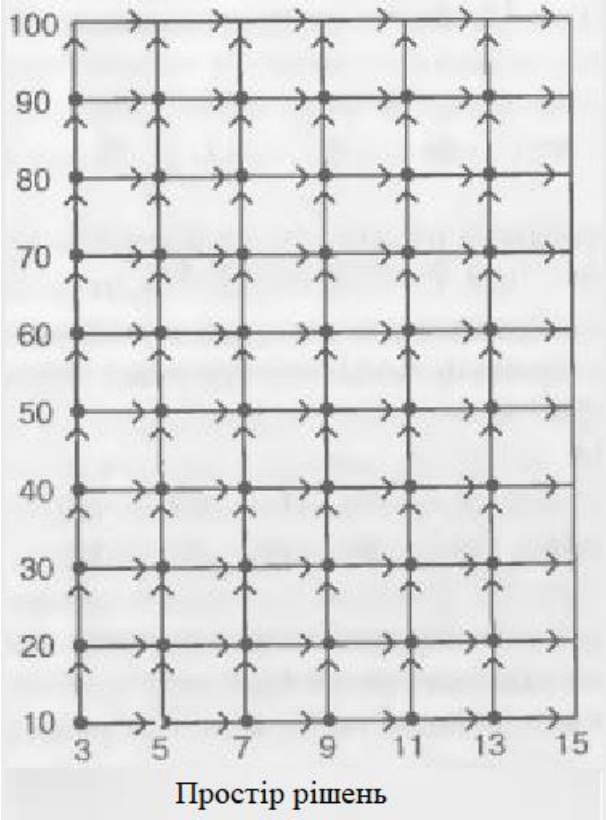

Рисунок 1.1 – Схема пошуку за решіткою для двох абстрактинх змінних

Також варто відзначити, що нам необхідно робити дискретизацію безперервних гіперпараметрів (таких як швидкість навчання) вручну, тобто задати кінцевий набір «прийнятних» значень з області визначення гіперпараметра (для швидкості навчання зазвичай (0, 1)). Це змушує нас шукати компроміс між меншим часом на пошук і більшою ймовірністю збіжності.

Переваги й недоліки:

– легко розпаралелюється;

– гарантовано знайде кращу комбінацію з запропонованої вибірки;

– маємо повний контроль над значеннями, що можуть приймати гіперпараметри;

– вимагає значної обчислювальної здатності через прокляття розмірності ;

– необхідність попередньої дискретизації безперервних параметрів і існує ймовірність промахнутися зі значеннями.

– реалізації пошуку по решітці:

– LIBSVM – модуль для здійснення пошуку решіткою;

– Scikit0learn – це пакет мовою Python, котрий включає пошук решіткою;

– Talos – містить пошук решіткою для пакету Keras.

1.3.2 Випадковий пошук

Є модифікацією пошуку по решітці з метою знизити вплив прокляття розмірності. Повний перебір всіх комбінацій в даному випадку замінюється на вибірку їх випадковим чином. При такому підході необхідно вибрати критерій зупинки, який ми вважаємо оптимальним. Підхід дозволяє знайти оптимальне рішення швидше пошуку по решітці тільки в тому випадку, якщо таке рішення існує для заданої підмножини гіперпараметрів. Нижче наведена візуальна інтерпретація метода випадкового пошуку (рисунок 1.2).

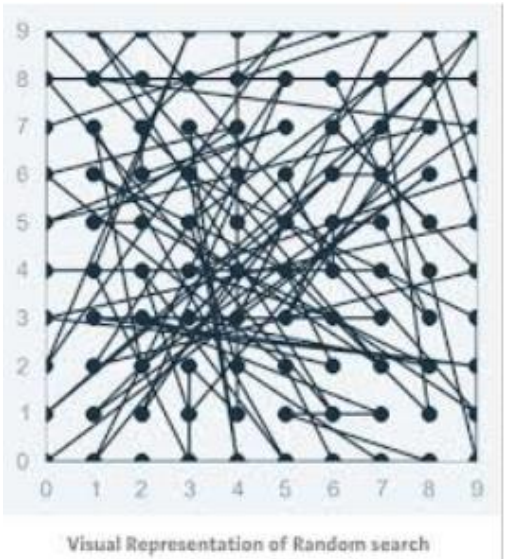

Рисунок 1.2 – Візуальна інтерпретація випадкового пошуку

Переваги й недоліки:

– можна легко розпаралелити;

– зазвичай потребує меншого часу на обчислення ніж звичайний

пошук решіткою;

– маємо повний контроль над значеннями, що можуть приймати гіперпараметри;

– при знаходженні першого рішення, що задовольняє критерію зупинки, процес пошуку зупиняється, а це значить, що більш кращі рішення можуть залишитися за кадром;

– у разі відсутності оптимального рішення, час повного перебору може перевищити час повного перебору по решітці;

– необхідність попередньої дискретизації безперервних параметрів.

Реалізації:

– hyperopt через hypera та hyperopt-sklearn – це пакети мовою Python, котрі містять випадковий пошук;

– Scikit-learn – це пакет мовою Python, що виконує випадковий пошук;

– H2O AutoML – забезпечує автоматичну підготовку даних, налаштування гіперпараметрів випадковим пошуком та багаторівневі збірки в розподіленій платформі навчання машин;

– Talos – містить випадковий пошук для Keras, що дозволяє налаштувати модель.

1.3.3 Байєсівська оптимізація

Це метод глобальної оптимізації для невідомої функції з шумом.

Байєсівські підходи, на відміну від пошуку по решітці і його модифікації – випадкового пошуку, відстежує результати минулих оцінок. Оцінюючи гіперпараметри, які здаються більш багатообіцяючими з минулих результатів, байєсовські методи можуть знаходити найкращі налаштування моделі, ніж випадковий пошук, за меншу кількість ітерацій.

Застосована до гіперпараметрічної оптимізації байєсівська оптимізація будує стохастичну модель функції відображення зі значень гіперпараметра в цільову функцію, застосовану на безлічі перевірок. Шляхом ітеративного

застосування перспективної конфігурації гіперпараметрів, заснованої на поточної моделі, а потім її оновлення, байєсівська оптимізація прагне зібрати якомога більше інформації про цю функцію і, зокрема, місце оптимуму.

В даному випадку, крім критерію зупинки, необхідно визначити функцію, що приймає значення гіперпараметрів і виводить якусь оцінку, яку ми хочемо максимізувати або мінімізувати, а також критерій, званий функцією вибору, для оцінки того які гіперпараметри варто вибирати наступними (рисунок 1.3).

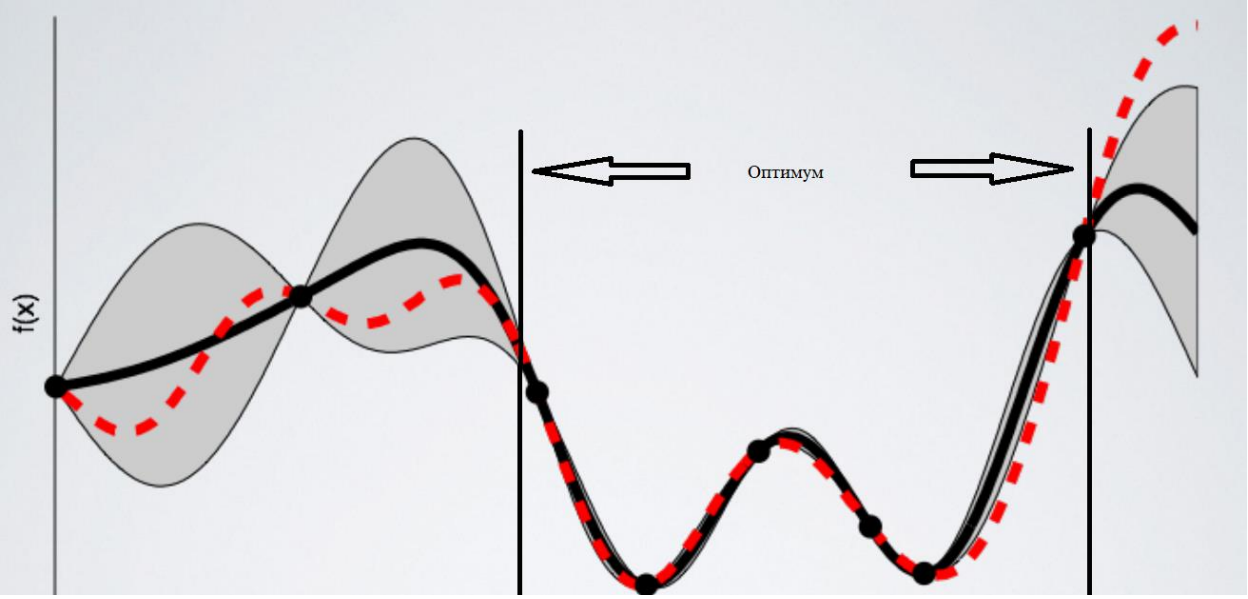

Рисунок 1.3 - Графік апроксимації (чорний) функції (червоний), яка описує ідеальний кортеж гіперпараметрів

На практиці байєсівська оптимізація показала кращі результати з меншою кількістю обчислень в порівнянні з пошуком по решітці і випадковим пошуком через можливість судження про якість експериментів ще до їх виконання.

Переваги і недоліки:

– більш швидка збіжність;

– якість оптимізаційного пошуку безпосередньо залежить від якості

побудованої нами моделі.

Реалізації баєсівської оптимізації:

– Spearmint – це пакет для байесівської оптимізації алгоритмів навчання машин;

– Bayesopt – ефективна імплементація байєсівської оптимізації на C/C++ з підтримкою Python, Matlab і Octave;

– MOE – це бібліотека для Python, C++ і системи паралельних обчислень CUDA, що імплементує Ба'єсову глбальну оптимізацію, використовуючи гаусові процеси;

– Auto-WEKA – це рівень для байєсівської оптимізації поверх WEKA;

– Auto-sklearn – це рівень для байєсівської оптимізації поверх scikit-learn;

– mlrMBO с mlr – це пакет мовою R для байєсівскьої оптимізації або для оптимізації на основі моделі невідомої функції (чорний ящик);

– tuneRanger – це пакет мовою R для налаштування випадкових лісів використовуючи оптимізацію на базі моделі;

– BOCS – це пакет Matlab, що використовує напіввизначене програмування для мінімізації невідомої функції при дискретних вхідних даних. Є також імплементація для Python 3;

– SMAC – це бібліотека мовами Python/Java, що імплементує байєсівську оптимізацію.

1.3.4 Еволюційна оптимізація

Це також метод глобальної оптимізації для невідомої функції з шумом. В процесі оптимізації використовуються так звані генетичні алгоритми (ГА [11]), що були навіяні біологічною концепцією еволюції:

1) створюється початкова популяція рішень випадковим чином, зазвичай 100+;

2) оцінюємо дані рішення за допомогою фітнес-функції (функції

пристосованості);

3) сортування і ранжування популяції, відповідно до фітнес-функції, від кращого рішення до гіршого;

4) заміна гіршої частини рішень новими за допомогою операторів схрещування та мутації;

5) повторюємо кроки 2-4 поки не досягнемо оптимуму або не переконаємося, що такого немає.

Зображення нижче (рисунок 1.4) демонструє схему роботи цього алгоритму.

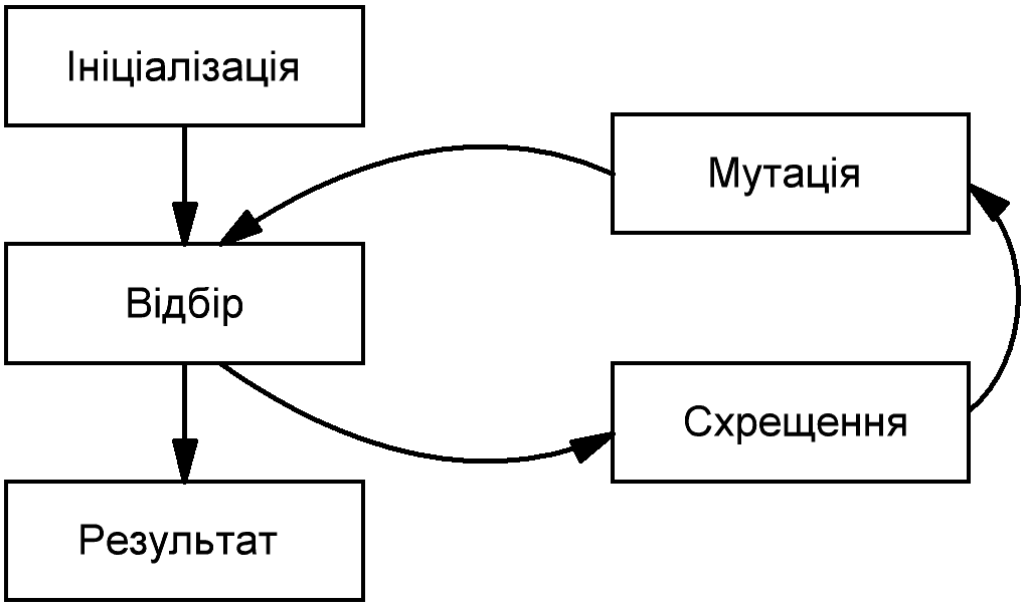

Рисунок 1.4 – Схема роботи генетичного алгоритму оптимізації

Переваги і недоліки:

– алгоритм зберігає краще рішення на кожній епосі до тих пір, поки його не перевершать, що не дозволяє алгоритму розходитися;

– збіжність швидше пошуку по решітці і його модифікації у вигляді випадкового пошуку.

Реалізації, що існують:

– DEvol – пакет (бібліотека), розроблена на мові Python, який здійснює

пошук архітектури глибокої нейронної мережі за допомогою генетичних алгоритмів. Самі моделі визначаються засобами бібліотеки keras;

– DEAP – це гнучкий Python фреймворк для загальних еволюційних обчислень, об'єднаний з пакетами розпаралелювання, такими як scoop і pyspark і Python фреймворками, на зразок scikit-learn через sklearn-deap;

– TPOT – пакет, ім'я якого розшифровується як Tree-based Pipeline Optimization Tool. TPOT – це інструмент автоматизованого машинного навчання на Python, який оптимізує ансамблеві моделі з використанням генетичного програмування.

Недоліком, який об'єднує перераховані вище рішення, є відсутність графічного інтерфейсу для зручного моніторингу процесу.

1.3.5 Оптимізація гіперпараметрів по Івахненко

Хотілося ще згадати підхід, описаний в роботах радянського вченого А.Г. Івахненко (середина 1960-х), який він застосовував для побудови так званих глибоких нейронних мереж. Метод групового урахування параметрів, як він сам його називав, полягав у наступному:

1) спочатку ми вибираємо загальний вигляд, параметричне сімейство моделей, які будемо навчати; Івахненко пропонував використовувати так звані поліноми Колмогорова-Габора, тобто по суті просто многочлени з невідомими коефіцієнтами, але можуть бути і будь-які інші;

2) будуємо і навчаємо різні варіанти обраних моделей;

3) обираємо за допомогою метрики якості кілька кращих моделей; якщо потрібну якість вже досягнуто, зупиняємося;

4) але якщо ще не досягнуто (і це ключовий момент) то ми починаємо будувати моделі наступного рівня, використовуючи виходи підібраних на попередньому кроці моделей як входи для наступних;

5) цей процес можна рекурсивно повторювати до тих пір, поки якість моделі або не досягне потрібного рівня, або не перестане поліпшуватися.

Зображення, наведене нижче (рисунок 1.5), демеонструє цю схему роботи.

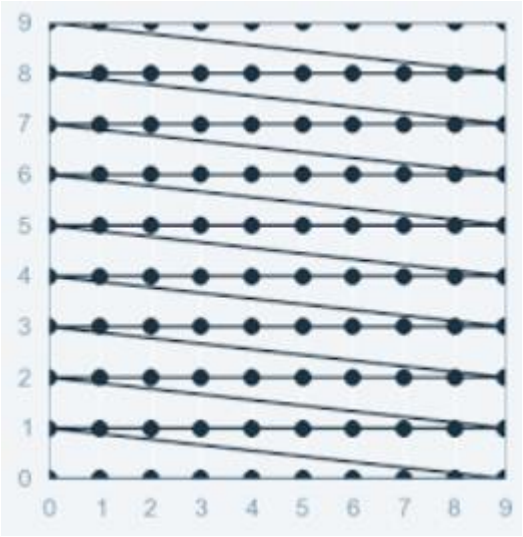

Рисунок 1.5 – Схема роботи методу групового обліку параметрів Івахненко

Як стверджують автори [1], підхід Івахненко виглядає напрочуд сучасно. Простежується подібність з описаним вище алгоритмом пошуку по решітці. Різниця полягає в явному русі від простої структури моделі до складної і акцент тут зроблений в основному на підборі глибини цієї мережі. З причини сучасності алгоритму, можна віддати данину поваги і спробувати застосувати такий підхід в узагальненому вигляді. На жаль, даний підхід має ті ж вади, що й пошук по решітці.

2.2 Вибір цільової платформи та інших технічних засобів

Розробка подібної системи під мобільні платформи видається неможливою, тому що останні на даний момент не в змозі надати належну обчислювальну потужність для коректної роботи процесу навчання нейронних моделей. Але для ARM ще не все втрачено - можлива реалізація клієнт-серверного додатка, тільки ось серверна частина все одно повинна розташовуватися на базі архітектури x86-64, бо для поставленого завдання у

ARM / x86 просто не вистачить пам'яті (за винятком окремих серверних реалізацій ARM). Однак навіть такий підхід дуже незручний, тому що кожному користувачеві потрібно буде мати свій власний і постійно працюючий сервер, який би зміг посилати результати розрахунків по мережі на клієнт. Виходячи з вищесказаного, мобільний клієнтський додаток варто розглядати тільки в якості додаткової модифікації.

Можливо, також варто звернути увагу на те, як це реалізовано в інших середовищах розробки. В переважній більшості, середовище розробки (такі як PyCharm або Visual Studio) представлене у вигляді десктопного додатка і повинно покривати більшість операційних систем, доступних для архітектури x86-64, а саме:

– Windows;
– Linux;
– MacOS.

Також необхідно визначитися з мовою програмування, яка буде використовуватися в процесі розробки. Високорівнева мова програмування Python на даний момент є поза конкуренцією, тому що він має найбільшу кількість бібліотек в своєму арсеналі для роботи з нейронними мережами та іншими моделями. Плюс, має прекрасний інструмент для розробки кроссплатформних десктопних додатків - PyQt, що дозволяє не замислюватися про вибір цільової платформи.

Серед бібліотек, які будують граф обчислень нейронної моделі, лідерами можна вважати Tensorflow і Theano, що підтримують розрахунок на графічних процесорах, а також бібліотека Keras, яка надає зручний інтерфейс для роботи з бібліотеками, згаданими раніше в цьому реченні. Також варто згадати, що у Keras є проблеми з її інтеграцією в багатопотокові програми, що не є добре, тому що розробка зручного графічного інтерфейсу передбачає використання багатопоточності. На цей рахунок, у Tensorflow є своя власна вбудована реалізація бібліотеки Keras, що позбавлена такого недоліку.

1.5 Постановка задач дослідження

В результаті аналізу джерел присвячених дослідженням в області проектування нейронних мереж та моделей ML вцілому, можна зробити наступні висновки.

По-перше, судячи з проаналізованих рішень, дослідження в цій області все ж таки ведуться. Спільнота розробників у сфері Data Science вживає спроби полегшити процес розробки моделей на всіх його етапах, котрі описуються концепцією AutoML. Слід також зауважити, що більш плідними на данний момент є дослідження автоматизації проектування статистичних моделей (регресії, методах заснованих на деревах рішень та ін.), реалізація котрих представлена у вигляді бібліотеки scikit-learn, яка також надає інструменти концепції AutoML для всіх етапів проетування.

По-друге, гіперпараметрична оптимізація нейронних мереж – складне ресурсомістке завдання, що потребує ретельного аналізу кожного з гіперпараметрів на предмет доцільності його варіювання задля зменшення обчислювальної складності оптимізаційного алгоритму

По-третє, у першу чергу, найбільш важливим є реалізація модульної структури середовища, що дозволила б у майбутньому легко додавати будь-які інструменти, що дозволить реалізувати повний цикл AutoML для будь-якої моделі і розширити сферу свого застосування для вирішення більшої кількості задач. Це, в свою чергу, робить данний проект проектом, який слід підтримувати.

Метою данної роботи є розробка середовища розробки нейромережевих моделей та додаткових інструментів роботи з такими моделями. Це середовище надасть можливість з легкістю створювати моделі для їх вбудови у більш складні системи, а також проводити статистичні дослідження впливу вхідних даних на структуру моделі.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– реалізація зручного інтерфейсу – основи середовища;

– реалізація основних ланцюгів концепції AutoML, а саме – підготовка вхідних даних, навчання конкретної моделі, та автоматичне оцінювання якості моделі після начання;

– дослідження гіперпараметричної оптимізації на предмет впливу гіперпатаметрів або їх поєднання на кінцевий результат, вибір оптимального кортежу гіперпараметрів для варіювання. Реалізація оптимізації одним з наведених у цьому розділі способів.

2 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

Для забезпечення комфортності, розробка буде проводиться в інтегрованому середовищі розробки PyCharm 2019 Community Edition міжнародної компанії JetBrains, а також буде використовуватися вбудована в неї система контролю версій з віддаленим репозиторієм на GitHub [10]. Система контролю версій дозволить поетапно вносити будь-які зміни в архітектуру програми і допоможе підтримувати програмний продукт тривалий час.

Було прийнято рішення розділити проект на три в достатній мірі незалежних архітектурних блоки:

– менеджер проектів;
– розрахунковий блок;
– інтерфейс користувача.

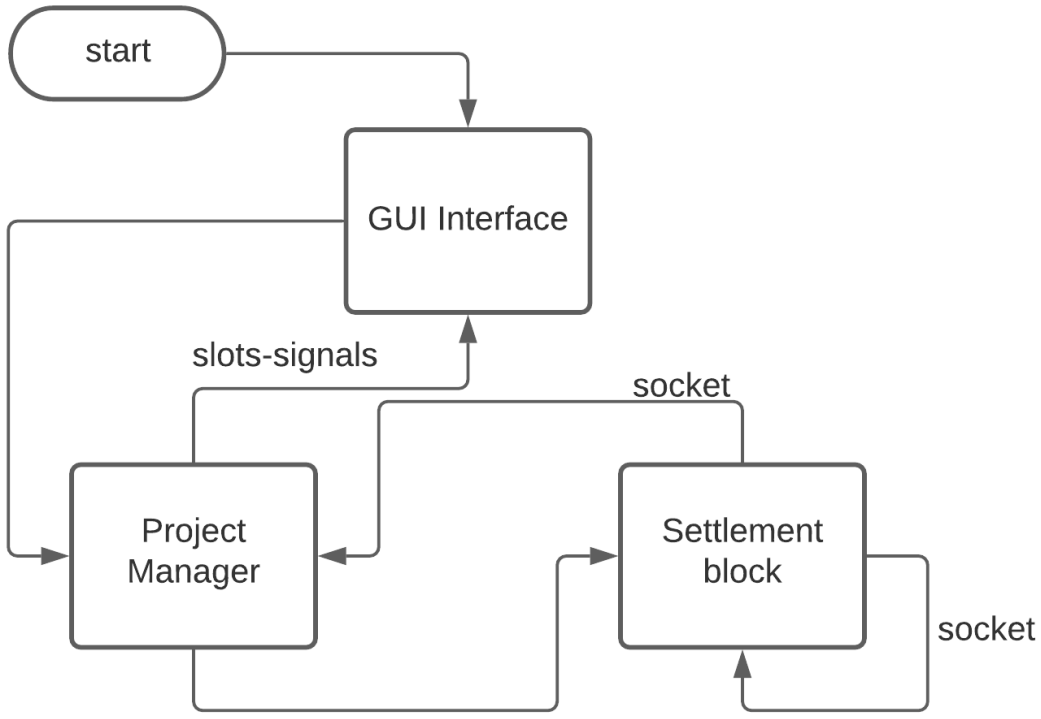

Рисунок 2.1 – Спрощена схема складових архітектурних блоків середовища

2.1 Менеджер проектів

Середовище розробки повинне мати можливість оперувати поняттям проекту в межах сесії свого виконання. Даний функціонал найлогічніше реалізувати у вигляді окремого класу (Project_controller), який буде містити в собі поля для зберігання станів проекту, історію і методи спілкування з іншими структурними блоками. Щодо поточної реалізації програмного продукту, при створенні екземпляра класу Project_controller, йому необхідно передати:

– mode: int – протокол, що дозволяє задати або ідентифікувати тип проекту. Протокол описаний в додатку А;

– is_open: bool = False – флаг, який визначає, створюємо ми новий проект або відкриваємо серіалізований раніше;

– project_name: str – Назва проекту, яке буде відображатися в інтерфейсі. Назва директорії, в якій будуть зберігатися серіалізірованние дані проекту збігається з назвою проекту;

– project_path: str – шлях до директорії, в якій зберігаються створені проекти, за замовчуванням береться шлях, вказаний в налаштуваннях програми;

– project_path + “/” + project_name – абсолютний шлях до директорії проекту.

Спілкування проекту з обчислювальними складовими програми було вирішено реалізувати через сокети, тобто за фактом розгорнути клієнт-серверну архітектуру, що дозволить в майбутньому домогтися масштабованості і виконувати розподілені обчислення за наявності декількох обчислювальних пристроїв. Спілкування ж з графічним інтерфейсом, в поточній версії програми, реалізовано через слот-сигнал систему, яка надається бібліотекою PyQt v5.

2.2 Обчислювальний блок

Основним функціоналом даного архітектурного блоку є безпосередньо навчання нейронної моделі. Клас, який буде будувати і навчати модель має бути досить універсальний і мати можливість додавати абсолютно будь-які структурні елементи нейронної мережі для реалізації моделей будь-якої складності і будь-якого призначення. При створенні об'єкта Network передаються такі аргументи:

– network_params: NetworkParams – описовий клас нейронної мережі, що зберігає в собі параметри шарів для створення послідовної моделей засобами tensorflow.keras;

– pc_port: int – ідентифікатор порту для спілкування з об'єктом класу Project_controller;

– opt_port: int – ідентифікатор порту для спілкування з об'єктом класу GeneticProgram, який реалізує генетичний оптимізаційний пошук;

– project_path: str – абсолютний шлях до директорії проекту, який став ініціатором даного розрахунку. Даний параметр необхідний для можливості сериализации навченої моделі засобами tensorflow.

Приклад методу створення послідовної моделі для класифікації зображень з екземпляра класу NetworkParams наведено нижче.

```
def createConvModel(self):
    modelSeq = Sequential()
    modelSeq.add(ZeroPadding2D((1, 1), input_shape=(128, 128, 3)))
    for filters, kernel, cActIndex, cDropout, maxpool in zip(self.network.filters,
                self.network.kernels, self.network.cActIndexes,
self.network.cDropouts,
                self.network.maxPools):
        activation = self.network.cActivations[cActIndex]
        modelSeq.add(
            Conv2D(filters=int(filters), kernel_size=kernel, strides=(1, 1),
                   padding='same', activation=activation))
        if cDropout != 0: modelSeq.add(Dropout(float(cDropout) / 100))
```

```
    if maxpool: modelSeq.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

modelSeq.add(Flatten())
for neurons, dActIndex, dDropout in zip(self.network.neurons,
                                        self.network.dActIndexes, self.network.dDropouts):
    activation = self.network.dActivations[dActIndex]
    modelSeq.add(Dense(units=int(neurons), activation=activation))
    if dDropout != 0: modelSeq.add(Dropout(float(dDropout) / 100))

modelSeq.add(Dense(units=self.network.outputs_numb, activation='softmax'))
return modelSeq
```

Приклад 2.1 – Метод створення згорткової моделі

В результаті навчання класифікаційної мережі ми отримаємо кількість її параметрів (вагових коефіцієнтів) і словник метрик оцінки (accuracy, f1-score, precision, recall), отриманий в результаті роботи методу sklearn.metrics.classification_report(). Кількість параметрів і метрики через сокет повернуться в об'єкт класу Project_controller. Спрощена схема взаємодії об'єктів класів Project_controller і Network представлена на рисунку 2.2.

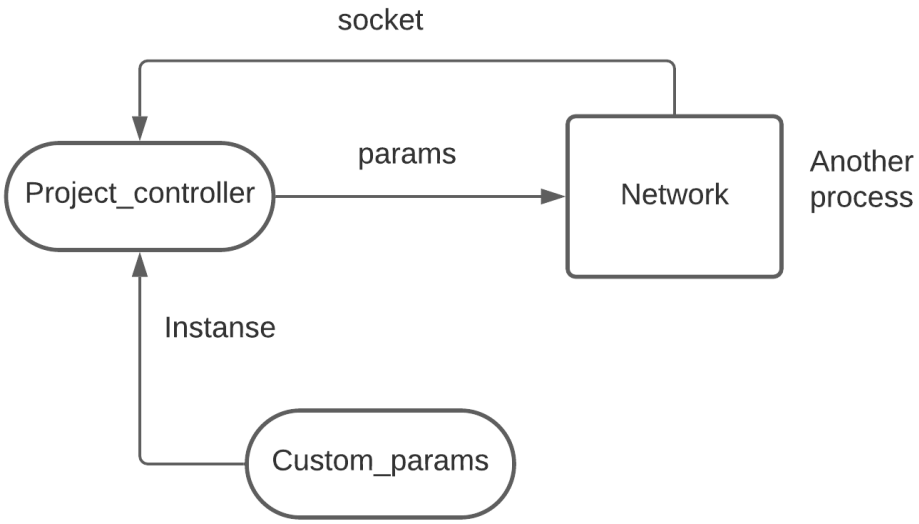

Рисунок 2.2 – Схема взаємодії Project_controller – Network

Окремо варто відзначити, що розрахунок нейронної мережі відбувається в окремому процесі, взаємодія з яким реалізовано мережевими засобами, що в майбутньому дозволить зробити даний проект масштабованим і створити на його базі розподілену систему розрахунку.

2.2.1 Оптимізація гіперпараметрів мережі

Перш ніж приступити до реалізації будь-якого підходу, необхідно визначитися зі списком гіперпараметрів, які ми будемо варіювати. Як стверджує автор [12], конструювання генетичного алгоритму в конжному конкретному випадку – вельми творчий процес, що дозволяє відходити від загальноприйнятої його складових. Для прикладу, візьмемо згорткову модель для розпізнавання зображень. Серед її гіперпараметрів можна виділити:

– параметри згорткових;
– параметри шарів перцептрону;
– Dropout;
– функція втрат;
– оптимізатор;
– швидкість навчання.

Розглянемо їх докладніше:

а) параметри згорткових шарів. Згорткові шари дозволяють витягати ознаки із зображень і узагальнювати їх. Саме вони з окремих пікселів роблять патерни від прямої лінії до цілого образу, але на відміну від повнозв'язних, ознаки зберігають своє просторове розташування. Серед змінних гіперпараметрів можна виділити:

1) кількість шарів. Від цього параметра безпосередньо залежить наскільки складними будуть вихідні карти ознак. Залежно від вхідних даних може знадобитися різна кількість шарів. Якщо шарів недостатньо мережа не зможе виділити найбільш значущі ознаки, властиві даному зображенню.

Іншою крайністю є той випадок, коли шарів занадто багато, що призводить не тільки до збільшення розміру мережі, але і збільшує шанс перенавчання;

2) кількість фільтрів. Визначає кількість вихідних карт ознак шару. Для кожного шару цей параметр має свою величину. Необхідно дотримуватися тенденції - кількість фільтрів має збільшуватися від шару до шару;

3) розмір ядра згортки. Для завдання розпізнавання зображення має два значення - ширину і висоту, що визначають розмір ковзаючого вікна;

4) Функція активації. Визначає собою спосіб введення нелінійності обчислень і дозволяє вирішувати нетривіальні завдання. Залежно від складності використовуваної функції, вона може або прискорити, або уповільнити процес навчання через необхідність розрахунку похідних цієї функції при процедурі зворотного поширення помилки;

б) параметри шарів перцептрону. Повнозв'язні шари обробляють розгорнуті карти ознак та дозволяють, у разі класифікаційної задачі, узагальнити їх до кортежу ймовірностей класів. Серед змінних гіперпараметрів можна виділити:

1) кількість шарів. Дозволяє групувати нейрони в групи і застосовувати для кожної свою функцію активації. Чим більше шарів - тим більше лінійних комбінацій нелінійних виходів ми можемо обробити, що позитивно впливає на запам'ятовування ознак, однак з їх збільшенням сильно зростає обсяг споживаної мережею пам'яті і зростає шанс перенавчання;

2) кількість нейронів. На відміну від фільтрів згорткових шарів, нейрони мають зв'язок типу «кожен з кожним» і тому зазвичай більша частина займаної мережею пам'яті припадає саме на них. У купі з кількістю шарів визначає обчислювальну потужність мережі, занадто мало - мережа буде не в змозі що-небудь запам'ятати, занадто багато - доб'ємося перенавчання;

3) функція активації. До вже сказаного про функції активації в згорткових шарах можна додати, що функція вихідного шару мережі повинна перетворювати вхідний сигнал в значення, які можна інтерпретувати як ймовірності класів;

4) наявність Dropout-шару і його відсоток. Цей параметр свій для кожного з шарів, при чому як для згорткового, так і для шарів перцептрону. Обнуляє вказаний відсоток вагових коефіцієнтів шару. Дозволяє уникнути перенавчання, проте в разі високого відсотка, мережа буде не в змозі чомусь навчитися;

в) функція втрат. Вона використовується для розрахунку помилки між реальними і отриманими відповідями. Наша глобальна мета - мінімізувати цю помилку. Таким чином, функція втрат ефективно наближає навчання нейронної мережі до цієї мети;

г) оптимізатор. Являє собою метод коригування вагових коефіцієнтів. Зазвичай використовується SGD-оптимізатор (Stochastic Gradient Descent), але існують різні модифікації, які заважають нам застрявати в локальних мінімумах;

д) швидкість навчання. Гіперпараметр, від якого безпосередньо залежить час навчання. Як стверджує автор [2], також буде корисним реалізувати сутність, яка автоматично:

– зменшить швидкість навчання, якщо втрати зростають або осцилюють в широкому діапазоні;

– збільшить швидкість навчання, якщо втрати постійно зменшуються, але дуже повільно.

2.2.2 Структура хромосомного відображення.

Головним завданням при реалізації даного алгоритму стосовно нейронних моделей - побудувати структуру, яка буде описувати гіперпараметри в термінах генетичної оптимізації. Дана структура і буде

хромосомою. Зважаючи на складність її організації, буде розумним розділити одну сутність на кілька сутностей і встановити ієрархічну залежність між ними, що, в свою чергу, дозволить використовувати їх повторно в інших типах моделей. Розглянемо її докладніше, починаючи з нижнього рівня ієрархії. Шар мережі є найпростішим і неподільним будівельним блоком будь-нейронної мережі. Згорткова мережа має два основних типи шарів, які задають її структуру - згорткові і повнозв'язні (Conv2D і Dense шари). Також необхідно визначити сутність, які будуть зберігати безліч прийнятних значень гіперпараметрів, які можуть приймати ті чи інші поля шарів. Розглянемо генетичну реалізацію класів, перерахованих вище шарів.

Клас C2D_RandomParams зберігає діапазони прийнятих значень гіперпараметров згорткової частини мережі. D2D_RandomParams виконує ту ж функцію для повнозв'язних шарів. На вхід вони отримують ці самі діапазони, які можна налаштовувати в ході виконання програми використовуючи графічний інтерфейс.

C2d_Layer на вхід конструктора приймає:

– c2d_rp: C2D_RandomParams – діапазони для згорткових шарів;

– filters: int – кількість фільтрів для даного шару. Число фільтрів для кожного шару визначається на рівень вище за ієрархією структури хромосоми, тому що існує висхідна залежність від шару до шару.

Список основних змінних класу:

– actIndex: int – індекс функції активації для отримання певної функції з колекції функцій активації. Змінна представлена в такому вигляді, тому що в даному випадку його легко змінювати випадковим чином. В ході роботи алгоритму може приймати число в діапазоні довжини списку обраних функцій активації;

– dropoutRate: int – визначає, чи буде за згортковим шаром Dropout-шар, а також те, яка частина (%) зв'язків мережі буде обнулена;

– kernel: list – список, що визначає розмір ядра згортки для даного шару. Також має флаг squareKernel, який змушує ядро приймати тільки квадратну форму.

D2d_Layer на вхід конструктора приймає:

– d2d_rp: D2D_RandomParams – діапазони для повнозв'язних шарів.

– neurons: int – кількість нейронів для даного шару. Схожий на параметр filters для згорткового шару, тільки має спадну залежність від шару до шару;

Змінні класу ідентичні до змінних класу згорткового шару, за винятком відсутності змінної kernel.

Також дані класи рівнів передбачають методи мутації для згаданих змінних.

На другому рівні ієрархії окремі шари перетворюються в структури шарів, які визначаються класами C2D_Structure і D2D_Structure і приймають на вхід відповідні об'єкти, що описують діапазони прийнятих значень гіперпараметрів. У кожному з класів, існує список, що містить відповідні екземпляри шарів, а також флаги для управління залежностями між даними шарами. Приклад 2.2 демонструє ініціалізацію змінних згорткової структури, яка відбувається випадковим чином, втім, як і будь-яка інша ініціалізація з даної ієрархічної структури.

```
self.layersNumb = self.sr.randrange(1, c2d_rp.layersRange)
self.layers = []

absorber = 0
if self.sameActivation: absorber = self.sr.randrange(c2d_rp.actIndexRange)
powIndex = 0
for i in range(self.layersNumb):
    if i == 0:
        powIndex = self.sr.randrange(c2d_rp.fPowRange[0], c2d_rp.fPowRange[1]+1)
    else:
        powIndex += self.sr.randrange(2)
    filters = round(math.pow(2, powIndex))
```

```
    self.layers.append(C2dLayer(c2d_rp, filters))
    if self.sameActivation: self.layers[i].actIndex = absorber
    if self.squaredKernels:
        index = self.sr.randrange(2)
        if index == 0:
            self.layers[i].kernel[1] = self.layers[i].kernel[0]
        else:
            self.layers[i].kernel[0] = self.layers[i].kernel[1]
        self.layers[i].squareKernel = True
if self.sameKernel:
    absorber = self.sr.randrange(len(self.layers))
    for i in range(len(self.layers)): self.layers[i].kernel =
self.layers[absorber].kernel
```

Приклад 2.2 – Ініціалізація згорткової структури

Дані класи також реалізують методи мутацій для даного рівня ієрархії, а саме:

– мутацію кількості фільтрів і нейронів відповідно, зі збереженням залежності між шарами;

– мутацію кількості шарів;

– мутацію прапорів, керуючих залежностями між шарами.

Дані мутації також викликають методи мутації кожного шару, тобто мутації, що знаходяться на рівень нижче за ієрархією.

Клас сверточное хромосоми C2d_Chromosome, який успадковується від абстрактного класу Chromosome займає третій і останній рівень ієрархічної структури. Приймає в себе структуру, яка крім відомих нам C2D_RandomParams і D2D_RandomParams містить екземпляр класу Network_RandomParams, який зберігає в собі діапазони для гіперпараметров навчання мережі (оптимізатор, функція втрат, кількість епох і швидкість навчання, список застосовуваних callback-ів, а також абсолютний шлях до набору тренувальних даних), а також параметри генетичної еволюції, такі як розмір популяції, кількість епох еволюції, шанс мутації, розподіл оператора

відбору а також ідентифікатор застосовуваної фінтесс-функції. Зберігає в собі такі змінні:

– conv_part: C2D_Structure – згорткова частина мережі;

– dense_part: D2D_Structure – повнозв’язна частина мережі;

– assessment: float – значення фітнес-функції;

– params_count: int – кількість параметрів (вагових коефіцієнтів) мережі;

– report: dict – словник оціночних метрик нейронної мережі;

– name: int – ідентифікатор хромосоми;

– optimizer: int – індекс оптимізатора мережі з колекції обраних оптимізаторів;

– loss_func: int – індекс функції втрат з колекції обраних функцій втрат, яка буде використовуватися для розрахунку помилки в процесі навчання мережі;

– lr: float – початкова швидкість навчання.

Також реалізований метод мутації, який, крім запуску мутацій нижнього рівня для згорткової і повнозв’язної частин, реалізує мутації інших своїх змінних. Також метод мутації поточного рівня ієрархії повертає логічне значення, що дозволяє зрозуміти, відбулася яка-небудь мутація чи ні. Такий підхід дозволяє уникнути повторного навчання у тому разі, якщо конфігурація мережі не змінилася. Реалізації деяких мутацій представлені в прикладі 2.3.

```
def mutate(self, mutateRate):
    is_mutateC2D = self.mutateCPart(mutateRate)
    is_mutateD2D = self.mutateDPart(mutateRate)
    is_mutated_self = self.mutate_self(mutateRate)
    if is_mutateC2D or is_mutateD2D or is_mutated_self:
        return True
    return False

def mutateCPart(self, mutateRate):
    if not self.sr.randrange(100) < mutateRate: return 0
```

```
    if self.sr.randrange(100) < 10:
        self.c2d_Part = C2dStructure(self.chr_p.c2d_rp)
    else:
        self.c2d_Part.mutateFilters(mutateRate)
        self.c2d_Part.mutateLayersNumb(mutateRate)
        self.c2d_Part.mutateActivations(mutateRate)
        self.c2d_Part.mutateDropouts(mutateRate)
        self.c2d_Part.mutateKernels(mutateRate)
    return 1

def mutateDPart(self, mutateRate):
    if not self.sr.randrange(100) < mutateRate: return 0
    if self.sr.randrange(100) < 10:
        self.d2d_Part = D2dStructure(self.chr_p.nrp, self.chr_p.d2d_rp)
    else:
        self.d2d_Part.mutateDropouts(mutateRate)
        self.d2d_Part.mutateActivations(mutateRate)
        self.d2d_Part.mutateLayerNumb(mutateRate)
        self.d2d_Part.mutateNeurons(mutateRate)
    return 1
```

Приклад 2.3 – Мутації останнього рівня

Також реалізовані методи to_str() (для відображення конфігурації мережі в текстовому вигляді) і to_json () (для подальшої серіалізації).

З огляду на і без того достатній рівень складності представленої структури, для даної версії програми було прийнято рішення відкласти реалізацію оператора кросовера до майбутніх версій. У поточній версії, обов'язки оператора схрещування виконує оператор мутації. Спрощена схема ієрархічної структури представлена на рисунку 2.3

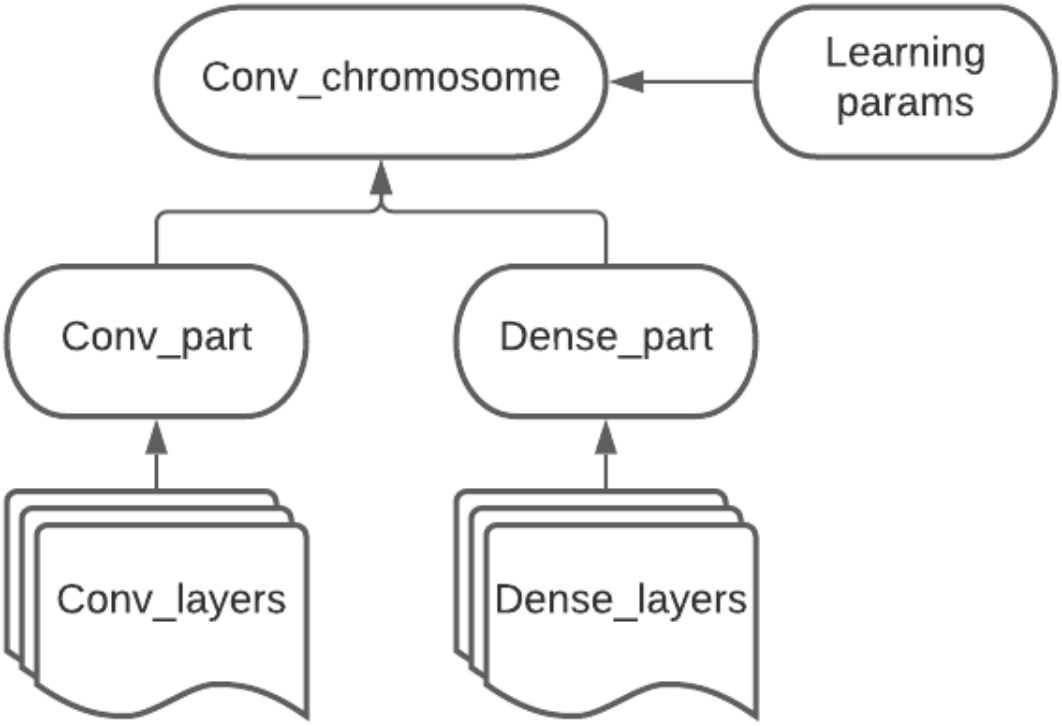

Рисунок 2.3 – Ієрархічна схема структури

Також варто згадати, що елемент випадковості для реалізованого генетичного алгоритму надає генератор псевдовипадкових послідовностей SystemRandom() зі стандартного пакета random, який в своїх обчисленнях використовує системну ентропію обчислювального пристрою, що допомагає уникнути повторень згенерованих значень.

2.2.3 Реалізація основного циклу алгоритму.

Схема алгоритму, що представлена на рисунку 1.4 ілюструє наступні етапи:

1) ініціалізація популяції хромосом (в даному випадку, список об’єктів класу C2D_Chromosome);

2) навчання популяції для отримання значень фітнес-функції;

3) сортування (ранжування) популяції від кращої до гіршої хромосоми;

4) застосування операторів схрещування і мутації у відповідності з оператором селекції (реалізація кросовера відкладена до майбутніх версій середовища розробки);

5) повтор 2-4 до виконання умови виходу з циклу.

Код методу, який реалізує прописаний вище алгоритм представлений в додатку Б. Спрощена схема взаємодії представлена на рисунку 2.4.

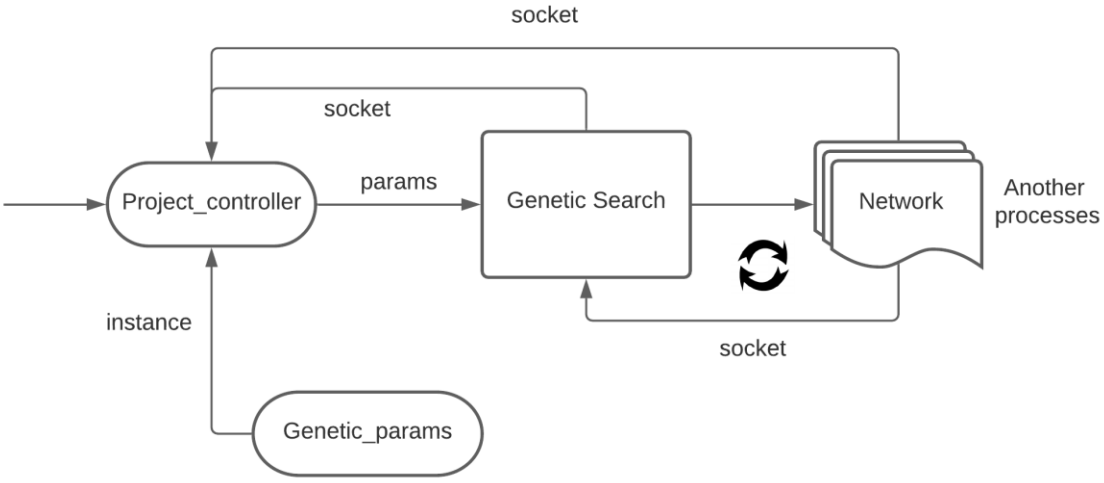

Рисунок 2.4 – Схема взаємодії класів Project_controller, GeneticProgramm і Network

2.2.4 Реалізація фітнес-функції для оцінки якості мережі.

Фітнес функція може неявно впливати на хід навчання, тому область значень повинна бути обмежена. Для пошуку оптимальної моделі необхідно вибрати метрики мережі, які б обмежили оптимум з обох сторін, кажучи математичною мовою, максимізувала одну змінну і мінімізувала іншу. Як максимізовану було вирішено обрати точність мережі, а в якості мінімізованої - кількість параметрів (вагових коефіцієнтів), які містяться в

моделі, що оцінюється. Для реалізації об'єктивної оцінки було вирішено використовувати функцію двох змінних власного виробництва, яка в процесі розробки отримала деякі модифікації:

$$A_i = \alpha_i + \frac{p_{\min}}{p_i} \tag{2.1}$$

$$A_i = \alpha_i + \alpha_i \cdot \frac{p_{\min}}{p_i} \tag{2.2}$$

$$\begin{cases} A_i = \alpha_i + \alpha_i \cdot \dfrac{p_{\min}}{p_i}, & if \quad \alpha_i \geq t \\ A_i = \alpha_i, & if \quad \alpha_i < t \end{cases} \tag{2.3}$$

де A_i - значення фітнес функції і-го члена популяції,

α_i - значення точності моделі і-го члена популяції,

p_i - кількість параметрів і-го члена популяції,

$p_{\min}$ - мінімальна кількість параметрів в межах поточної епохи еволюції,

t – порогове значення точності нейронної мережі.

Всі три функції визначені на проміжку (0, 2].

Порівняльний аналіз функцій (2.1), (2.2) і (2.3) представлений на рисунках 2.5 та 2.6. Функція (1) нестійка до великого розкиду мінімізованого параметра і як видно на графіках, мережа з низькою точністю, але незначною кількістю вагових коефіцієнтів отримує оцінку краще, ніж модель, у якій обидві змінні на порядок більші, що зовсім не відповідає критерію оптимальності для поставленого завдання. Функція (2.2) є модифікацією функції (2.1), яка в значній мірі менше схильна до проблеми, описаної вище, за рахунок збільшення значущості змінної α_i, а функція (2.3) і зовсім позбавлена такої за рахунок введення нової змінної-константи, яка по суті є граничним значенням.

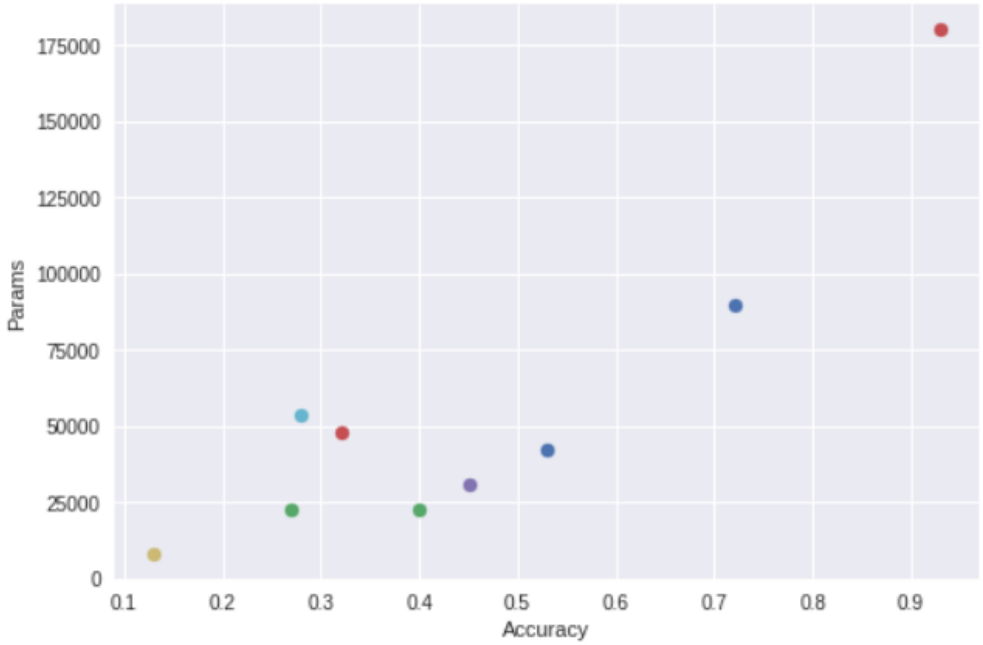

Рисунок 2.5 – Тестова множина для оцінки фітнес-функції

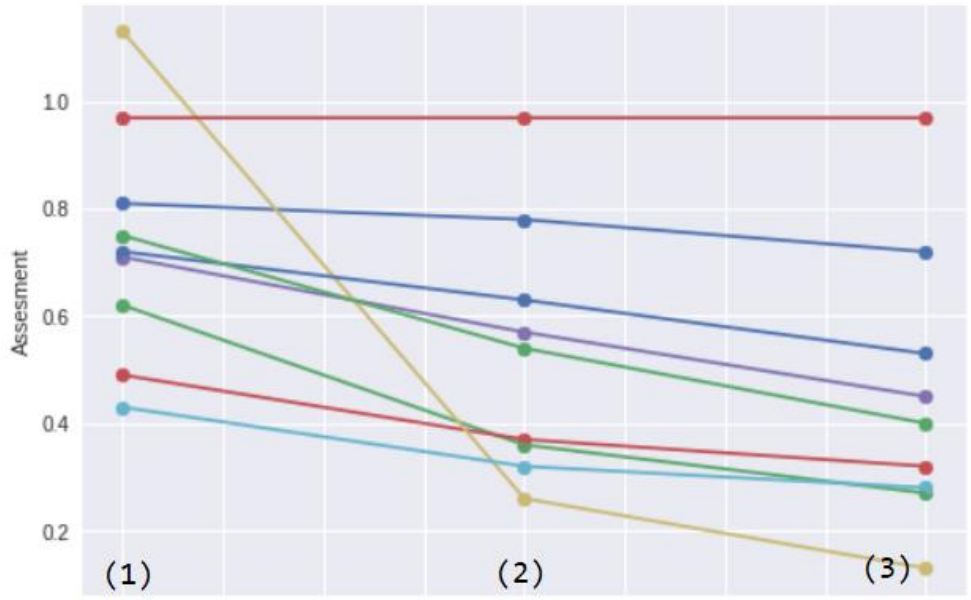

Рисунок 2.6 – Порівняльний графік оцінки тестової множини наявними функціями

Реалізація функцій (2) і (3) показана в прикладі 2.4.

```
def setAssessment(self, mode, metrics):
    if mode == 0:
        minParam = metrics[0][0]
        for i in metrics:
            if i[0] < minParam and i != 0: minParam = i[0]
```

```
        for i in range(len(metrics)):
            self.population[i].assessment = metrics[i][1].get("accuracy") +
                        metrics[i][1].get("accuracy") * minParam / metrics[i][0]
    elif mode == 1:
        minParam = metrics[0][0]
        for i in metrics:
            if i[0] < minParam and i != 0: minParam = i[0]
        for i in range(len(metrics)):
            if metrics[i][1].get("accuracy") > 60:
                self.population[i].assessment = metrics[i][1].get("accuracy") +
                     metrics[i][1].get("accuracy") * minParam / metrics[i][0]
            else:
                self.population[i].assessment = metrics[i][1].get("accuracy")
```

Приклад 2.4 – Реалізації фітнес-функцій

2.3 Графічний інтерфейс користувача.

Тут реалізується інтерактивна взаємодія з користувачем, а саме надається функціонал для управління параметрами проектів, відображається текстовий вивід інформації про поточне розрахункове завдання для проекту, інформації про результати розрахунку в рамках кожного з проектів, представленої у вигляді набору різноманітних графіків і схем, а також, зрозуміло, в даному архітектурному розділі реалізовані засоби менеджменту проектів (створення, видалення, експорт, імпорт) і безпосередньо управління процесом розрахунку. Інтерфейс реалізований засобами бібліотеки PyQt v5, що має в своєму розпорядженні зручний дизайнер для швидкої верстки елементів інтерфейсу.

На даний момент, графічний інтерфейс представлений у вигляді наступних форм:

1) Головне вікно (Home Page). Є точкою входу в додаток і його основним потоком (рисунок 2.7), з якого стартують всі інші, зокрема QueueProgramThread, який реалізує функціонал черги проектів на розрахунок.

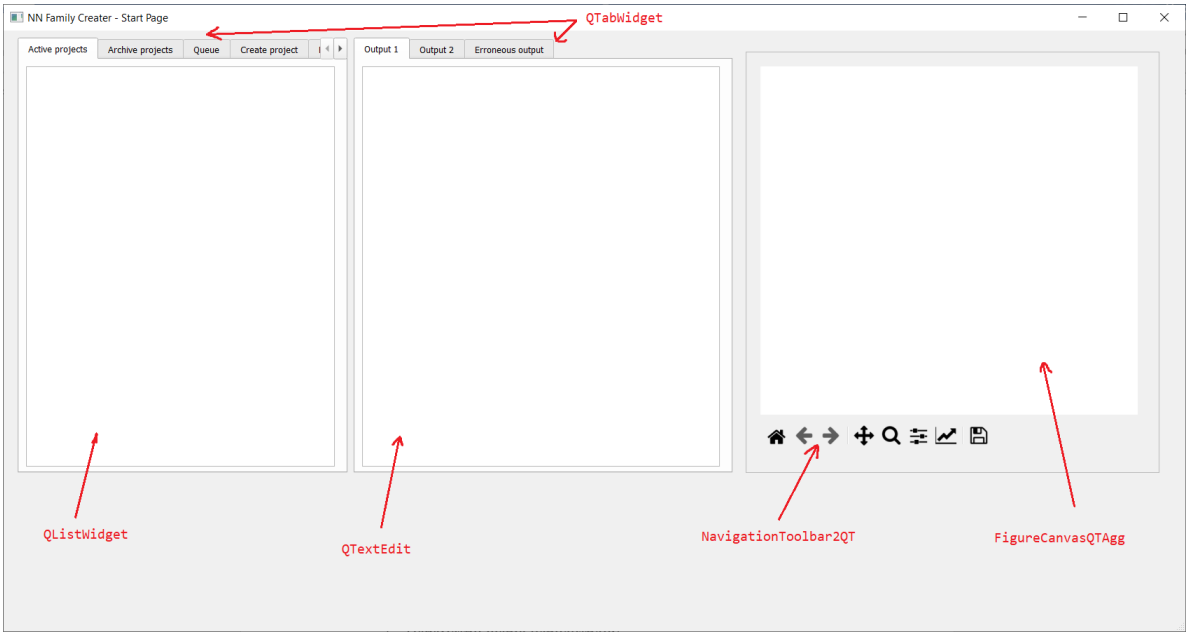

Рисунок 2.7 – Головне вікно середовища розробки

У поточній версії програми реалізовані наступні вкладки, що керують її виконанням:

– створення проекту. Вкладка дозволяє задати назву проекту, шлях для серіалізації даних і найважливіше - тип проекту. Наприклад, при заданій конфігурації, що проілюстрована на рисунку 2.8, при натисканні на кнопку «Create» створиться проект зазначеного типу, буде додано до списку проектів і відкриється вікно налаштування генетичних параметрів;

– три вкладки для відображення списків проектів. Вкладка «Active projects» показує всі створені проекти, які можна або перемістити у вкладку «Archive projects», або підготувати для розрахунку, дублюючи його у вкладку «Queue» (рисунок 2.9).

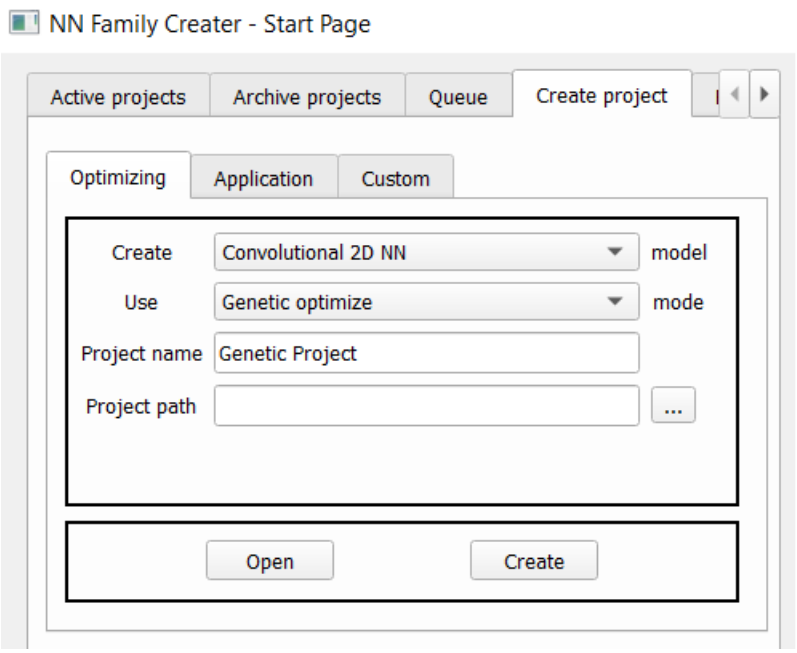

Рисунок 2.8 – Вкладка створення проекту

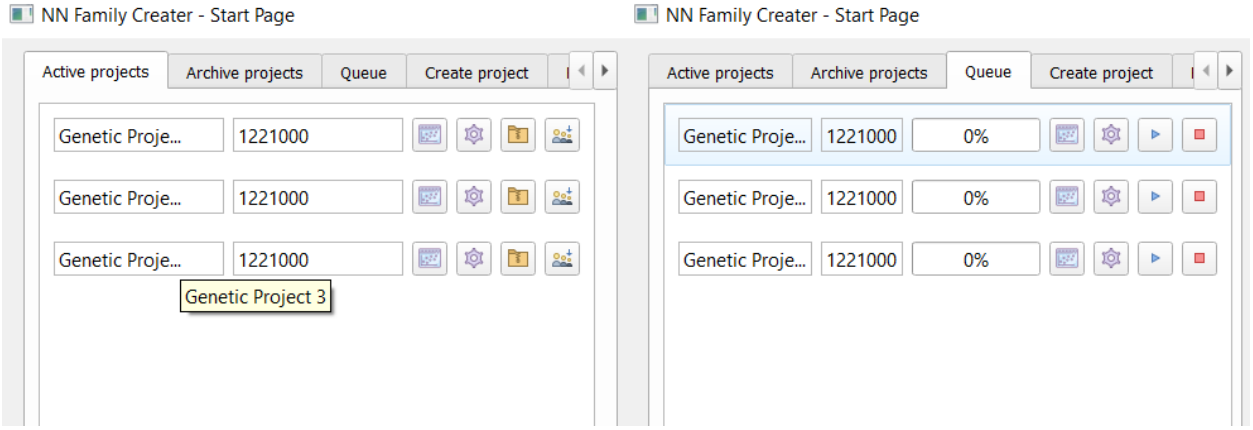

Рисунок 2.9 – Демонстрація додавання проекту в чергу на навчання

Також головне вікно містить елемент інтерфейсу для виведення логів процесу розрахунку (рисунок 2.10) і простір для малювання основного графіка виділеного проекту.

Рисунок 2.10 – Логування процесу рачетов

2) Вікно налаштування гіперпараметрів генетичного пошуку (рисунок 2.11). Створюється відразу при створенні проекту відповідного типу і при натисканні на кнопку у головному вікні середовища розробки;

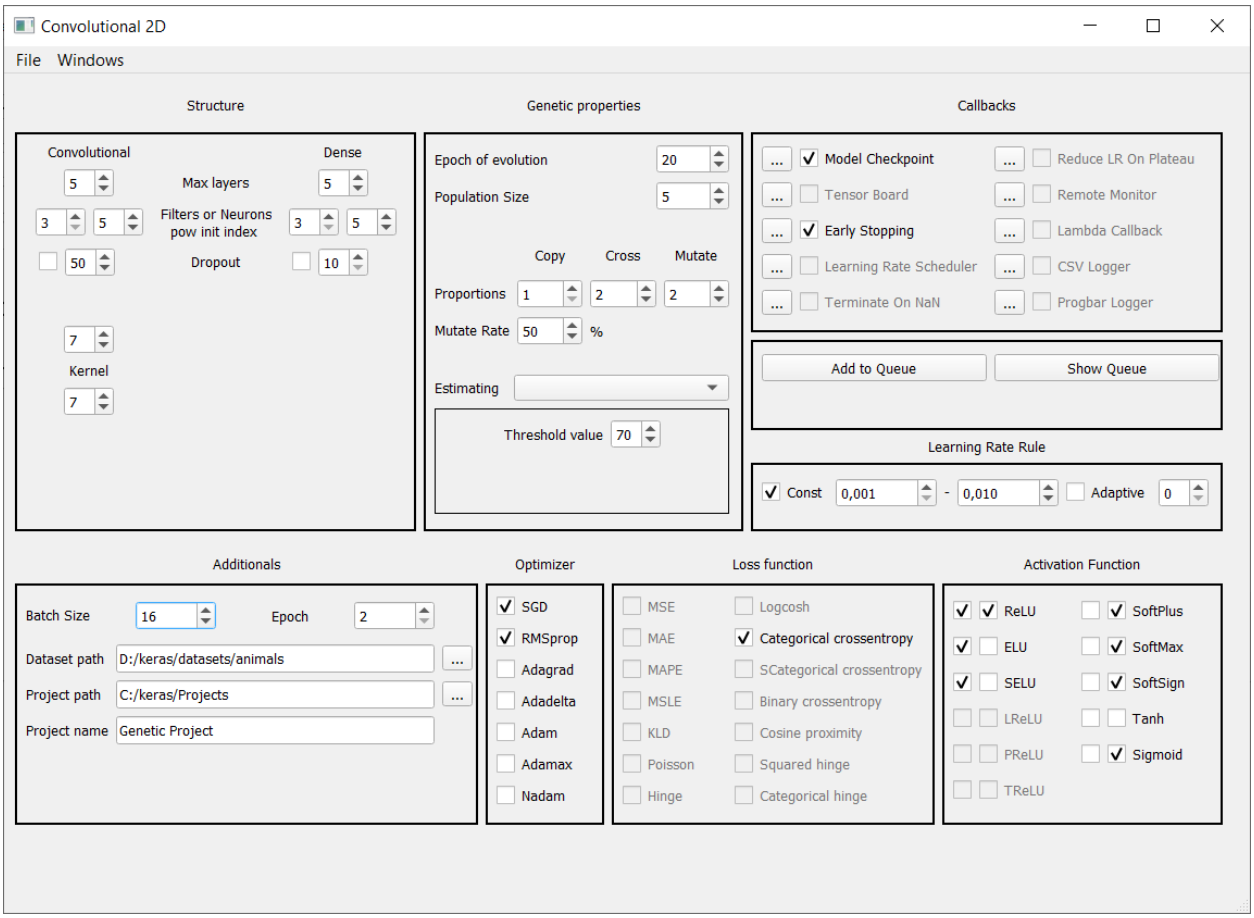

Рисунок 2.11 – Вікно налаштування параметрів для запуску генетичного пошуку

3) Діалогове вікно налаштувань програми. У даній версії програми дозволяє лише змінювати теми додатку (рисунок 3.12)

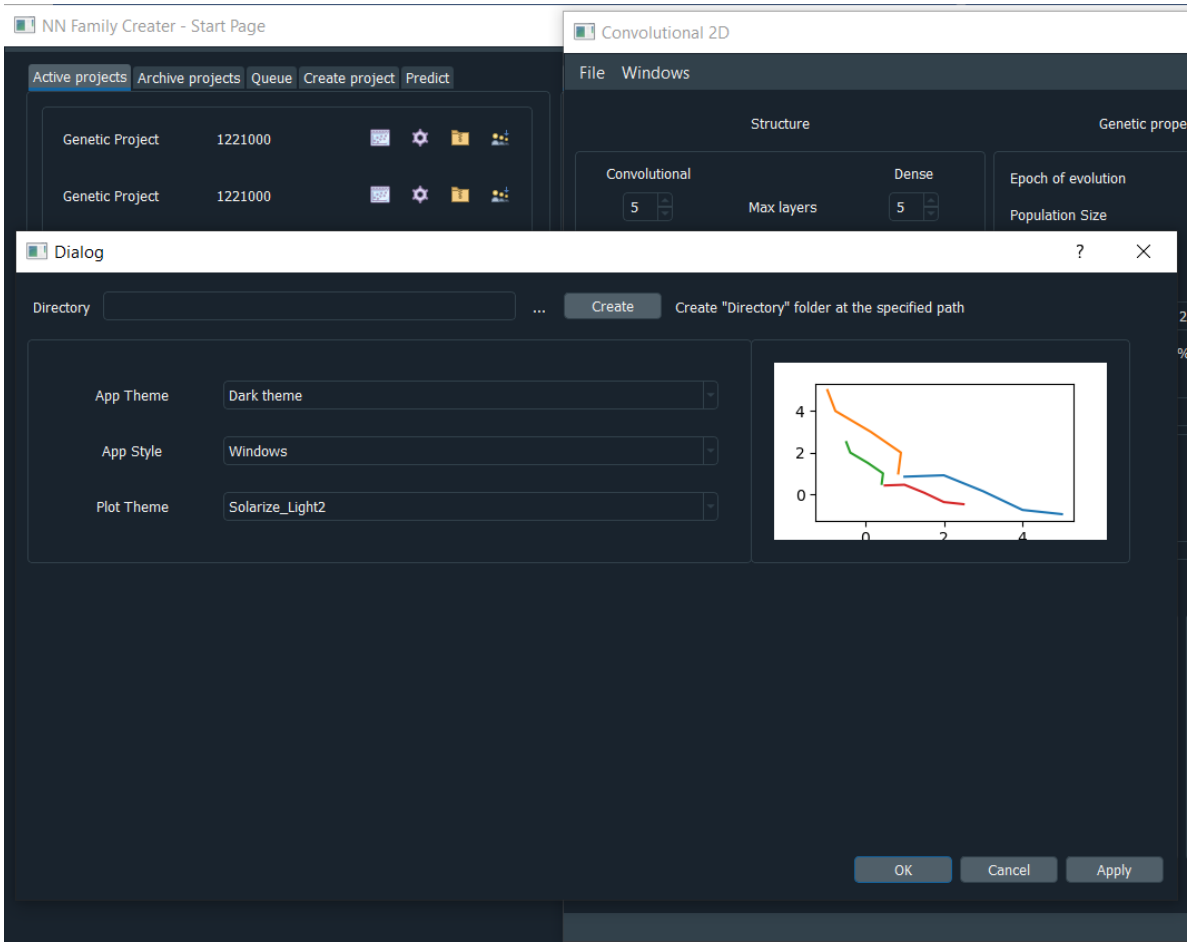

Рисунок 3.12 – Зміна теми додатку

4) Вікно, що відображує усі графіки запущеного проекту (рисунок 3.13).

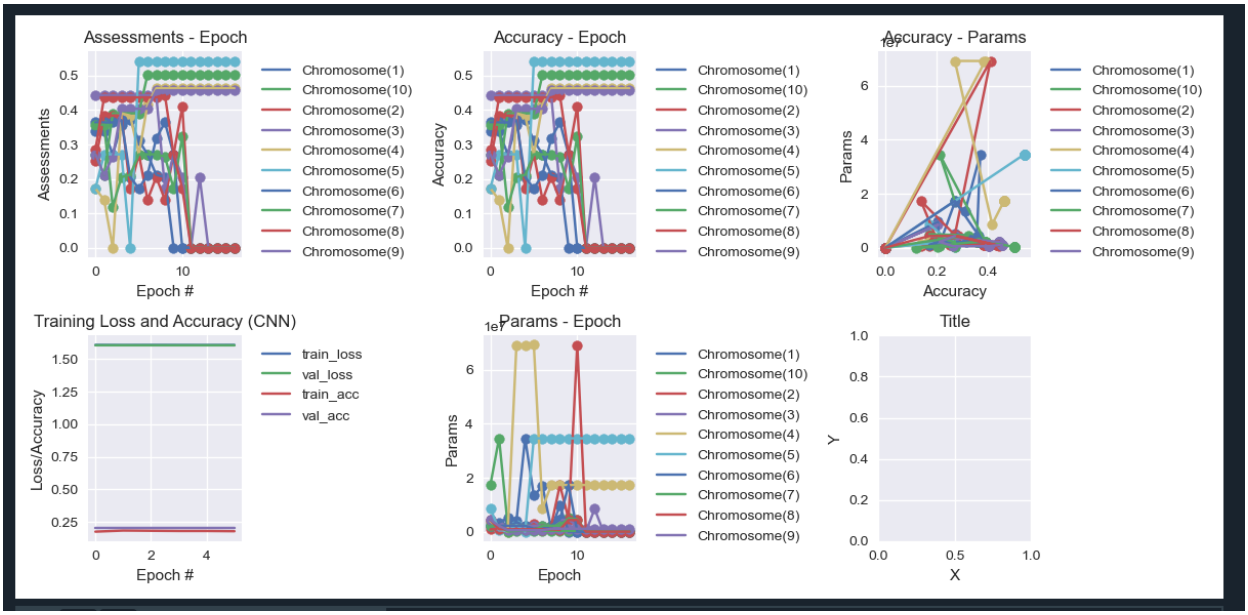

Рисунок 3.13 – Вікно графіків запущеного процесу розрахунку

ВИСНОВКИ

В ході роботи було частково реалізовано функціонал середовища розробки для роботи с нейромережевими моделями, були проаналізовані існуючі рішення в цьому напрямі. Середовище розробки надає інструменти концепції AutoML, що були розглянуті в атестаційному проекті, а саме оптимізацію гіперпараметрів нейронних мереж за допомогою генетичних алгоритмів. Деякі отримані моделі досягають точності 80-85%, що вказує на правильний напрям роботи, хоча й потребують додаткового дослідження і реалізацію обробки гіперпараметричних значень, що у поточній релізації не підтримуються

Задана архітектура проекту дозволяє додавати нові типи моделей, нові оптимізаційні алгоритми та будь-які інші інструменти з мінімальною корекцією поточної реалізації. На базі отриманого продукту створена спільнота підтримки даного проекту, що складається з членів AI Community – спільноти, зацікавленої в підтримці проектів AI сфери на території країн СНД.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. С. Николенко, А. Кадурин, Е. Архангельская. Глубокое обучение. Погружение в мир нейронных сетей. СПб.: Питер, 2018. 480 с. ISBN: 978-5-496-02536-2.

2. Бартеньев О. В. Параметры, влияющие на эффективность нейронной сети, созданной средствами Keras. URL: http://www.100byte.ru/python/factors/factors.html

3. Hinton G., Srivastava N., Swersky K. Neural networks for machine learning. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides _lec6.pdf.

4. Convolutional layers. Документація до бібліотеки. URL: https://keras.io/layers/convolutional/

5. Keras activations. Документація до бібліотеки. URL: https://keras.io/activations/

6. Keras optimizers. Документація до бібліотеки. URL: https://keras.io/optimizers/

7. Keras documentation. How can I obtain reproducible results using Keras during development? URL: https://keras.io/getting-started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development

8. Neuroph Studio. Головна сторінка. URL: http://neuroph.sourceforge. net/index.html

9. Lobe. Головна сторінка. URL: https://lobe.ai

10. Репозиторій проекту. URL: https://github.com/RinnetenseiQ/NN _Family_Creater_PythonQt

11. Саймон Д. Алгоритмы эволюционной оптимизации. М: ДМК Пресс, 2020. 940 с. ISBN 978-5-97060-812-8.

12. Р.В. Шамин. Генетические алгоритмы. Лекция №6. URL: http://ai.lector.ru/?go=lection06