Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)
Кафедра Автоматизації проектування обчислювальної техніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
(рівень вищої освіти)

Модель нейронної мережі для аналізу руху людини на STM32

(тема)

Виконав: студент 2 курсу, групи СКСм-20-1
Адамович В.Р.
(прізвище, ініціали)
Спеціальність *123 Комп'ютерна інженерія*
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма *Спеціалізовані комп'ютерні системи*
(повна назва освітньої програми)
Керівник роботи доц. Філіппенко І.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерної інженерії та управління_____
Кафедра _____Автоматизації проектування обчислювальної техніки_____
Рівень вищої освіти _____другий (магістерський)_____
_____
Спеціальність _____123 Комп'ютерна інженерія_____
Тип програми _____Освітньо-професійна_____
Освітня програма ____Спеціалізовані комп'ютерні системи_____
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студентові ____Адамовичу Владиславу Романовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту)____Модель нейронної мережі для аналізу руху____людини на STM32_____
_____
_____
_____
затверджена наказом по університету від "_04_"_11_2021 р. № __1635 Ст._____
2. Термін подання студентом роботи (проекту) _____20.12.2021_____
3. Вихідні дані до роботи (проекту)
Нейронні мережі
Аналізу руху людини
NUCLEO-G474RE
Keras
4. Зміст пояснювальної записки (перелік питань, що потрібно розробити)
Розпізнавання активності за допомогою набору даних
Розробка згорткової нейронної мережі за допомогою Keras
Підвищення точності додатковими налаштуваннями
Квантування мережі у форматі 8-бітних чисел з фіксованою точкою
Підготовка проекту для портування на мікроконтролер

Перевірка даних на прошитому мікроконтролері

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 16 слайдів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	04.11.2021 - 05.11.2021	
2	Аналіз предметної області	06.11.2021 - 07.11.2021	
3	Аналіз джерел з проблемної галузі	08.11.2021 - 11.11.2021	
4	Розробка підходу для реалізації поставленої	12.11.2021 - 16.11.2021	
5	Реалізація поставленої задачі	17.11.2021 - 26.11.2021	
6	Оформлення пояснювальної записки	29.11.2021 - 05.12.2021	
7	Оформлення графічного матеріалу	06.12.2021 - 10.12.2021	
8	Перевірка виконаного проекту керівником	13.12.2021 - 15.12.2021	
9	Захист проекту	16.12.2021 – 20.12.2021	
10			

Студент _____
(підпис)

Керівник роботи _____ доц. каф. АПОТ Філіппенко І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 83 сторінок, 25 рисунок, 13 джерел за переліком посилань.

НЕЙРОННА МЕРЕЖА, АНАЛІЗ РУХУ, МІКРОКОНТРОЛЕР, НАВЧАННЯ, STM32, X-CUBE-AI, KERAS, HAR.

Метою кваліфікаційної роботи є реалізації згорткової нейронної мережі для розпізнавання людської діяльності на базі налагоджувальної плати NUCLEO-G474RE шляхом портування нейронної мережі, навченої на класичній ЕОМ.

Для цього в роботі було розглянуто розпізнавання активності застосовуючи набір даних записаних за допомогою акселерометра та гіроскопа. Визначено переваги використання CNN для класифікації послідовностей дій. Виконана розробка згорткової нейронної мережі за допомогою відкритої нейромережевої бібліотеки Keras з наступним підвищенням точності додатковими налаштуваннями.

У готовій нейронній мережі генератором коду було перетворено значення вагів і зміщення, а також відповідні активації з формату 32-бітних чисел з плаваючою точкою в формат 8-бітних чисел з фіксованою точкою, після чого отриману модель було пристосовано для портування на мікроконтролер та виконано прошивку. На готовому мікроконтролері було виконано перевірку працездатності.

ABSTRACT

The explanatory note of the qualification work: 83 pages, 25 figures, 13 sources.

NEURAL NETWORK, MOTION ANALYSIS, MICROCONTROLLER, LEARNING, STM32, X-CUBE-AI, KERAS, HAR.

The purpose of the qualification work is to implement a convolutional neural network for recognizing human activity based on the NUCLEO-G474RE debugging board by porting a neural network trained on a classical computer.

To do this, the paper considered activity recognition using a set of data recorded using an accelerometer and Gyroscope. The advantages of using CNN to classify action sequences are determined. A convolutional neural network was developed using the Keras open neural network library, followed by improved accuracy with additional settings.

In the finished neural network, the code generator converted the values of weights and offsets, as well as the corresponding activations, from the format of 32-bit floating-point numbers to the format of 8-bit fixed-point numbers, after which the resulting model was adapted for porting to the microcontroller and firmware was performed. A health check was performed on the finished microcontroller..

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГП – графічний процесор;

ЕКГ – електрокардіографія;

ЕОМ – електронно-обчислювальна машина;

ОС – операційна система;

ПК – персональний комп'ютер;

ПЛІС – програмована логічна інтегральна схема;

РНМ – рекурентна нейронна мережа;

ШНМ – штучна нейронна мережа;

ANN – штучні нейронні мережі (англ., Artificial Neural Networks);

API – інтерфейс програмування застосунків (англ., Application Programming Interface);

CAA – архітектура адаптивного масиву (англ., Crossbar Adaptive Array);

GPU – графічний процесор (англ., Graphics Processing Unit);

HAR – розпізнавання людської діяльності (англ., Human Activity Recognition);

IDE – інтегроване середовище розробки (англ., Integrated Development Environment);

KBS – система, заснована на знаннях (англ., Knowledge-based systems);

MCU – мікроконтролер (англ., Micro Controller Unit);

MSE – середньоквадратична похибка (англ., Mean Squared Error);

NAS – пошук нейронної архітектури (англ., Neural Architecture search);

SMPS – імпульсний стабілізатор напруги (англ., Switched-mode power supply);

SOM – самоорганізаційна карта (англ., Self Organizing Maps);

SVM – метод опорних векторів (англ., Support Vector Machine);

TPU – тензорний блок обробки (англ., Tensor Processing Unit).

ВСТУП

Світ активно розвивається, особливо в технологічному напрямку. Майже щороку технічний прогрес робить великий і важливий крок вперед, винаходячи все більш різноманітні технології. З моменту створення першого ЕОМ и до сьогодні значна частина програмного забезпечення складається за допомогою алгоритмічних мов, що за своїм принципом припускають незмінну структуру алгоритму. Іншими словами всі можливі варіанти програми отримуються після завершення деякого завчасно прописаного набору умов. Подібні алгоритми справно працюють завдяки високому рівню прогнозованості та при певному обсязі моделювання зменшують шанс виникнення неординарних ситуацій. Натомість програмне забезпечення створене за допомогою таких алгоритмічних мов не кращим чином виконують розрахунок задач з нечітким результатом, тобто результатом, що знаходиться в деякому діапазоні значень з певною ймовірністю. Саме для подібних завдань були створені штучні нейронні мережі, що імітують роботу біологічних нейронних мереж. Штучна нейронна мережа являє собою сукупність нейронів, з'єднаних між собою синапсами. Мережа з подібною архітектурою надає можливість продуктивного вирішення задач класифікації, передбачення, розпізнавання і так далі. Для реалізації таких нейронних мереж використовуються ЕОМ, що мають досить великі обчислювальні ресурси, при цьому може використовуватися як центральний, так і графічний процесор.

В рамках даної роботи розглядається можливість реалізації нейронної мережі на базі мікроконтролерів шляхом портування нейронної мережі, навченої на класичній ЕОМ.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Штучна нейронна мережа

Штучні нейронні мережі (ШНМ, англ. artificial neural networks, ANN) – це обчислювальні алгоритми, в основі яких принцип імітації роботи нейронів мозку живих організмів. Дані системи мають змогу зменшувати похибку своєї роботи без конкретного програмування під конкретну задачу, тобто навчаються розглядаючи зразкові рішення. Прикладом такої роботи може служити розпізнавання об’єктів або тварин на зображеннях. Для цього нейронну мережу навчають на зображеннях з відповідними мітками в результаті ANN може ідентифікувати об’єкти на нових зображеннях. Цей процес проходить без апріорного знання про ціль, натомість створюються персональний комплекс характеристик з навчального матеріалу, що був розглянутий раніше.

Головною ідеєю покладеною в основу роботи штучної нейронної мережі є наближення до природних нейронів мозку тварин шляхом створення комплексу з'єднаних вузлів, що мають назву штучних нейронів. Усі зв’язки між штучними нейронами транспортують сигнал між собою. Отримуючи сигнал, нейрон має змогу опрацьовувати його після чого повідомити сусіднім нейронам.

У більшості працюючих прикладах нейронних мереж сигнал, що передається між нейронами є дійсним числом, а вихідне значення на штучних нейронах вираховуються по сумі вхідних сигналів. Особливістю нейронів та зв’язків між них є наявність вагового коефіцієнта, що корегується у процесі навчання. Ваговий коефіцієнт впливає на силу сигналу. Існує можливість, що поріг в нейронах в штучних мереж буде надсилатися тільки при сумі перевищуючої даний поріг. Найчастіше нейрони зображуються у вигляді шарів. При цьому від різновиду такого шару буде

залежати яке перетворення буду виконано. Сигнал не завжди доходить до вихідного шару (нейрону) з першої спроби, є шанс, що для цього потребується декілька проходжень для перевищення порогу.

Ціллю створення штучних нейронних мереж було створення можливості вирішення завдань імітуючи людський мозок. Але з розвитком такого підходу більша частина сил сконцентрувалася на схожості з певними розумовими властивостями віддаляючись від біології. Сфери застосування штучних нейронних мереж майже необмежені до їх складу входять: комп'ютерне бачення, розпізнавання мовлення, машинний переклад, розпізнавання рухів, соціально-мережеве фільтрування, гра в настільні та відеоігри, медичне діагностуванням та інше.

1.2 Історія

У 1943 році Воррен Маккалох та Волтер Піттс вперше створили модель з використанням математичних алгоритмів для нейронних мереж [1]. Дана модель стала першим кроком на шляху до поділу досліджень нейронних мереж на два підходи. Перший підхід сконцентрувався на біологічних процесах у мозку, тоді як інший зосереджується на застосуванні нейронних мереж до штучного інтелекту. Ця праця привела до дослідження мережів нервових з’єднань зі скінченними автоматами [2].

1940 року була створена одна з гіпотез навчання, що дістала назву по прізвищу свого засновника Дональда Гебба («Геббове навчання»). Ця гіпотеза базувалося на механізмі нейропалстичності. Геббове навчання є спонтанним навчанням. Пізніше була створена модель довготривалого потенціювання, що мало в своїй основі гіпотезу Гебба. Втілення в розрахункових моделях дані ідеї отпримали у 1948 році в машинах Тюрінга.

Фарлі та Клар (1954) вперше використали обчислювальні машини, звані тоді «калькуляторами», для представлення мережі гебба. Інші нейромережеві обчислювальні машини втілилися у життя у 1956 році групою

дослідникі та інженерів у складі Рочестера, Голланда, Гебіта та Дуди.

У 1958 році Розенблатом було створено алгорит для визначення образів. Використовуючі матиматичний запис він представив схему виключного «або», тобто схему не примітивного перцептрону, яке в той час обробляти нейронними мережами було неможливо.

Г'юбел та Візел лауреати нобелівської премії у 1059 році запропонували біологічну модель, що була заснована на дослідженні різновидів клітин у первинній зоровій корі: простих клітин та складних клітин.

Перші працездатні мережі з багатьма шарами було опубліковано Івахненком та Лапою 1965 року, вони стали методом групового урахування аргументів.

Дослідження нейронних мереж зупинилося слідом за вивченням машинного навчання Мінського та Пейперта (1969), які відкрили дві головні проблеми з обчислювальними машинами, що обробляли нейронні мережі. По-перше, стандартним перцептронам було не підсилу опрацьовувати схему виключного «або». По-друге, комп'ютери того часу не володіли обчислювальною потужностію достатньою для ефективного виконання роботи, потрібної великим нейронним мережам. Дослідження нейронних мереж уповільнилося, поки комп'ютери не досягли набагато більшої обчислювальної потужності.

Найбільшу частину нейронних мереж було сконцентровано на оброблювальних алгоритмах складних моделей, які характеризуються уособленням в параметрах пізнавальної моделі. Прикладом таких моделей були експертні системи з данними представленими у вигляді правил “if – else”, але у кінці вісімдесятих років 20 століття дослідження не розповсюдились на низькорівневе машинне навчання.

Основним ключем до нової хвилі інтересу до нейронних мереж та навчання став алгоритм зворотоного поширення Вербоса (1975), що мав підвищину ефективність у вирішенні проблем, і який прискорив навчання

багаторівневих мереж. Зворотне поширення розповсюджувало член похибки шарами в зворотному напрямку, змінюючи ваги в кожному вузлі.

Приблизно в 1985 році почала набувати розповсюдження розподілена паралельна обробка, що дістала назву конективізму. У 1986 році Румельхартом та МакКлелландом було описано використання конективізму для моделювання нейронних процесів.

Проте більш прості методи, наприклад, лінійні класифікатори та метод опорних векторів поволі наздогнали нейронні мережі за популярністю в машинному навчанні.

Попередні виклики в тренуванні глибинних нейронних мереж було успішно розв'язано за допомогою таких методів, як спонтанне попереднє тренування, в той час як доступна обчислювальна потужність зросла через застосування ГП та розподілених обчислень. Нейронні мережі почали активно використовувати в завданнях визначення об’єктів на зображеннях та відео. Такий процес дістав назву глибинне навчання, проте він не є строго синонімічним до глибинних нейронних мереж.

У 1992 році було презентовано рішення для аналізу та знаходження тривимірних об’єктів шляхом максимального агрегитування, що спрощує терпимість до деформації та інваріантність до найменшого зсуву.

Проблема зникання градієнту впливає на багатошарові мережі прямого поширення, які використовують зворотне поширення, а також на рекурентні нейронні мережі (РНМ). При накопиченні похибки на різних рівнях, відбувається експоненціальне зменьшення РНМ з кількістю шарів. Такий підхід заснований на похибках, що стримують налаштування вагових коефіцієнтів нейронів.

У 1992 році Шмідгубер для обходу даної проблеми викоритсав багатошарову ієрархію мереж, що заздалегіть було навчано по одному шарові за раз за допомогою спонтанного навчання, а потім тонко налаштовуваних зворотним поширенням. Бенке в завданнях з відновлення зображень та знаходження положення обличь, використовував еластичне

зворотне поширення, тобто розраховував тільки на знак.

Хінтон та ін. (2006) запропонували використовувати послідовні двійкові шари або дійснозначні латентні змінні для тренувааня представлень високого рівня з обмеженою машиною Больцмана для моделювання кожного шару. Після завершення тренування достатьої кількості шарів, можна застосовувати глибинну архітектуру як породжувальну модель, користуючись спадковим прохідод, що значило виконная вибірки моделі донизу від збудження ознак верхнього рівня. 2012 року Ин та Дін відтворили мережу, що тренувалася знаходити поняття виского рівня, наприклад, котів, використовуючі для цього зображення без позначення, отриманих з YouTube відео.

Було створено обчислювальні пристрої в КМОН, як для біофізичного моделювання, так і для нейроморфних обчислень. Нанопристрої для аналізу важливих компонентів масштабом збільшеної величини та згортки мають шанс створити новий різновид нейронних обчислень, оскільки вони у своїй базі є аналоговими, а не цифровими. Чирешан з колегами (2010) з групи Шмідгубера показали, що, незважаючи на проблему зникання градієнту, ГП роблять зворотне поширення придатним для багатошарових нейронних мереж з прямим поширенням.

В період з 2009 по 2012 рік рекурентні нейронні мережі та глибинні нейронні мережі прямого поширення, розроблені при участі експериментальній групі Шмідгубера, отримали перемогу у восьми міжнародних змаганнях з розпізнавання образів та машинного навчання. Наприклад, двоспрямована та, що має багато вимірів, довга, короткочасна, пам'ять Ґрейвса та ін. отримали перемогу у трьох змаганнях з розпізнаванні неперервного рукописного тексту на Міжнародній конференції з аналізу та розпізнавання документів (англ. ICDAR) 2009 року без жодного попереднього знання про три мови, яких було потрібно навчитися.

Чирешан з колегами виграли змагання з розпізнавання образів, включно зі Змаганням з розпізнавання дорожніх знаків IJCNN 2011 року, Змаганням із сегментування нейронних структур у стекових типах даних цифрової мікроскопії ISBI 2012 року та іншими. Їхні нейронні мережі були першими, що досягли порівняної зі звичайною людською, або понадлюдської продуктивностю на таких еталонах, як розпізнавання дорожніх знаків (IJCNN 2012) та задача рукописних цифр MNIST.

У 2010 році вчені довели зниження похибки в завданнях з іпользованіем словників великого розміру розпізнавання мови (голосовий пошук) при іспользоваіі глибинних нейронних мереж, які пов'язані з прихованою марковської моделлю з контекстно-залежними станами, які беруть участь у визначенні шару виходу нейронної мережі.

Реалізація даного способу, в основі якого лежить голосовий пошук, отримали перше місце у багатьох змаганнях з ідентифікації образів, а також зі змаганням з розпізнавання дорожніх знаків IJCNN 2011 року, змаганням із сегментування нейронних структур в EM-стеках ISBI 2012 року, змаганням ImageNe та іншими.

Глибинні, високонелінійні нейронні архітектури, подібні до неокогнітрону та «стандартної архітектури бачення», натхнені простими та складними клітинами, було попередньо треновано спонтанними методами Хінтоном. Команда з заснованої ним лабораторії отримала перше місце 2012 року, яке проспонсувала компанія Merck, для створення програмного забезпечення для допомоги в пошуку молекул, що можуть знайти нові ліки.

У 2011 році розпочалося змагання за звання передової в мережах прямого розповсюдження глибинного навчання між шарами максимізаційного агрегування та згортковими шарами, після яких йде рівень фінальної класифікації. Тренування зазвичай виконується без спонтанного

попереднього навчання.

Такі керовані методи глибинного навчання були першими, що досягли в певних задачах продуктивності, порівняної з людською.

ШНМ змогли гарантувати інваріантність до зсуву, щоби обходитися з маленькими та великими природними об'єктами у великих загромаджених сценах, тільки після інваріантності розповсюдження за кордони зсуву, для всіх тренованих ANN поняття, такі як місце знаходження, тип, масштаб, освітлення та інші. Це було реалізовано в еволюційних мережах, чиїми втіленнями є мережі «де-що», від WWN-1 (2008) до WWN-7 (2013).

1.3 Моделі

ANNs починалася як спроба використовувати структуру біологічного нейрону для розрахунку задач, що були занадто складні для простих алгоритмів не мали великого розповсюдження. Невдовзі вони переорієнтувалися на поліпшення практичних результатів, в основному відмовившись від зусилля неухильного дотримання попередників біологічного шляху. Нейрони з'єднані один з одним уникальніми візерунками, з цілью поєднання нейронівм між собою. Мережа створена за таким принципом схожа на орієнтований зважений граф.

До складу нейронної мережі входять сукупніть змодельваних нейронів. А нейрони самі по собі є вузлами цієї мережі поєднанними між собою зв'язками, що є аналогами біологічним з’єднаннями аксон-синапс-дендрит. Усі вузли мають певний ваговий коефіцієнт, що вирішує з якої сили будут мати вплив ці ланки

Енн, в свою чергу, складаються з нейронів, що в своїй основі мають біологічну архітектуру. Будь-який з них має по декілька входів і лише один вихід, який далі може бути відправлений сукупності нейронів. До вхідної інформації можуть відноситися: зображення, властивості вібірки, вихідна інформація з інших нейронів. Вихідні зв’язки усієї мережі мають дані, що є

відвовіддю на поставлену задачу.

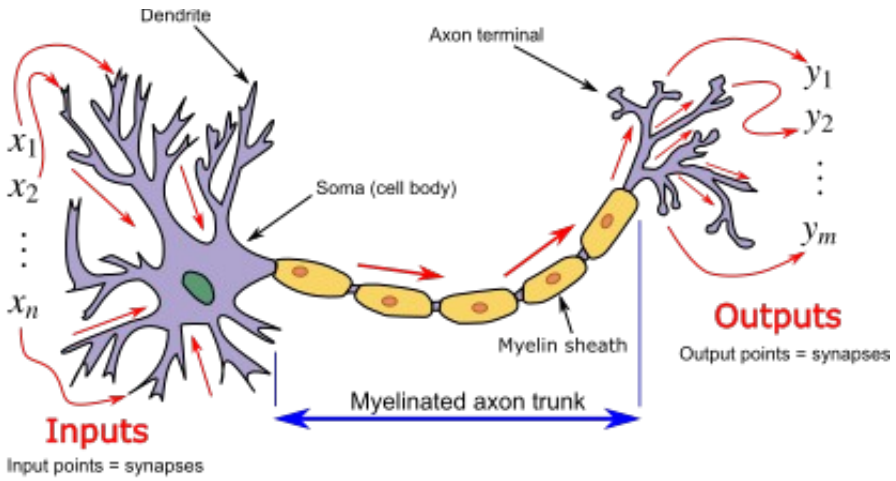

Рисунок 1.1 – Біологічний нейрон

Для пошуку інформації, що дійде до виходу нейрона, необхідно проаналізвуати зважену суму, що приходить на нього, інколи її називають активаційною. Далі до неї додається термін. Наступним кроком є транспортування цієї суми за допомогою нелінійної функції, щоб дізнатися вихідну інформацію. До інформації, що може приходити або виходити з нейрона відносять, наприклад, зображення або документи. Фінальне обчислення і є відповіддю на поставлену задачу, наприклад, назва об’єкта, що зображений на картинці.

До структури нейронної мережі входять з'єднання, що гарантують вихід принайні одного вузла до входу наступного. По аналогії з вузлами у зв’язків теж є ваговий коефіцієнт, що представляє його силу.

Функція розповсюдження вираховує вхідну інформацію нейрона за допомогою кінцях нейронів-попередників, що поєднуються у зважену суму.

В залежності від ріжновиду тренувань нейрони можуть бути поєднані у декілька шарів, кількіть цих шарів називають глибиною навчання. При такій архітектурі нейрони, що входять до складу певного шару пожуть поєднуватися лише з сусідніми. Шар до якого надходить первинна

інформація називають вхідним. Напротивагу йому результуючий шар, називають вихідним, або кінцевим. Шари, що розташовані між вхідним та кінцевим називаються прихованими. Їх може бути нуль, або більше. Інколи застосовують мережі одношарові та багатошарові. Існує не один варінат поєднання двох шарів. При цілковитому поєднанні усі вузли одного шару маєть зв’язок з усіма вузлами наступного шару.

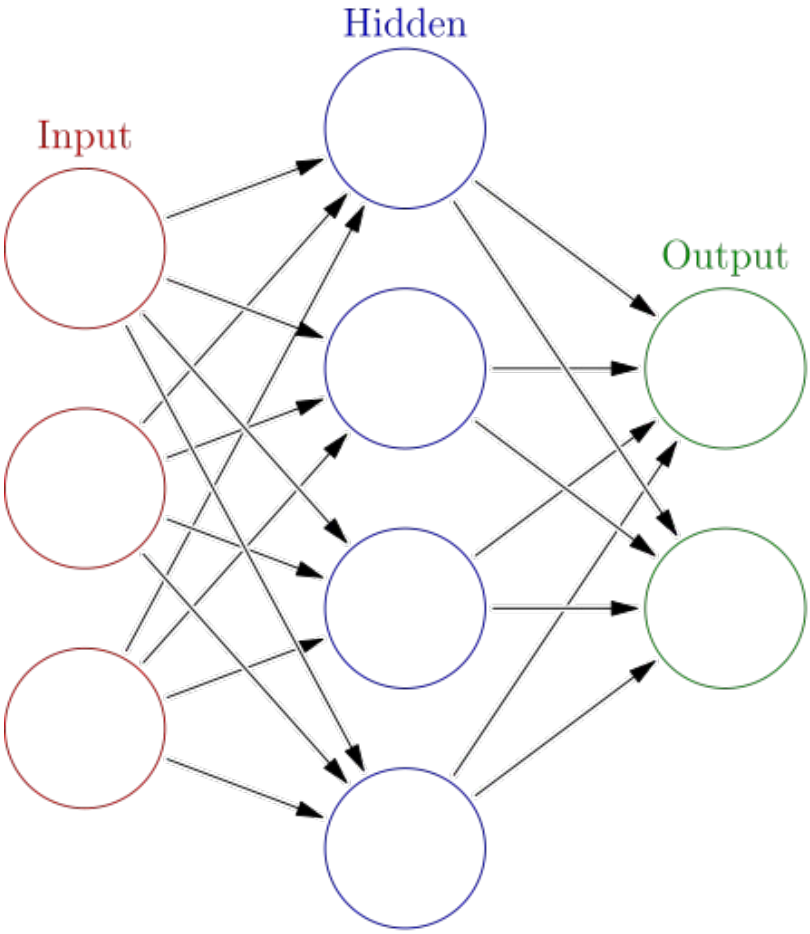

Рисунок 1.2 – Штучна нейронна мережа

При зв’язку сукупності вузлів одного шару з лише одним вузлом сусіднього отримується зменшення вузлів на цьому шарі. При такій архітектурі нейронних з’язків є можливість отримати мережу з прямим зв’язком інколи іменуєму спрямованим ациклічним графом. На противагу,

нейронні мережі, що дозволяють утворювати контакт між вузлами в одному і тому ж або попередніх шарах, називаються рекурентними.

Гіперпараметром називають такий параметр зміст якого наповнюється до старту тренувань. Швидкість з якою проходять тренування, розмір архітектури, це все приклади гіперпараметрів. Інколи вони пов’язані між собою прямою залежністю, наприклад, величина шарів може знаходитися у залежності від числа шарів.

Навчання – це адаптація мережі для кращого виконання завдання з урахуванням вибіркових спостережень. Навчання включає в себе настройку ваг (і додаткових порогових значень) мережі для підвищення точності результату. Це робиться за рахунок мінімізації спостережуваних помилок. Навчання завершено, коли вивчення додаткових спостережень не призводить до значного зниження частоти помилок. Навіть після навчання частота помилок зазвичай не досягає 0. Якщо після навчання частота помилок занадто висока, мережа, як правило, повинна бути перероблена. Практично це робиться шляхом визначення функції витрат, яка періодично оцінюється під час навчання. Поки його продуктивність продовжує знижуватися, навчання триває. Вартість часто визначається як статистика, значення якої може бути тільки приблизним. Вихідні дані насправді є числами, тому, коли помилка невелика, різниця між висновком (майже напевно кішкою) і правильною відповіддю (кішкою) невелика. Навчання намагається зменшити загальну кількість відмінностей між спостереженнями [3]. більшість моделей навчання можна розглядати як просте застосування теорії оптимізації та статистичної оцінки.

Швидкість навчання визначає розмір коригувальних кроків, які модель вживає для коригування помилок у кожному спостереженні. Висока швидкість навчання скорочує час навчання, але з меншою кінцевою точністю, в той час як більш низька швидкість навчання займає більше часу, але з потенціалом для більшої точності. Такі оптимізації, як Quickprop, в першу чергу спрямовані на прискорення мінімізації помилок, в той час як

інші поліпшення в основному спрямовані на підвищення надійності. Щоб уникнути коливань всередині мережі, таких як змінні ваги з'єднань, і підвищити швидкість збіжності, удосконалення використовують адаптивну швидкість навчання, яка збільшується або зменшується в міру необхідності [4]. концепція імпульсу дозволяє зважувати баланс між градієнтом і попередньою зміною таким чином, щоб коригування ваги в деякій мірі залежала від попередньої зміни. Імпульс, близький до 0, підкреслює градієнт, в той час як значення, близьке до 1, підкреслює остання зміна.

Функція витрат. Хоча можна визначити функцію витрат ad hoc, часто вибір визначається бажаними властивостями функції (такими як опуклість) або тому, що вона випливає з моделі (наприклад, у імовірнісній моделі апостеріорна ймовірність моделі може використовуватися в якості зворотної вартості).

Зворотне поширення – це метод, який використовується для налаштування ваг з'єднань для компенсації кожної помилки, виявленої під час навчання. Кількість помилок ефективно розподіляється між сполуками. Технічно backprop обчислює градієнт (похідну) функції витрат, пов'язаної з даним станом, по відношенню до ваг. Оновлення ваги може бути виконано за допомогою стохастичного градієнтного спуску або інших методів, таких як екстремальні навчальні машини, мережі “без опори”, навчання без повернення, “невагомі” мережі, і неконнекціоністські нейронні мережі [5].

Трьома основними парадигмами навчання є навчання під наглядом, навчання без нагляду і навчання з підкріпленням. Кожен з них відповідає певній навчальній задачі

Контрольоване навчання використовує набір парних вхідних даних і бажаних вихідних даних. Завдання навчання полягає в тому, щоб отримати бажаний результат для кожного входу. У цьому випадку функція витрат пов'язана з усуненням неправильних відрахувань [6]. зазвичай використовуваною вартістю є середньоквадратична помилка, яка намагається

мінімізувати середньоквадратичну помилку між виходом мережі і бажаним виходом. Завданнями, придатними для контрольованого навчання, є розпізнавання образів (також відоме як класифікація) і регресія (також відома як апроксимація функцій). Контрольоване навчання також застосовується до послідовних даних (наприклад, для письма від руки, розпізнавання мови і жестів). Це можна розглядати як навчання з “вчителем” у формі функції, яка забезпечує безперервний зворотний зв'язок про якість рішень, отриманих на даний момент.

Під час використання навчання без вчителя відхід від кібернетичного експерименту та побудова теорії потребують математичної формалізації. Прикладом одного із різновидів використання даної формалізації є відображення в теорії розпізнавання образів.

Це відхилення від побудови теорії та експерименту з’явилося через конфлікт поглядів у різних фахівців. Основа даного відхилення викликана питанням: «чи можливі єдині принципи адекватного опису образів різної природи, або ж такий опис кожен раз є завдання для фахівців конкретних знань?»

З одного боку, головною метою визначення повинно бути знаходження загальних ідей застосування апріорної інформації при створенні адекватного опису образів. Важливим доповненням є, що ці відомості повинні мати однаковий принцип обліку навіть за різної природи цих образів. З іншого боку, питання одержання опису відокремлюється від загального визначення, що сприяє спрощенню теорії навчання машинного зору до питання зменшення середнього ризику у спеціалізованому класі головних правил [7].

Навчання з підкріпленням. У таких додатках, як відеоігри, актор виконує ряд дій, отримуючи в цілому непередбачувану відповідь від навколишнього середовища після кожного з них [8]. Мета полягає в тому, щоб виграти гру, тобто отримати найбільш позитивні (з найменшими витратами) відповіді. Під час навчання з підкріпленням мета полягає в тому, щоб зважити мережу (розробити політику) для виконання дій, які

мінімізують довгострокові (очікувані сукупні) витрати. У кожен момент часу агент виконує дію, і середовище генерує спостереження і миттєву вартість відповідно до деяких (зазвичай невідомими) правилами. Правила і довгострокову вартість зазвичай можна оцінити тільки приблизно. У будь-який момент агент вирішує, чи слід вивчати нові дії, щоб виявити їх вартість, або використовувати попереднє навчання, щоб діяти швидше.

ANNs служать компонентом навчання в таких додатках. динамічне програмування в поєднанні з ANNs (нейродинамічне програмування) застосовується до таких завдань, як маршрутизація транспортних засобів, відеоігри, управління природними ресурсами і медицина через здатність ANNs знижувати втрати точності навіть при зменшенні щільності сітки дискретизації для чисельного наближення рішення задач управління. Завданнями, які підпадають під парадигму навчання з підкріпленням, є завдання управління, Ігри та інші завдання послідовного прийняття рішень.

Самонавчання в нейронних мережах було введено в 1982 році разом з нейронною мережею, здатною до самонавчання, під назвою Crossbar Adaptive Array (CAA). це система тільки з одним входом, ситуацією s, і тільки одним виходом, дією (або поведінкою) a. він не має ні зовнішніх рекомендацій, ні зовнішніх підкріплень, що надходять з навколишнього середовища. CAA обчислює, у вигляді поперечини, як рішення про дії, так і емоції (почуття) з приводу виниклих ситуацій. Система управляється взаємодією між пізнанням і емоціями. враховуючи матрицю пам'яті, W=||w(a,s)||, алгоритм Самонавчання поперечини на кожній ітерації виконує наступні обчислення:

- у ситуації s виконайте дію a;
- отримати наслідки ситуації s';
- обчисліть емоцію перебування в ситуації наслідків v (s');
- оновіть пам'ять поперечини w'(a,s) = w(a,s) + v (s').

Зворотне значення (Вторинне підкріплення) - це емоція по відношенню до ситуації наслідків. CAA існує в двох середовищах: одне-поведінкове середовище, в якому він поводиться, а інше-генетичне середовище, в якому

він спочатку і тільки один раз отримує початкові емоції, пов'язані з ситуаціями, з якими він стикається в поведінковому середовищі. Отримавши вектор генома (видовий вектор) з генетичного середовища, CAA навчиться цілеспрямованій поведінці в поведінковому середовищі, яка містить як бажані, так і небажані ситуації [9].

Нейроеволюція може створювати топології і ваги нейронних мереж з використанням еволюційних обчислень. Він конкурує зі складними підходами градієнтного спуску. Однією з переваг нейроеволюції є те, що вона може бути менш схильна до потрапляння в “тупики”.

1.4 Типи нейронних мереж

ANNs перетворилися на широке сімейство методів, які просунули сучасний стан у багатьох областях. Найпростіші типи мають один або кілька статичних компонентів, включаючи кількість одиниць вимірювання, кількість шарів, вага одиниць вимірювання та топологію. Динамічні типи дозволяють одному або декільком з них розвиватися за допомогою навчання. Останні набагато складніше, але можуть скоротити терміни навчання і дати кращі результати. Деякі типи дозволяють/вимагають, щоб навчання “контролювалося” оператором, в той час як інші працюють незалежно. Деякі типи працюють виключно на апаратному забезпеченні, в той час як інші є чисто програмними і працюють на комп'ютерах загального призначення.

Деякі з основних досягнень включають в себе: згорткові нейронні мережі, які виявилися особливо успішними в обробці візуальних та інших двовимірних даних; довготривала короткочасна пам'ять дозволяє уникнути проблеми зникаючого градієнта і може обробляти сигнали, що містять поєднання низькочастотних і високочастотних компонентів, що сприяють розпізнаванню мови з великим словниковим запасом синтез тексту в мову, і фотореальні балакучі голови; конкурентні мережі, такі як генеративні змагальні мережі, в яких кілька мереж різної структури) конкурують один з

одним, про такі завдання, як перемога в грі або про обман противника щодо автентичності вхідних даних [10].

1.5 Проектування мережі

Пошук за нейронною архітектурою (NAS) використовує машинне навчання для автоматизації проектування ANN. Різні підходи до NAS розробили мережі, які добре поєднуються з системами ручної розробки. Основний алгоритм пошуку полягає в тому, щоб запропонувати модель-кандидат, оцінити її по набору даних і використовувати результати в якості зворотного зв'язку для навчання мережі NAS. доступні системи включають AutoML і AutoKeras.

Питання проектування включають в себе визначення кількості, типу і зв'язності мережевих рівнів, а також розміру кожного і типу з'єднання (повне, об'єднання в пул, и т.д.).

Гіперпараметри також повинні бути визначені як частина дизайну (вони не вивчаються), регулюючи такі питання, як кількість нейронів в кожному шарі, швидкість навчання, крок, крок, глибина, поле сприйняття і заповнення і т. д.

1.6 Використання

Завдяки своїй здатності відтворювати і моделювати нелінійні процеси штучні нейронні мережі знайшли застосування в багатьох дисциплінах.

Області застосування включають ідентифікацію та управління системами (управління транспортними засобами, прогнозування траєкторії, управління технологічними процесами, управління природними ресурсами), квантову хімію, загальну гру, розпізнавання образів (радіолокаційні системи, ідентифікація осіб, класифікація сигналів, 3D-реконструкція, розпізнавання об'єктів і багато іншого), аналіз даних датчиків, розпізнавання послідовності (розпізнавання жестів, мови, рукописного та друкованого тексту), медичну

діагностику, фінанси (наприклад, автоматизовані торгові системи), інтелектуальний аналіз даних, візуалізація, машинний переклад, фільтрація соціальних мереж і фільтрація спаму по електронній пошті [11].

ANNs використовувалися для діагностики декількох типів раку і для відмінності високоінвазивних ліній ракових клітин від менш інвазивних ліній, використовуючи тільки інформацію про форму клітин [12].

ANN використовувалися для прискорення аналізу надійності інфраструктур, схильних до стихійних лих, і для прогнозування поселень фундаментів. ANN також використовувалися для побудови моделей чорного ящика в науках про землю: гідрології, моделюванні океану і прибережної інженерії, і геоморфології.

ANNs були використані в сфері кібербезпеки з метою проведення відмінності між законними діями і шкідливими. Наприклад, Машинне навчання використовувалося для класифікації шкідливих програм для Android, для ідентифікації доменів, що належать суб'єктам загроз, і для виявлення URL-адрес, що становлять загрозу безпеці, ведуться дослідження систем ANN, призначених для тестування на проникнення, для виявлення ботнетів, шахрайства з кредитними картками і мережевих вторгнень [13].

ANN були запропоновані в якості інструменту для вирішення рівнянь в приватних похідних у фізиці і моделювання властивостей відкритих квантових систем багатьох тіл.

У дослідженнях мозку ANNs вивчали короткострокову поведінку окремих нейронів. Динаміка нейронних схем виникає в результаті взаємодії між окремими нейронами і того, як поведінка може виникати з абстрактних нейронних модулів, які представляють собою цілісні підсистеми. У дослідженнях розглядалася довгострокова і короткострокова пластичність нейронних систем і їх зв'язок з навчанням і пам'яттю від окремого нейрона до системного рівня.

Часто в медицині можна застосовувати нейронні мережі для визначення об'єкта. Наприклад, замість: «виберіть всі зображення, де є

дорожні знаки» CAPTCHA запитає: «виберіть всі зображення, де є ракова пухлина». Зрозуміло, що звичайні люди не впораються, але якщо це будуть медики або вчені потрібної спеціалізації?

За дуже схожим на CAPTCHA принципом зараз стали створювати нейронні мережі, які можуть знаходити патології. Зрозуміло, що поки це можливо тільки з зображеннями, тому що інші види медичних даних складніше переводити в формати, які розуміє комп'ютер, але тим не менш, це вже величезні масиви інформації.

За даними Національного інституту онкології, щодня в одних тільки Сполучених Штатах діагностується близько 5000 нових випадків раку, а пухлини виявляють часто на зображеннях. Дуже велике значення для лікування раку має рання діагностика і моніторинг стану. А у якого числа людей діагностують переломи і пневмонії? Нейронні мережі можуть справлятися з цими завданнями, і їх поступово починають використовувати для таких рішень.

На рисунку 1.3 видно, як нейронна мережа взяла два шматки зрізу знімка легкого з різними клітинами, після чого перевела пікселі в цифри. Потім це все виявилося оброблено в зв'язках нейронної мережі, в чорному ящику, після чого нейромережа видала відповідь: в одному шматку поширення ракових плям — 273, в іншому — 45, після чого дала точну назву виду раку.

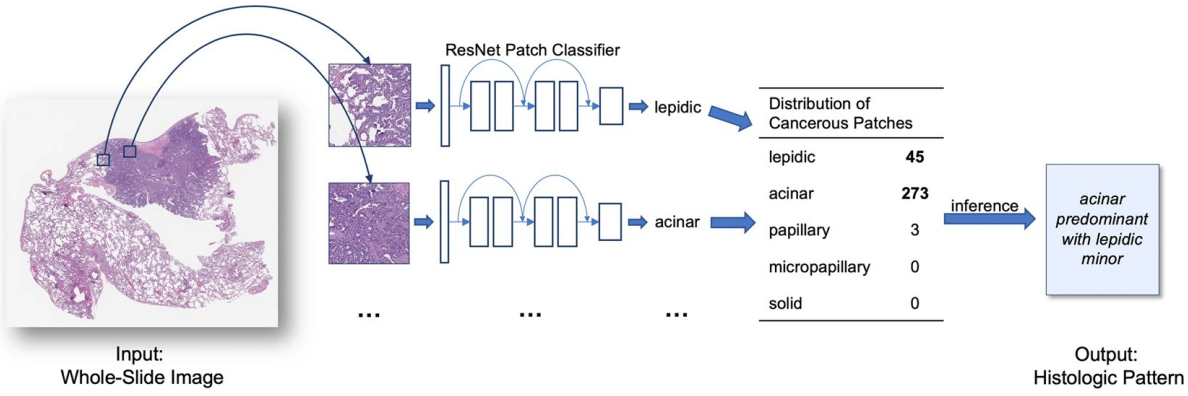

Рисунок 1.3 – Нейронна мережа визначає ракові ділянки в зрізах легені

Безумовно, думка лікарів поки більш авторитетно в порівнянні з “думкою” нейромереж, але як друга думка і як інструмент, що спрощує роботу лікаря, вони ефективні. Це добре видно на прикладі нейронної мережі, яка розпізнає нирково-клітинний рак і ниркові кісти. Ваги і зв'язки були налаштовані вченими; нейромережа навчалася на цифрових даних ультразвукового дослідження близько 100 реальних випадків раку, які вже достовірно визначили. Після цього нейронної мережі дали в обробку 52 випадки (17 злоякісних, 30 кіст і 5 інших) з даних УЗД в лікарні Мемфіса. Навчена нейромережа без помилок визначила 47 випадків, яких не було в даних, що використовуються для навчання. У цьому дослідженні було досить мало інформації для навчання, але тим не менш ефективність роботи мережі була велика.

Найбільші труднощі, чому нейронні мережі поки не можна повсюдно використовувати – це те, що вони ще не до кінця перевірені. Для їх перевірки потрібно дуже багато даних, а не так-то просто створювати бази даних з медичними зображеннями, щоб навчити нейронні мережі розпізнавати хвороби, — це не фотографії котиків і собачок.

Поки в таких серйозних питаннях, як рак, нейронні мережі можуть бути тільки другою думкою. Але зараз прикладається багато зусиль, щоб створювати бази даних, на яких нейромережі можуть краще навчатися і на яких можна їх використовувати.

Не так давно вчені створили нейромережу, яку навчили на 4000 гістологічних знімках, і тепер вона може визначати тип раку легенів з точністю 97% — навіть випереджаючи в цьому живих фахівців. На рисунку 1.4 видно, що висновок фахівців дуже близько збігається з висновком нейромережі: точки — це області, відмічені нейромережею, а суцільні області виділили фахівці.

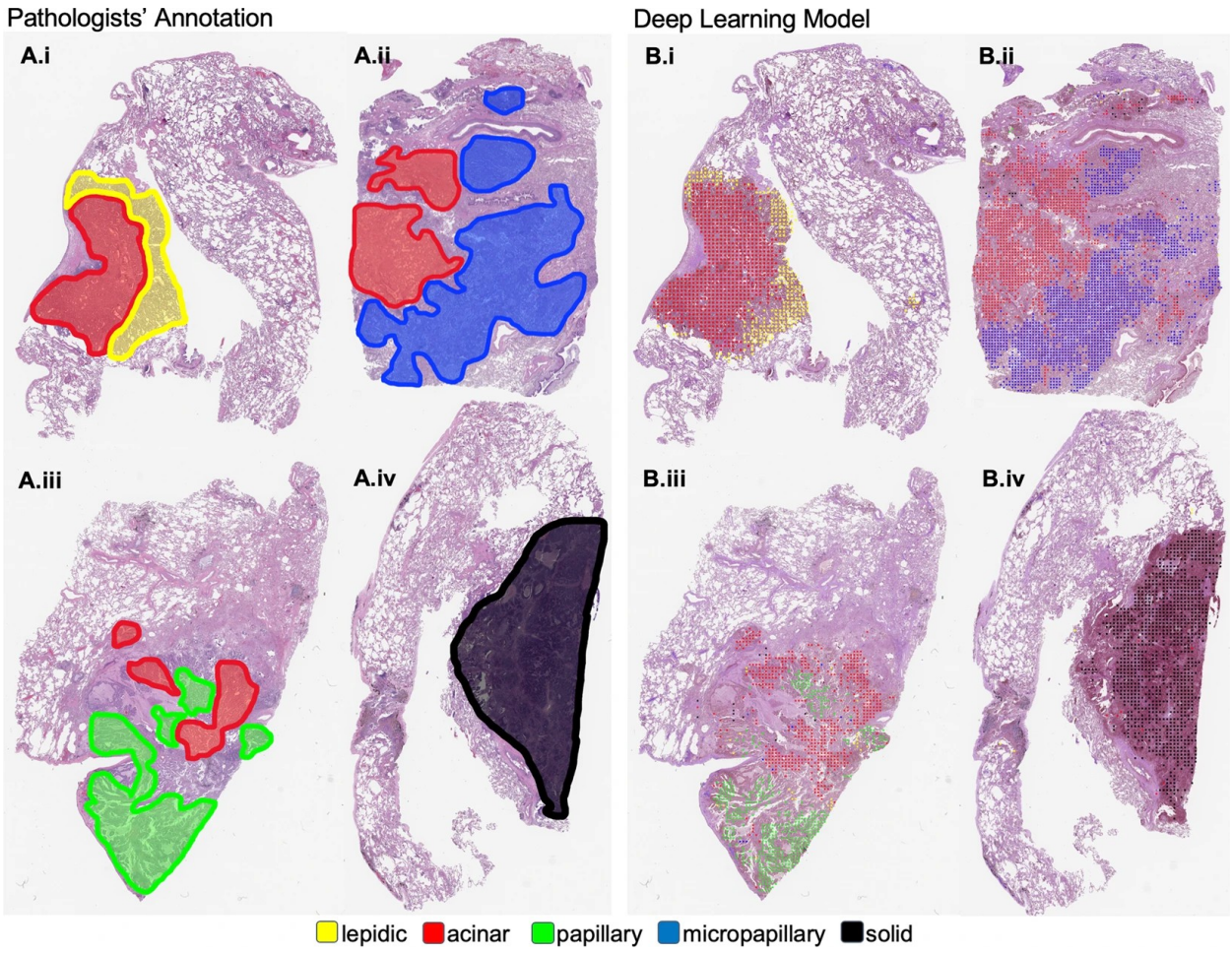

Рисунок 1.4 – Оцінка одних і тих же гістологічних зрізів професійними медиками (зліва) і нейромережею (праворуч)

Висновок щодо раку: нейронні мережі можуть виділяти ті зразки, з якими повинен ознайомитися фахівець, і виступати в якості другої думки. Аналогічним чином застосовують нейронні мережі для моніторингу серйозних хронічних захворювань, наприклад, діабету.

Нейромережі можна застосовувати для самих різних діагностик, не тільки раку, хоча принцип їх роботи зберігається. Взяти хоча б хвороби серця: для визначення діагнозу класичним методом є ЕКГ з подальшою розшифровкою медиком. На жаль, не завжди за отриманим записом ЕКГ можна визначити захворювання. Особливо це відноситься до аритмії — досить небезпечного захворювання, при якому порушується частота серцевих скорочень і їх регулярність. Щоб зафіксувати такий стан, люди іноді

цілодобово ходять з електродами для запису ЕКГ — і все одно в деяких випадках аритмія залишається невиявленою, що небезпечно і сумно.

Зараз вченими Стенфорда на чолі з Ендрю Енджі створена нейронна мережа, яка може навіть точніше і швидше лікарів діагностувати аритмію. Коли таке нововведення впровадять в медичну практику, це сильно прискорить діагностику аритмії і спростить роботу лікарів. Ця нейронна мережа навчалася на 30 000 тридцяти секундних записах ЕКГ, а потім ще на 300 довших.

Крім цього, створюються нейронні мережі для діагностики захворювань хребта. Для навчання однієї з таких нейронних мереж використовувалося 250 наборів записів про стан пацієнтів. В результаті точність тестування виявилася дорівнює 83%, що дуже непогано для такого невеликого набору вихідних даних.

Оцінка ефективності виконання спортивних вправ – це важливий елемент занять спортом і є предметом біомеханіки спорту. Біомеханіка спорту, як правило, визначається як вивчення і застосування фізики і техніки до спортивних тренувань і виступів.

Owusu стверджує, що з обчислювальної точки зору оцінка ефективності передбачає моделювання в специфічному спортивному домені і моделювання процесу її оцінки.

Модель процесу оцінки ефективності є предметнонезалежною, оскільки вона описує механізми для маніпулювання моделями, специфічними для даного спорту. Owusu розглядає модель оцінки ефективності процесу як три етапний лінійний процес: розпізнавання, критика та рекомендації.

З обчислювальної точки зору, розпізнавання виконаного рухового навички (тобто. руху) передбачає придбання та інтерпретації біомеханічних даних, достатніх для опису розглянутого вміння (майстерності).

Для цілей автоматизації процесу оцінки ефективності можна виділити два рівні розпізнавання рухів. Етап розпізнавання на низькому рівні

стосується автоматизації кількісної оцінки біомеханічних даних.

Автоматизація на низькому рівні стала можливою з винаходом систем візуального аналізу руху. Розпізнавання високого рівня присвоює сенс біомеханічним даним, отриманим на низькому рівні. Ключовим питанням розпізнавання на високому рівні є побудова відповідного класифікатора для досліджуваного спорту.

З обчислювальної точки зору етап «Критика» може бути грубо класифікований як діагностика та прогностичне моделювання. Діагностика виявляє будь-які невідповідності між очікуваною ефективністю виконуваних рухових навичок і тим, що спортсмен виконує фактично, і досліджує причини цих розбіжностей.

Сучасні підходи до діагностики в контексті оцінки ефективності спортивної техніки включають розробку систем, заснованих на знаннях (knowledge-based systems-KBS). У прогностичному моделюванні акцент робиться на аналіз того, чого спортсмен може домогтися. Нарешті, на етапі рекомендацій за допомогою AI пропонуються рішення про виправлення виявлених недоліків.

Розглянуті нижче конкретні приклади реалізують один або декількох етапів описаної моделі процесу оцінки ефективності для кластеризації, класифікації, розпізнавання і прогнозування конкретних спортивних даних, таких, як послідовності рухів. Sands описує KBS для моніторингу гімнастичних тренувань.

Система ідентифікує неадекватні фізіологічні та психологічні параметри і попереджає про них. В якості переваг експертної системи автор назвав її здатність виявляти тенденції до отримання травм, ідентифікувати перетренованість, сигналізувати про захворювання до появи яскравої симптоматики і про сплески зростання у спортсменів-підлітків. Sands пропонує базовану на кінезіології експертну систему.

Кінезіологія використовує анатомічну інформацію для визначення того, які м'язи здійснюють рух людини і в якій мірі. За допомогою

запропонованої автором кінезіологічної експертної системи руху (KMES) студенти, визначаючи суглоби, дії і типи напружень, можуть отримати список м'язів, які сприятимуть запитаному руху. Крім того, студент може вибрати конкретний м'яз і тип напруги, і система поверне всі рухи, в яких задіяна дана м'яз.

KMES була написана в PDC PrologTM і має базу знань з 1583 рухів. В Sands і співр. пропонують використовувати нейронні мережі для розпізнавання всіх аспектів виконання гімнастичних вправ і забезпечування зворотного зв'язку.

Перспективність застосування штучних нейронних мереж (ANNs)в області спортивної біомеханіки обговорюється в ряді робіт. У роботі Baca і співр. самоорганізована нейронна мережа (Self Organizing Maps — SOM), що складається з 400 нейронів, навчалася для кластеризації стабільності процесу прицілювання елітними біатлоністами. В аналізували тимчасову динаміку моторного навчання в плаванні брасом. Обробка безлічі даних здійснювалася дворівневою кластеризацією за допомогою алгоритму Fisher-EM.

З точки зору спортивних наук конкретною метою дослідження була оцінка динаміки навчання та оцінка впливу різних умов на динаміку навчання, і далі використання результатів аналізу для оптимізації процесу навчання.

Мета Silva і співр. — виявити фактори, які можуть пояснити ефективність (досягнутий результат) юних плавців в індивідуальному плаванні на дистанціях 200 метрів змішаним стилем і 400 метрів кролем, а також моделювати досягнення в цих дисциплінах за допомогою штучних нейронних мереж (багатошарові персептрони-MLP), і оцінити можливості нейронних мережевих моделей для передбачення ефективності.

У дослідженні використовуються дані 138 плавців (65 чоловіків і 73 жінки) національного рівня. В якості факторів враховувалися змінні чотирьох областей: кінатропометричні (антропометричні показники, композиція тіла і соматотип), функціональні оцінки (сила, гнучкість), специфічні функції

(гідродинамічні, гідростатичні і біоенергетичні характеристики) і плавальна техніка.

На основі результатів нелінійного аналізу з використанням нейронної мережі з прямим зв'язком (feed-forward) створені чотири моделі ефективності. Різниця між прогнозованим і дійсним результатом для тестового набору даних менше 0.8%. Аналогічні результати доповідалися в роботах.

На підставі свого і аналогічних досліджень автори роблять висновок, що інструмент нейронних мереж є вдалим підходом вирішення складних завдань, якими є моделювання ефективності і виявлення талантів в широкому розмаїтті видів спорту і зокрема, в плаванні ілюструються у можливості методів штучного інтелекту в спорті на прикладі силових тренувань.

Дослідники застосовували метод розпізнавання образів (pattern recogniion) для оцінки вправ, виконаних на тренажерах. Збір даних здійснювався з використанням датчиків, прикріплених до тренажерів. З їх допомогою безпосередньо вимірювалися основні дані-діючі сили і переміщення під час тренування, на підставі яких визначалися додаткові характеристики, такі, як періоди часу, швидкості руху, прискорення. Ці параметри застосовувалися для автоматичної оцінки виконуваного вправи. Моделювання здійснювалося за допомогою штучної нейронної мережі та її навчання на основі накопичених даних. Застосовувався метод навчання з учителем (supervised learning). Попередньо оброблений сенсорний вхід був використаний для класифікації та оцінки виконань. Розроблені методики показали задовільні результати.

У практиці такі методи можуть мати вирішальне значення для дослідження якості виконання, для надання допомоги спортсменам і тренерам, для навчання з метою оптимізації і для профілактичних цілей.

Кінцева мета розробників-застосування даної методики в мобільних системах тренувань, що надають спортсменам автоматичну миттєву оцінку їх виконання для здійснення зворотного зв'язку. У дослідженні

використовувалися самоорганізуються карти (self-organizing maps-SOM) для класифікації моделі координації за даними чотирьох баскетболістів, що виконують три різних види кидків з різних відстаней.

Автори відзначають, що SOM є більш об'єктивним методом для пояснення координації рухів у порівнянні з більш традиційними підходами, такими як візуальний аналіз або аналіз часових рядів даних. Schmidt досліджував кінематичний ланцюг штрафного кидка баскетболістів різних рівнів майстерності.

Автор поставив перед собою мету (1) дослідити, яку інформацію можна отримати з патерну руху і (2) вивчити методологічні можливості аналізу образів (pattern analysis). Він виконав тріангуляцію отриманих даних і аналізував їх за допомогою одного з різновидів нейронних мереж — динамічні контрольовані мережі (Dynamically Controlled Networks — DyCoN).

В ході аналізу були виявлені індивідуальні особливості спортсменів, а також фази рухів кидка. Успішно класифіковані патерни кидків і визначена їх стабільність і мінливість. Як можна було припустити на підставі попередніх досліджень, підтвердилася індивідуальність людського руху і за допомогою розпізнавання образів — знайдені патерни руху мали чітко індивідуальну структуру, а також формувалися рівнем майстерності.

У роботі порівнюються можливості нейронних мереж (ANN) і методу опорних векторів (SVM) для моделювання складних кінематичних даних при аналізі структури людської ходьби і бігу. Тестова множина включає дані, зібрані при виконанні цих рухів на біговій доріжці при трьох різних швидкостях. В експерименті брали участь вісім чоловіків-професійні бігуни на середні і довгі дистанції. Результати показують, що SVM дає більш високу точність класифікації, що можна пояснити виконанням оптимізації при поділі даних. У пропонується використовувати імітаційні ігри в настільному тенісі при підготовці спортсменів.

1.7 Теоретичні властивості

Обчислювальна потужність. Теорема про універсальну апроксимацію, що підтверджує універсальність апроксимуючої функції багаторівневого персептрона. Проте у доказі є конструктивні недоліки, що проявляють себе питаннях до потрібної кількості нейронів, ваг, критеріїв навчання та топології мережі.

Під час використання своєрідної рекурентної архітектури з підходящими вагами з'являється можливість отримати властивості універсальної машини Тюрінга, що дозволяє застосовувати кінцеву кількість нейронів та нормальних ліній зв'язку. Окрім цього, застосування ірраціональних змісту у вагах дозволяє розвити машину станом схожу на супер потужну машину Тюрінга.

Місткість. Ця якість моделі дозволяє симулювати будь-яку функцію, що задаються. Дана властивість тісно зв’язана з концепцією складності та розміром даних, що використовуються або зберігаються у мережі. Існують два широковідомі поняття здатності. Інформаційна ємність і вимірювання VC. Поглибленні відомості про інформаційну ємність можна дізнатися з книги Девіда Маккея, що є узагальненням праці Томаса Кавера. Загалом ємність мережі, саме стандартних, а не згорткових можна отримати користуючись чотирма правилами, що є логічним висновком з міркування нейрона як одного з електричних елементів. Інформаційна ємність віддзеркалює функції, що було змодельовані мережею при розгляданні будь-якої інформації як вхідної. Інше поняття являє собою визначення VC. Воно допомагає при знаходженні найбільшого екстремума пропускної спроможності в ідеальних умовах, засновуючись на принципах теорії вимірювань. Ця інформація є інтиною при використанні у якості вхідної інформації у певній формі. Для випадкової вхідної інформації розмірність VC становить 50% від загальної інформаційної складової персептрона. Час від часу знаходження VC для випадкових точок має назву обсяг пам'яті.

Конвергенція. Існують випадки коли моделі не залежно один від одного можуть мати однакове рішення. Для цього є ряд факторів, по-перше, є шанс на існування локальних мінімумів, мають залежність від моделі та функції витрат. По-друге, алгоритм. Що застосовується для підвищення продуктивності може негативно впливати на збіжність, при початковій позиції віддаленої від локальних мінімумів. По-третє, при обробці надто великих даних або параметрів існує ймовірність використання непрактичного методу.

Деякі архітектури мають зрозумілішу поведінку конвергенції на противагу іншим. У випадку наближення ширини мережі до нескінченності, нейронну мережу можна легко описати розширення Тейлора для першого порядку на протязі усього процесу навчання, що отримую поведінку збіжності афінних моделей. Інший випадок, параметри мережі наближаються до нулю, відмічається відповідність нейронної мережі цільовим функціям на різних рівнях частот від низьких до високих.

Така реакція називається спектральним зміщенням, нейронних мереж, інша назва частотний принцип. По своїй природі воно має протилежну дії певних ітераційних чисельних схем, наприклад, метод Якобі. Нейронні мережі с більш глибокого рівня мають більшу с схильність до функцій низької частоти.

Узагальнення і статистика. Для програм головною ціллю яких є утворення системи, що вдало об'єднує невидимі приклади характерним є спроможність до перенавчання. Причиною цього стає той факт, що пропускні властивості мережі значно перевершують необхідні параметри. Найчастіше це спостерігається в заплутаних та системах підвищеного навантаження. Щоб уникнути даного ефекту існують два можливі шляхи. Перший полягає у застосуванні вибіркової перевірки та додаткової спостереження за навантаженням та добором гіперпараметрів, що дозволяє зменшити помилку узагальнення. Другий шлях заснований на застосуванні регуляції. Найчастіше его можна використовувати в байєсівській структурі, коли є

ймовірність на застосування регуляції методом добору апріорної ймовірності на противагу простішим моделям. Також другий шлях можна застосувати при статистичному навчанні, ціллю якого є мінімізація декількох величин, які наближають помилку в невидимій інформації через переоснащення («Емпіричний ризик» і «Структурний ризик») .

Існують нейронній мережі, яким під силу застосовувати формальні статистичні алгоритми під час знаходження правильності навченої моделі за допомогою використання функції вартості, вони мають назву контрольовані нейронній мережі (MSE). Є варіант застосування контрольованих нейронних мереж для оцінки відхилення у складі системи перевірки. Пізніше отримане значення, за умови, що розподіл буде нормальним, можна застосувати для обрахунку інтервалу довіри вихідної інформації. Актуальність цього аналізу буде підтверджуватися на протязі незмінності ймовірності розподілу та самої мережі.

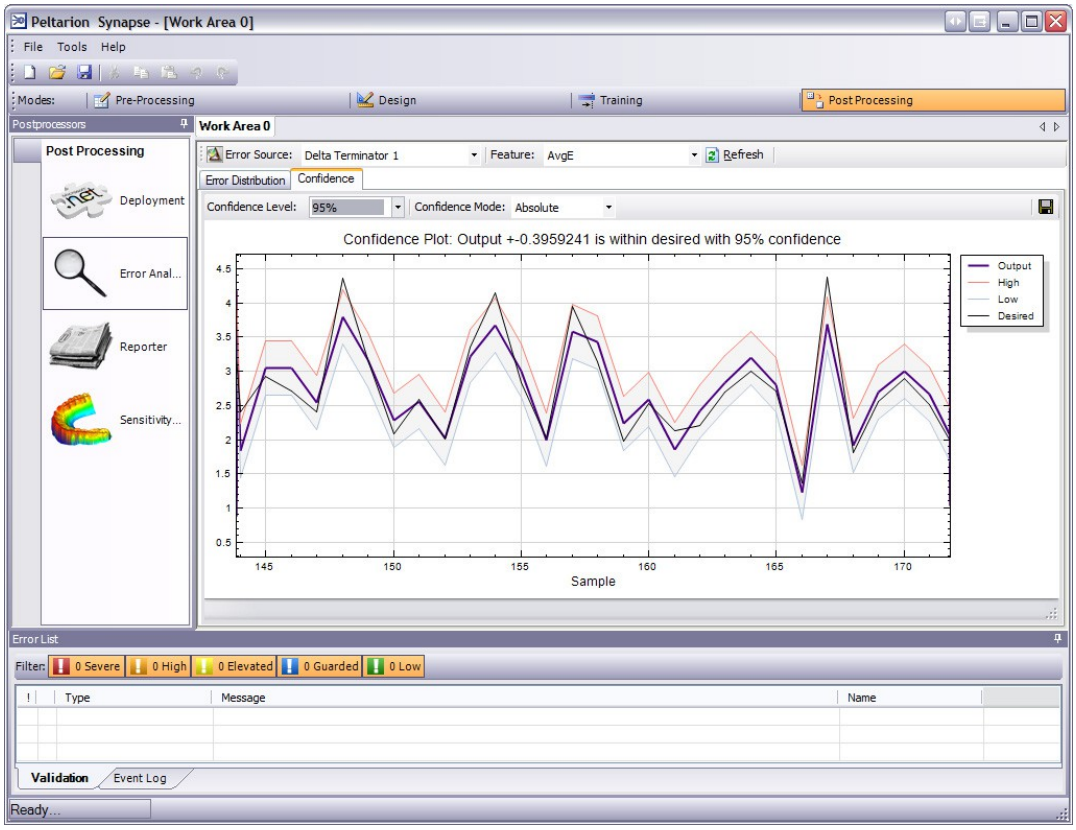

Рисунок 1.3 – Аналіз достовірності нейронної мережі

При використанні функції активації softmax, вихідна інформація може бути розцінена як апостеріорна ймовірність, при підсумовуванні логістичної функції для категоріальних цільових змінних. Це надає найбільшої вигоди при класифікації, тому що призводить до виникнення міри визначеності у класифікації.

1.8 Недоліки нейронних мереж

Розповсюдженими недоліками застосування нейронних мереж ґрунтується на занадто тривалому часі підготовки перед реальним використанням. До складу можливих результатів входять сплутування тренувальних даних використовуючи спосіб чисельної оптимізації, що не значною мірою впливає на мережеві з'єднання, оскільки притримується поєднання у міні-пакети або додаючи рекурсивні алгоритми мінімальних квадратів

Ґрунтовний протест знаходиться у малому втіленні функцій нейронів. Зворотне розповсюдження є критичним рішенням, при відсутності схожої системи в біологічних нейронах. Спосіб кодування даних справжніми нейронами невідомий. Чутливі нейрони змушують реагувати з більшою частотою при активації чутливого сенсору, оскільки зв’язані рухові нейрони одержують потенціали дії з більшою частотою. Винятком є транспортування даних від чутливого нейрона до рухового нейрона.

Головним затвердженням ANN є реалізація інноваційних та більш потужних принципів опрацювання даних. Проте ці принципи визначені не точно. Постійно доводиться, що вони беруться з самої мережі. Це робить можливим представлення звичайної статистичної асоціації, як тренування або розпізнавання. Олександр Дьюдні у 1997 році стверджував, що у нейронних мереж є властивість “щось даром”, призводячи до недостачі зацікавлення у правильності використання обчисленнях систем. Людина не дере участі у знаходженні результату, відповідь отримується “сама собою” і

ніхто нічого не навчився. Нейронні мережі розв’язують значну кількість задач від польоту літака до гри в Го, казав Дьюдні.

Роджер Бріджман стверджував, що нейронні мережі, є не правильними не лише через над велику популярність але і через створення успішної мережі без розуміння її роботи, що робить її даремним науковим ресурсом.

По при все, повідомлення про не лише технологічність науки, Дьюдн, знеславлює нейронні мережі, як не правильну науку, при цьому майже всі хто бере участь у розробці хочуть бути чудовими інженерами. Незрозуміла таблиця, що може розпізнати тільки машина, все одно була би варта того, щоб її мати.

Людський мозок показує великий спектр інваріантності, застосовуючи як невеликі так і глибокі контури. Венг заявляв, що біологічний мозок здатен регулюватися самостійно опираючись на статистичні сигнали і саме через це каскад не знаходить усі можливі залежності.

Найбільші нейронні мережі, що показують чудову ефективність потребують величезну кількість обчислювальних ресурсів, на противагу цьому мозок використовує апаратне забезпечення, що пристосоване до задач з аналізу сигналів через графік нейронів, саме тому процес створення навіть спрощеного нейрона використовуючи архітектуру фон Неймана може витрачати недосяжні масиви пам’яті. Також необхідно використання великої потужності процесора, оскільки майже постійно доводиться транспортувати сигнали через поєднані нейрони.

Шмідхубер стверджував, що нова хвиля популярності до нейроннх мереж 21 століття відбувається через значні здобутки у сфері апаратного забезпечення: у період з 1991 по 2015 рік обчислювальна потужність, а особливо в GPU, виросла майже в мільйон разів, що дозволяє використовувати процес зворотного поширення для тренувальних мереж на декілька рівнів глибше у порівнянні з минулим. Застосування каталізаторів, наприклад, ПЛІС і графічних процесорів, допомагає зменшити час тренування з місяців до днів.

Нейроморфна інженерія або як її ще називають фізична нейронна мережа допомагає знизити апаратні проблеми, шляхом утворення не фон неймановстких чіпів, що дає можливість утворення нейронних мереж прямо на схемах. Ще один з різновидів чіпів оптимальних для використання нейронних мереж називається блоком тензорної обробки (TPU).

Послідовники використання гібридних моделей затверджують, що її застосування допоможе з більшою точністю відтворити механізми людського мозку.

1.9 Нейронні мережі у платформах з обмеженими ресурсами

Глибоке навчання може привнести обчислювальний інтелект в персональні пристрої та пристрої Інтернету речей. Використання моделей глибокого навчання безпосередньо на периферійних пристроях забезпечує більшу економічність, масштабованість і конфіденційність для кінцевих користувачів в порівнянні з використанням віддаленого сервера для виконання обробки. Однак розробка легких нейронних мереж, придатних для такого малопотужного обладнання, зосереджена навколо мобільних телефонів в якості цільової платформи.

Але якщо піти далі і дослідити платформу з більш обмеженими ресурсами – мікроконтролерні блоки (MCU). Мікроконтролери дешеві, широко поширені і орієнтовані на енергоефективні робочі навантаження. Вони пропонують альтернативу розробці спеціально виготовленого на замовлення чіпа, дозволяючи заощадити на витратах на розробку і часу. Однак пристрій, як правило, складається з низькочастотного процесора і всього декількох сотень кілобайт вбудованої пам'яті, і, отже, має значно меншу потужність в порівнянні з мобільними пристроями.

Нейронну мережу можна розглядати як обчислювальний графік, який виражає залежності між окремими операціями (також званими шарами або операторами). Оператор, наприклад, двовимірна згортка або додавання,

приймає один або кілька вхідних тензорів і видає один вихідний сигнал. Сучасні платформи глибокого навчання оптимізують графік обчислень мережі для попереднього виведення, наприклад, об'єднуючи суміжні оператори і складаючи шари пакетної нормалізації в попередні лінійні операції. Виконання виконується шляхом обчислення одного оператора за раз в топологічному порядку вузлів графа.

Оператору потрібно, щоб буфери для його вхідних і вихідних даних були присутні в пам'яті до початку його виконання. Як тільки оператор завершить виконання, пам'ять, зайнята його входами, може бути відновлена (якщо не використовується в іншому місці), і вихідний буфер в кінцевому підсумку буде використовуватися в якості вхідних даних для інших операторів. Деякі дослідження показують перспективу визначення робочого набору як набору тензорів, які необхідно зберігати в пам'яті в будь-якій заданій точці виконання. Це включає в себе вхідні та вихідні тензори чекаючого оператора та інші тензори, які вже були створені і повинні бути збережені в пам'яті для наступних операторів.

Класичні архітектури нейронних мереж, такі як оригінальний багатошаровий персептрон, AlexNet, VGG, складаються з лінійної послідовності шарів, які ітеративно застосовуються для перетворення вхідних даних. Однак пізніші архітектури, такі як ResNet, Inception, NASNet, вводять розходяться шляхи обробки, в яких один і той же тензор може оброблятися декількома шарами, тобто їх графік обчислень більше не лінійний і має гілки. Це означає, що програмне забезпечення для виведення може мати кілька операторів, доступних для виконання на будь-якому заданому кроці.

Розробка компактних моделей є активною темою досліджень в області глибокого навчання, хоча, як правило, в умовах менш жорстких обмежень, ніж у апаратного забезпечення мікроконтролерів. Можна отримати нейронну мережу меншого розміру, використовуючи розкладання по шарах, обрізку, квантування і бінаризацію, дистиляцію або використання розрідженості.

Популярні мобільні архітектури CNN включають в себе MobileNet і ShuffleNet. MNasNet і EfficientNet розробляють алгоритми пошуку архітектури для проектування мережі в рамках певної кількості операцій з плаваючою комою або бюджету пам'яті. Зокрема, Федоров та ін. включають максимальний обсяг робочої пам'яті оператора в свою мету оптимізації.

Відносно недостатньо вивчений набір методів включає складні стратегії оцінки для частин мережі для економії пам'яті під час виконання. Наприклад, автори MobileNet відзначають, що будівельний блок їх моделі має канальну операцію, вихід якої накопичується в іншому тензорі, що дозволяє обробляти вхідний тензор по частинах. Крім того, Альвані та ін. пропонують взагалі не матеріалізувати вихідний тензор великої операції згортки в пам'яті і обчислювати його окремі вихідні елементи в міру необхідності наступними операторами.

Розвитку моделей машинного навчання з низьким енергоспоживанням сприяють саміт TinyML і конкурс слів візуального пробудження, які шукали високопродуктивні CNN розміром з Мікроконтролер для виявлення людей. Компактні моделі глибокого навчання були створені для використання на носимих пристроях і для визначення ключових слів. Побоювання щодо використання пам'яті нейронними мережами та переміщення даних під час виконання також обговорюються в літературі з проектування мікросхем прискорювачів нейронних мереж.

Результати показують, що використання іншого порядку виконання оператора для виведення нейронної мережі може призвести до того, що раніше не розгортаються моделі будуть відповідати обмеженням пам'яті апаратного забезпечення MCU. Оператори переупорядкування можуть бути реалізовані лише в програмному забезпеченні для виведення, що робить його ортогональним більшості інших підходів до стиснення мережі, які, ймовірно, вже використовувалися для створення моделі розміром з Мікроконтролер. На відміну від мобільних і серверних платформ, апаратним засобам MCU часто не вистачає пам'яті для статичного попереднього виділення всіх тензорних

буферів мережі, що вимагає програмного забезпечення виведення для підтримки динамічного виділення пам'яті. Проте є дослідження, що показують простоту стратегії дефрагментації є життєздатним варіантом з невеликими накладними витратами. Однак, коли розклад виконання відомо заздалегідь, може бути попередньо обчислено оптимальне розміщення буфера тензора в пам'яті. Наявність способу точного обчислення пікового використання пам'яті для моделей зі складними графіками обчислень принесло б користь процедурам пошуку нейронної архітектури (NAS). Алгоритм може бути розширений для підтримки різних прийомів економії пам'яті: наприклад, якщо один з входів оператора додавання не використовується в іншому місці, результат може бути накопичений в ньому, усуваючи необхідність у вихідному буфері.

Мікроконтролери є життєздатною платформою для запуску програм глибокого навчання, якщо розробник моделей може подолати обмеження, що накладаються невеликою пам'яттю і сховищем.

1.10 Формування вимог до комп’ютерної системи

Перед розробкою нейронної мережі для подальшого її використання на мікроконтролері слід брати до уваги характеристики, що можуть впливати на правильну роботу. До них відносяться: вибір моделі, алгоритм навчання, надійність.

Від вибору моделі залежить подання даних і додатків. Надмірно складні моделі приведуть до повільного тренування.

Між алгоритмами тренування є багато компромісів. Майже усі вони будуть справна працювати з вірними гіперпараметрами для навчання на певному наборі інформації. Але вибір і конфігурація алгоритму навчання на невидимих даних вимагає значних експериментів.

Якщо модель, функція витрат і алгоритм навчання обрані належним чином, результуюча ANN може стати надійною.

Не менш важливими факторами, що впливають на вимоги є майбутнє використання на мікроконтролеру, тобто на платформі з обмеженими ресурсами, що також має бути враховано.

До вимог, що стосуються вибору мікроконтролера слід віднести, можливість роботи з нейронними мережами, відносну дешевизну та доступність.

1.11 Вибір мікроконтролера

Виходячи з усього оглянутого та вимог до роботи для проектування пристрою, було вирішено обрати мікроконтроллер STM32G474RET6, що интегрований на налагоджувальній платі NUCLEO-G474RE для зручності побудови макету.

Налагоджувальна плата STM32 Nucleo – 64 надає простий і пристосований спосіб тестувати нові рішення і створювати прообрази, роблячи вибір з окремих характеристик ефективності та енергозбереження, що дає мікроконтролер STM32. При використанні сполучних плат зовнішній SMPS дорволяеє суттево зменьшити енергоспоживання в режимі запуску. Підтримка застосування ARDUINO Uno V3 та заголовків ST morpho дозволяє легко розширювати функціональні можливості відкритої платформи розробки STM32 Nucleo з широким вибором спеціалізованих екранів.

Плата STM32 Nucleo – 64 не вимагає окремого пристрою для запису, оскільки вона інтегрує відладчик/програматор ST-LINK. Вона поставляється з комплексними бібліотеками вільного програмного забезпечення STM32 і прикладами, доступними в пакеті MCU STM32Cube.

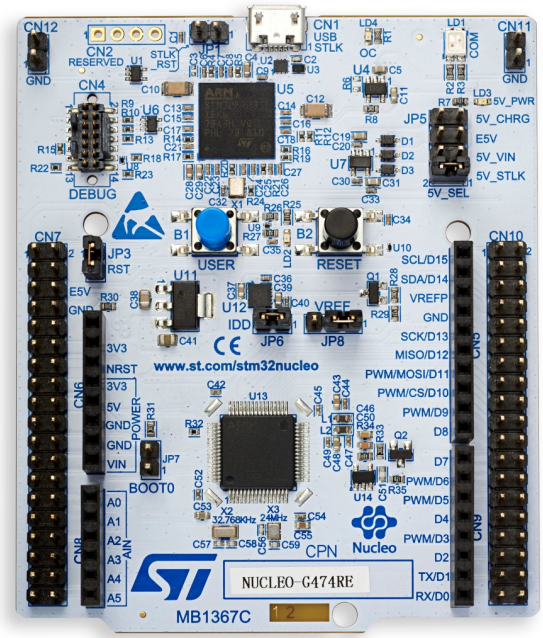

Рисунок 1.5 – Налагоджувальна плата NUCLEO-G474RE

На платі присутні мікроконтролер STM32 в пакеті LQFP64, один користувальницький світлодіод, що спільно використовується з ARDUINO, одна програмована кнопка, кнопка скидання в початкове значення, кварцовий генератор частотою 32,768 кГц,

Роз'єми плати: Роз'єми розширення ARDUINO Uno V3 роз'єми розширення morpho роз'єми розширення для повного доступу до всіх входів/ОС STM32, зовнішній спеціальний роз'єм для експериментів SMPS Micro-AB або USB-роз'єм Mini-AB для налагоджувального роз'єму ST-LINKMIPI

Також до основних переваг використання саме цього рішення відносяться:

– різноманітні способи живлення(ST-LINK, USB VBUS або зовнішні джерела);

– вмонтований відладчик/програматор ST-LINK з можливістю повторного перерахування USB;

– комплексні бібліотеки вільного програмного забезпечення та

варіанти використання, що входять до пакету MCU STM32Cube;

– підтримка великої кількості поєднаних середовищ розробки (IDE), до складу яких вхоядть iar Embedded Workbench, MDK-ARM і STM32CubeIDE.

Таким чином ця плата містить Мікроконтролер STM32G474RET6 в корпусі з 64 висновками. Максимальна частота ядра процесора – 170 МГц, об'єм Flash-пам'яті – 512 кбайт, об'єм статичної пам'яті – 128 Кбайт. Також плата містить вбудований високошвидкісний відладчик/програматор ST-LINK/V3, що дозволяє виконувати програмування і налагодження плати без використання будь-яких зовнішніх налагоджувальних засобів.

Мікроконтролери сімейства STM32F4 володіють високою продуктивністю і багатим набором периферійних пристроїв. Також виробник пропонує широкий набір програмних і налагоджувальних засобів для розробки, в тому числі рішень в області нейронних мереж, тому представник саме цього сімейства був обраний для практичного розгляду.

1.12 Мета та задачі кваліфікаційної роботи

Метою кваліфікаційної роботи є реалізації згорткової нейронної мережі для розпізнавання людської діяльності на базі налагоджувальної плати NUCLEO-G474RE шляхом портування нейронної мережі, навченої на класичній ЕОМ.

Для досягнення поставленої мети були сформульовані і вирішені наступні основні завдання:

– розпізнавання активності за допомогою набору даних;
– розробка згорткової нейронної мережі за допомогою Keras;
– підвищення точності додатковими налаштуваннями;
– квантування мережі у форматі 8-бітних чисел з фіксованою точкою;
– підготовка проекту для портування на мікроконтролер;

– перевірка данних на прошитому мікроконтролері.

2 ВИБІР ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ЗАДАЧИ

2.1 Відкрита нейромережева бібліотека Keras

Keras – відкрита бібліотека, написана на мові Python, що забезпечує взаємодію з штучними нейронними мережами. Вона являє собою надбудову над фреймворком TensorFlow. До версії 2.3 підтримувалися різні версії нейромережевих бібліотек, такі як TensorFlow, Microsoft Cognitive Toolkit, Deeplearning4j, і Theano. Бібліотека націлена на оперативну роботу з мережами глибинного навчання, при цьому спроектована так, щоб бути компактною, модульною і розширюваною. Головним автором Keras є Франсуа Шолле інженер Google, що створював її як частину досліджень проекту ONEIROS.

Keras – це API для глибокого навчання, написаний на Python, що працює на вершині платформи машинного навчання TensorFlow. Він був розроблений з упором на забезпечення можливості швидкого експериментування. Здатність якнайшвидше перейти від ідеї до результату є ключем до проведення хороших досліджень. До головних переваг Keras відносять: простоту – він знижує когнітивне навантаження розробника, дозволяючи зосередитися на тих частинах проблеми, які дійсно важливі; гнучкість – Keras дотримується принципу поступового розкриття складності: прості робочі процеси повинні бути швидкими і легкими, в той час як довільно просунуті робочі процеси повинні бути можливі за допомогою чіткого шляху, який ґрунтується на тому, що вже відомо; потужність – він забезпечує високу продуктивність і масштабованість в галузі через використання організаціями та компаніями, включаючи NASA, YouTube або Waymo.

Планувалося що Google буде підтримувати Keras в основній бібліотеці TensorFlow, проте Шолле виділив Keras в окрему надбудову, так як згідно

концепції Keras є скоріше інтерфейсом, ніж наскрізною системою машинного навчання. Keras надає високорівневий, більш інтуїтивний набір абстракцій, який робить простим формування нейронних мереж, незалежно від використовуваної в якості обчислювального бекенда бібліотеки наукових обчислень.

TensorFlow2 – це комплексна платформа машинного навчання з відкритим вихідним кодом. Це дуже схоже на окремий рівень інфраструктури для диференційованого програмування. Він поєднує в собі чотири ключові здібності:

- ефективне виконання низькорівневих тензорних операцій на процесорі, графічному процесорі або TPU;
- обчислення градієнта довільних диференційованих виразів;
- масштабування обчислень на безліч пристроїв, таких як кластери з сотень графічних процесорів;
- експорт програм (“графіків”) в зовнішні середовища виконання, такі як сервери, браузери, мобільні і вбудовані пристрої;

Keras – це високорівневий API TensorFlow 2: доступний, високопродуктивний інтерфейс для вирішення завдань машинного навчання з акцентом на сучасне глибоке навчання. Він надає необхідні Абстракції та будівельні блоки для розробки та доставки рішень для машинного навчання з високою швидкістю ітерації.

Keras дозволяє повною мірою використовувати масштабованість і крос-платформні можливості TensorFlow 2: існує можливість запуску Keras на TPU або на великих кластерах графічних процесорів, а також експортувати моделі Keras для запуску в браузері або на мобільному пристрої.

2.2 Пакет розширення X-CUBE-AI для STM32CubeMX

Екосистема STM32CubeMX – це комплексне програмне рішення для мікроконтролерів і мікропроцесорів STM32. Вона створена як повне і

безкоштовне середовище розробки для STM32. Але також є можливість використання сторонньої IDE, включаючи Keil або IAR, в яку можна легко інтегрувати різні компоненти, такі як STM32CubeMX, STM32CubeProgrammer або STM32CubeMonitor.

STM32Cube являє собою комбінацію програмних засобів і вбудованих програмних бібліотек до складу яких входять:

– набір програмних засобів для повного циклу розробки проекту на ПК;
– вбудовані програмні блоки для мікроконтролерів і мікропроцесорів STM32, що забезпечують різні функціональні можливості.

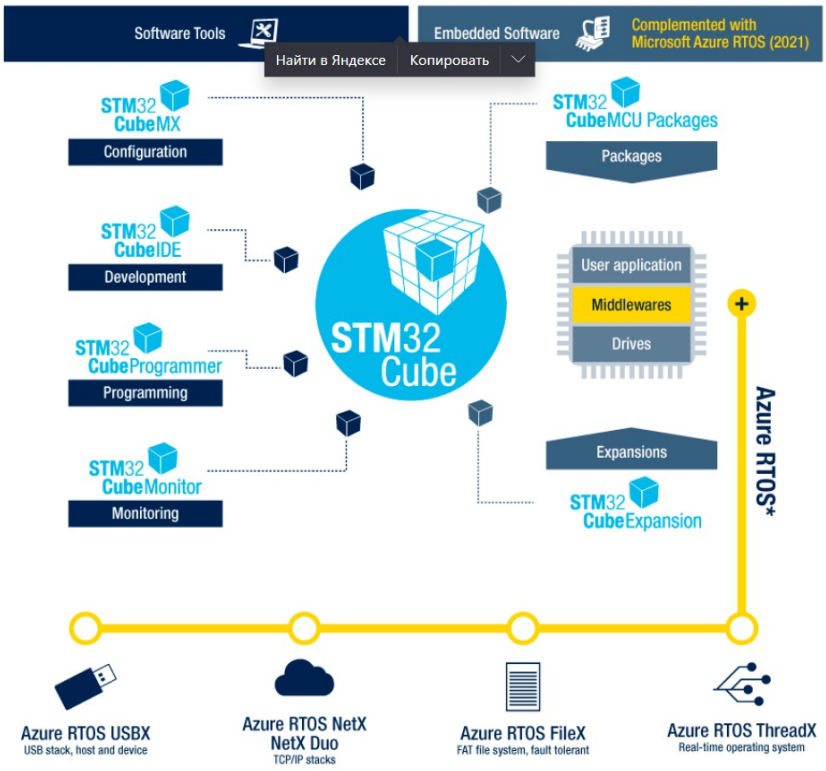

Рисунок 2.1 –Екосистема STM32Cube

До складу STM32Cube входять програмні інструменти для кожного етапу розробки.

STM32CubeMX – інструмент налаштування для будь-якого пристрою

STM32. Його головною задачею є генерація коду ініціалізації C для ядер Cortex-M і генерація джерела дерева пристроїв Linux для ядер Cortex-A за допомогою простого графічного інтерфейсу.

STM32CubeIDE – інтегроване середовище розробки. Воно ґрунтується на рішеннях з відкритим кодом, таких як Eclipse або набір інструментів GNU C/C++, це середовище розробки включає функції складання звітів і розширені функції налагодження. Також є можливість інтеграції додаткових функцій, присутніх в інших інструментах екосистеми, таких як ініціалізація HW і SW і генерація коду з STM32CubeMX.

STM32CubeProgrammer – інструмент програмування. Він забезпечує просте у використанні і ефективне середовище для читання, запису і перевірки пристроїв і зовнішніх запам'ятовуючих пристроїв за допомогою широкого спектру доступних засобів зв'язку (JTAG, SWD, UART, USB DFU, I2C, SPI, CAN і т.д.).

До складу сімейства інструментів також входять потужні інструменти моніторингу, які допомагають розробникам точно налаштовувати поведінку і продуктивність своїх додатків в режимі реального часу.

Вбудоване програмне забезпечення STM32Cube допомагає прискорити розробку проекту:

- пакети MCU і MPU STM32Cube, призначені для кожної серії STM32;
- пакети розширення STM32Cube для прикладних рішень.

Пакети MCU і MPU STM32Cube містять всі необхідні вбудовані програмні блоки для роботи з доступним набором периферійних пристроїв STM32. Вони включають драйвери, проміжне програмне забезпечення та безліч прикладів коду, що використовуються в різних реальних випадках використання.

Пакети розширення STM32Cube для прикладних рішень доповнюють і розширюють пакетний склад MCU STM32Cube додатковими блоками вбудованого програмного забезпечення.

STMicroelectronics пропонує повний набір рішень для штучного інтелекту: програмне забезпечення та інструменти, повністю інтегровані з повним портфелем апаратних засобів, що допомагають ефективно розробляти проекти в області штучного інтелекту.

«Edge AI» швидко набуває популярності, що дозволяє вважати його стандартом завдяки вирішальній перевазі: завдяки функціям машинного навчання, вбудованим в мікроконтролер, є можливість аналізувати і інтерпретувати дані локально. Це дозволяє усувати затримки з точки зору прийняття рішень і може діяти швидко і безпечно, оскільки дані залишаються на периферії і не потребують передачі в хмару. Обробка та аналіз виконуються в дуже енергоефективному мікроконтролері, в той час як архітектури, що використовують нейронні мережі, попередньо навчені на серверах, все ще дуже енергоємні.

Рішення для штучного інтелекту повністю інтегровані з повним портфелем STM32 і використовують різні плати розробки та оцінки.

Широкий асортимент продуктів для мікроконтролерів ST охоплює надійні, недорогі 32-розрядні мікроконтролери Cortex - M на базі Arm з широким вибором периферійних пристроїв. Його широта гарантує, що інженери-проектувальники знайдуть поєднання продуктивності, енергоефективності та безпеки, яке потрібно для їх застосування.

Від мікроконтролерів наднизької потужності для аналізу часових рядів до високопродуктивних мікроконтролерів для більш складної обробки, такої як комп'ютерне бачення, включаючи основні мікроконтролери і бездротові мікроконтролери для вбудовування стека підключень в дизайн.

3 РЕАЛІЗАЦІЯ МОДЕЛІ НЕЙРОНОЇ МЕРЕЖІ

3.1 Розпізнавання активності за допомогою набору даних смартфонів

Розпізнавання людської діяльності, або скорочено HAR – це проблема прогнозування того, що робить людина, на основі відстеження його руху за допомогою датчиків.

Стандартним набором даних для розпізнавання людської діяльності є «Набір даних для розпізнавання активності з використанням смартфонів», доступний в 2012 році.

Він був підготовлений та наданий Давідом Ангітом та ін. з Університету Генуї, Італія, і повністю описаний у своїй статті 2013 року «Набір даних суспільного надбання для розпізнавання людської діяльності з використанням смартфонів». Набір даних був змодельований за допомогою алгоритмів машинного навчання в їх статті 2012 року під назвою «Розпізнавання людської діяльності на смартфонах з використанням багатокласової апаратно-орієнтованої векторної машини підтримки».

Набір даних є доступним і його можна безкоштовно завантажити з репозиторію «Human Activity Recognition Using Smartphones Data Set, UCI Machine Learning Repository».

Дані були зібрані у 30 суб'єктів у віці від 19 до 48 років, які виконували одну з шести стандартних дій, носячи смартфон на поясі, який записував дані про рух. Було записано відео кожного суб'єкта, що виконує дії, і дані про рух були позначені вручну з цих відео.

Було виконано шість наступних заходів:

- ходьба;
- підойм по сходах;
- спуск по сходах;
- сидіння;

– стояння;
– біг.

Записаними даними про рух були дані акселерометра x, y і z (лінійне прискорення) і гіроскопічні дані (кутова швидкість) зі смартфона, зокрема Samsung Galaxy S II. Спостереження реєструвалися з частотою 50 Гц (тобто 50 точок даних в секунду). Кожен випробуваний виконував послідовність дій двічі, один раз з пристроєм на лівій стороні і один раз з пристроєм на правій стороні.

Вихідні дані недоступні. Замість цього була доступна попередньо оброблена версія набору даних. Етапи попередньої обробки включали:

– попередня обробка акселерометра і гіроскопа з використанням шумових фільтрів;
– поділ даних на фіксовані проміжки тривалістю 2,56 секунди (128 точок даних) з 50% перекриттям;
– поділ даних акселерометра на гравітаційну (загальну) і компоненти руху тіла.

До даних проміжків була застосована розробка функцій, і була надана копія даних з цими розробленими функціями.

З кожного проміжку був витягнутий ряд тимчасових і частотних характеристик, зазвичай використовуваних в області розпізнавання людської діяльності. В результаті вийшов вектор об'єктів з 561 елемента.

Набір даних був розділений на навчальні (70 %) і тестові (30 %) набори на основі даних з предметів, наприклад, 21 предмет для навчання і дев'ять для тестування.

Результати експерименту з машиною опорних векторів, призначеної для використання на смартфоні (наприклад, арифметика з фіксованою комою), призвели до точності прогнозування 89% в тестовому наборі даних, досягнувши аналогічних результатів у незміненій реалізації SVM.

Набір даних знаходиться у вільному доступі і може бути завантажений з репозиторію машинного навчання UCI.

Дані надаються у вигляді одного zip-файлу розміром близько 58 мегабайт.

3.2 Розробка згорткової нейронної мережі за допомогою Keras

Згорткові нейромережеві моделі були розроблені для задач класифікації зображень, коли модель вивчає внутрішнє представлення двовимірних вхідних даних в процесі, званому навчанням ознаками.

Цей же процес може бути використаний для одновимірних послідовностей даних, наприклад, у разі прискорення і гіроскопічних даних для розпізнавання людської діяльності. Модель вчиться витягувати об'єкти з послідовностей спостережень і тому, як зіставляти внутрішні об'єкти з різними типами діяльності.

Перевага використання CNN для класифікації послідовностей полягає в тому, що вони можуть безпосередньо витягувати уроки з необроблених даних часових рядів і, в свою чергу, не вимагають спеціальних знань в області для ручного проектування вхідних функцій. Модель може вивчити внутрішнє представлення даних часових рядів і в ідеалі досягти продуктивності, порівнянної з моделями, придатними для версії набору даних з інженерними функціями.

Першим кроком розробки моделі було завантаження необробленого набору даних в пам'ять.

У вихідних даних є три основні типи сигналів: загальне прискорення, прискорення тіла і гіроскоп тіла. Кожен з них має три осі даних. Це означає, що для кожного тимчасового кроку існує в цілому дев'ять змінних.

Крім того, кожна серія даних була розділена на перекриваються проміжки даних тривалістю 2,65 секунди, або 128 тимчасових кроків. Ці проміжки даних відповідають інтервалам інженерних функцій (рядків) у попередньому розділі.

Це означає, що один рядок даних містить (128 * 9), або 1152 елементів. Це трохи менше, ніж удвічі перевищує розмір векторів 561 елемента в попередньому розділі, і цілком ймовірно, що є деякі надлишкові дані.

Сигнали зберігаються в каталозі Inertial Signals в підкаталогах train і test. Кожна вісь кожного сигналу зберігається в окремому файлі, що означає, що кожен з наборів даних тренувань і тестів має дев'ять вхідних файлів для завантаження і один вихідний файл для завантаження. Можно розподілити завантаження цих файлів по групах, враховуючи узгоджену структуру каталогів і угоди про іменування файлів.

Вхідні дані представлені у форматі CSV, де стовпці розділені пробілами. Кожен з цих файлів може бути завантажений у вигляді масиву NumPy. Функція load_file() завантажує набір даних із зазначенням шляху до файлу і повертає завантажені дані у вигляді масиву NumPy.

Лістинг 3.1 – Функція load_file()

```
def load_file(filepath):
      read_csv(filepath, header=None, delim_whitespace=True)
      dataframe
```

Далі можно завантажити всі дані для даної групи (навчати або тестувати) в єдиний тривимірний масив NumPy, де розміри масиву є [samples, time steps, features].

Щоб зробити це більш зрозумілим, існує 128 тимчасових кроків і дев'ять функцій, де кількість вибірок дорівнює кількості рядків у будь-якому заданому файлі даних необробленого сигналу.

Функція load_group() реалізує цю поведінку. Функція stack() NumPy дозволяє складати кожен з завантажених 3D-масивів в один 3D-масив, де змінні розділені по третьому виміру (об'єктам).

Лістинг 3.2 – Функція load_group()

```
def load_group(filenames, prefix=''):
	loaded = list()
	for name in filenames:
		data = load_file(prefix + name)
		loaded.append(data)
	loaded = dstack(loaded)
	return loaded
```

Використовуючи цю функцію можна завантажити всі дані вхідного сигналу для даної групи, таких як навчання або тестування.

Функція load_dataset_group() завантажує всі дані вхідного сигналу і вихідні дані для однієї групи, використовуючи узгоджені принципи про іменування між каталогами train і test.

Лістинг 3.3 – Функція load_dataset_group()

```
def load_dataset_group(group, prefix=''):
	filepath = prefix + group + '/Inertial Signals/'
	# load all 9 files as a single array
	filenames = list()
	# total acceleration
	filenames += ['total_acc_x_'+group+'.txt',
	'total_acc_y_'+group+'.txt', 'total_acc_z_'+group+'.txt']
	# body acceleration
	filenames += ['body_acc_x_'+group+'.txt',
	'body_acc_y_'+group+'.txt', 'body_acc_z_'+group+'.txt']
	# body gyroscope
	filenames += ['body_gyro_x_'+group+'.txt',
	'body_gyro_y_'+group+'.txt', 'body_gyro_z_'+group+'.txt']
	# load input data
	X = load_group(filenames, filepath)
	# load class output
	y = load_file(prefix + group + '/y_'+group+'.txt')
	return X, y
```

Нарешті, можна завантажити кожен з навчальних і тестових наборів даних.

Вихідні дані визначаються як ціле число для номера класу. Необхідно одним гарячим способом закодувати ці цілі числа класів, щоб дані підходили для підгонки нейромережевої багатокласової моделі класифікації. Можна зробити це, викликавши функцію to_categorical().

Функція load_dataset() реалізує цю поведінку і повертає навчальні та тестові елементи X і Y, готові для підгонки і оцінки певних моделей.

Лістинг 3.4 – Функція load_dataset()

```
def load_dataset(prefix=''):
      # load all train
      trainX, trainy = load_dataset_group('train', prefix +
                                          'HARDataset/')
      print(trainX.shape, trainy.shape)
      # load all test
      testX, testy = load_dataset_group('test', prefix + 'HARDataset/')
      print(shape, testyshape)
      # zero-offset class values
      trainy = trainy - 1
      testy = testy - 1
      # one hot encode y
      trainy = to_categorical(trainy)
      testy = to_categorical(testy)
      print(trainX.shape, trainy.shape, testX.shape, testyshape)
      return trainX, trainy, testX, testy
```

Тепер, коли є дані, завантажені в пам'ять, готові для моделювання, можна визначити, підігнати і оцінити модель CNN.

Можна визначити функцію з назвою evaluate_model(), яка приймає навчальний та тестовий набір даних, поміщає модель у навчальний набір даних, оцінює її в тестовому наборі даних і повертає оцінку продуктивності моделей.

По-перше, необхідно визначити модель CNN, використовуючи бібліотеку глибокого навчання Keras. Модель вимагає тривимірного введення з [samples, time steps, features].

Саме так було завантажено дані, де один зразок – це один проміжок даних часових рядів, кожне такий інтервал містить 128 тимчасових кроків, а часовий крок містить дев'ять змінних або функцій.

Результатом для моделі був шестиелементний вектор, що містить ймовірність того, що даний проміжок належить кожному з шести типів дій.

Ці вхідні і вихідні вимірювання необхідні при підгонці моделі, і є можливість “витягти” їх з наданого навчального набору даних.

Лістинг 3.5 – Ініціалізація моделі CNN, використовуючи бібліотеку глибокого навчання Keras.

```
      n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], 
trainy.shape[1]
```

Для простоти модель визначається як послідовна модель Keras.

Визначається модель як така, що має два шари CNN, а потім шар відсіву для регуляризації, а потім шар об'єднання. Зазвичай шари CNN визначаються групами по два, щоб дати моделі хороші шанси на вивчення функцій з вхідних даних. CNN навчаються дуже швидко, тому рівень відсіву покликаний допомогти уповільнити процес навчання і привести до кращої остаточної моделі. Шар об'єднання зменшує вивчені об'єкти до 1/4 їх розміру, об'єднуючи їх тільки з найважливішими елементами.

Після CNN і об'єднання вивчені об'єкти згладжуються в один довгий вектор і проходять через повністю підключений шар перед вихідним шаром, використовуваним для прогнозування. Повністю підключений шар в ідеалі забезпечує буфер між вивченими об'єктами і висновком з метою інтерпретації вивчених об'єктів перед виконанням прогнозу.

Для цієї моделі було використано стандартну конфігурацію з 64 паралельних карт об'єктів і розмір ядра 3. Карти об'єктів – це кількість разів, коли вхідні дані обробляються або інтерпретуються, тоді як розмір ядра – це кількість тимчасових кроків введення, що враховуються при зчитуванні або обробці вхідної послідовності на картах об'єктів.

Ефективна версія Adam стохастичного градієнтного спуску буде використовуватися для оптимізації мережі, і буде використовуватися категоріальна функція втрати перехресної ентропії, враховуючи, що розглядалося проблема багатокласової класифікації.

Лістинг 3.6 – Визначення моделі

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(n_timesteps,n_features)))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(Dropout(0.5))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Модель підходить для фіксованого числа епох, в даному випадку 10, і буде використовуватися пакет розміром 32 вибірки, в якому 32 вікна даних будуть представлені моделі до оновлення ваг моделі.

Як тільки модель підходить, вона оцінюється в тестовому наборі даних і повертається точність моделі відповідності в тестовому наборі даних.

Лістинг 3.7 – Повна функція evaluate_model()

```
# fit and evaluate a model
def evaluate_model(trainX, trainy, testX, testy):
	verbose, epochs, batch_size = 0, 10, 32
	n_timesteps, n_features, n_outputs = trainX.shape[1],
trainX.shape[2], trainy.shape[1]
	model = Sequential()
	model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(n_timesteps,n_features)))
	model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
	model.add(Dropout(0.5))
	model.add(MaxPooling1D(pool_size=2))
	model.add(Flatten())
	model.add(Dense(100, activation='relu'))
	model.add(Dense(n_outputs, activation='softmax'))
	model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
	# fit network
	model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
	# evaluate model
	_, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose=0)
	return accuracy
```

Для оцінки роботи моделі недостатньо однієї оцінки. Оскільки нейронні мережі є стохастичними, а це означає, що при навчанні однієї і тієї

ж конфігурації моделі на одних і тих же даних буде отримана інша конкретна модель.

Це особливість мережі в тому, що вона надає моделі адаптивну здатність, але вимагає дещо складнішої оцінки моделі.

Тому оцінка моделі була проведена кілька разів, а потім підведено підсумки продуктивності моделі по кожному з цих запусків. Було викликано функцію evaluate_model() в цілому 10 разів. Це призвело до сукупності оцінок моделі, які було необхідно підсумувати.

Лістинг 3.8 – Виклик функції evaluate_model() в циклі

```
# repeat experiment
scores = list()
for r in range(repeats):
      score = evaluate_model(trainX, trainy, testX, testy)
      score = score * 100.0
      print('>#%d: %.3f' % (r+1, score))
      scores.append(score)
```

Було узагальнено вибірку балів, розрахувавши і повідомивши середнє і стандартне відхилення показників. Середнє значення дає середню точність моделі в наборі даних, тоді як стандартне відхилення дає середнє відхилення точності від середнього.

Функція summarize_results () підсумовує результати виконання.

Лістинг 3.9 – Функція summarize_results ()

```
# summarize scores
def summarize_results(scores):
      print(scores)
      m, s = mean(scores), std(scores)
      print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))
```

Було об'єднати повторну оцінку, збір результатів і узагальнення результатів в основну функцію для експерименту, названу run_experiment().

За замовчуванням модель оцінюється 10 разів, перш ніж було

повідомлено про продуктивність моделі.

Лістинг 3.10 – Оцінка моделі за допомогою run_experiment()

```
# run an experiment
def run_experiment(repeats=10):
	# load data
	trainX, trainy, testX, testy = load_dataset()
	# repeat experiment
	scores = list()
	for r in range(repeats):
		score = evaluate_model(trainX, trainy, testX, testy)
		score = score * 100.0
		print('>#%d: %.3f' % (r+1, score))
		scores.append(score)
	# summarize results
	summarize_results(scores)
```

При виконанні спочатку виводилося форма завантаженого набору даних, потім форма навчального і тестового наборів, а також вхідних і вихідних елементів. Це підтверджує кількість вибірок, часових кроків і змінних, а також кількість класів.

Потім створювалися і оцінювалися моделі, і для кожної з них друкувалося налагоджувальне повідомлення.

Друкувалась вибірка балів, за якою слідувало середнє значення і стандартне відхилення. Таким чином модель показала хороші результати, досягнувши точності класифікації близько 90,9%, навченої на необробленому наборі даних, зі стандартним відхиленням близько 1,3.

Це хороший результат, враховуючи, що в оригінальній статті був опублікований результат 89%, навчений набору даних з інтенсивною розробкою функцій для конкретної області, а не необробленого набору даних.

Лістинг 3.11 – Оцінка вибірки балів

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
(7352, 128, 9) (7352, 6) (2947, 128, 9) (2947, 6)
```

```
>#1: 91.347
>#2: 91.551
>#3: 90.804
>#4: 90.058
>#5: 89.752
>#6: 90.940
>#7: 91.347
>#8: 87.547
>#9: 92.637
>#10: 91.890

[91.34713267729894, 91.55072955548015, 90.80420766881574,
90.05768578215134, 89.75229046487954, 90.93993892093654, 91.34713267729894,
87.54665761791652, 92.63657957244655, 91.89005768578215]

Accuracy: 90.787% (+/-1.341)
```

До цього не було виконано жодної підготовки даних, а використовувалися дані як є. Далі після розгляду завантаження даних та відповідності моделі CNN, було виконано дослід, що повинен був ще більше підвищити кваліфікацію моделі за допомогою деякого налаштування гіперпараметрів.

Кожен з основних наборів даних (прискорення тіла, гіроскопічне тіло і загальне прискорення) був масштабований до діапазону -1, 1.

Одним з можливих перетворень, що могло призвести до поліпшення, була стандартизація спостережень до підгонки моделі.

Стандартизація відноситься до зміни розподілу кожної змінної таким чином, щоб вона мала середнє значення, рівне нулю, і стандартне відхилення, Рівне 1. Це мало би сенс тільки в тому випадку, якщо розподіл кожної змінної є гаусовим.

Для швидкої перевірити розподілу кожної змінної було побудовано гістограму кожної змінної в навчальному наборі даних.

Невелика складність у цьому полягала в тому, що дані були розділені на проміжки з 128 тимчасових кроків з 50% перекриттям. Тому, щоб отримати справедливе уявлення про розподіл даних, спочатку було видалено дубльовані спостереження (перекриття), а потім видалені проміжки даних.

Ця операція була виконана за допомогою NumPy, спочатку розрізавши масив і зберігши тільки другу половину кожного проміжку, а потім

вирівнявши вікна в довгий вектор для кожної змінної. Це швидко і “брудно” і означає, що є можливість втрати даних в першій половині першого проміжку.

Лістинг 3.12 – Видалення дубльованих спостережень та проміжків даних

```
# remove overlap
cut = int(trainX.shape[1] / 2)
longX = trainX[:, -cut:, :]
# flatten windows
longX = longX.reshape((longX.shape[0] * longX.shape[1], longX.shape[2]))
```

Виконання коду створило фігуру з дев'ятьма гістограмами, по одній для кожної змінної в наборі навчальних даних.

Порядок графіків відповідає порядку, в якому були завантажені дані, зокрема:

- загальне прискорення x;
- загальне прискорення y;
- повне прискорення z;
- прискорення тіла x;
- прискорення тіла y;
- прискорення тіла z;
- корпус гіроскопа x;
- корпусний гіроскоп y;
- корпус гіроскопа z.

Кожна змінна має розподіл, подібний до гаусова, за винятком першої змінної (загальне прискорення x).

Розподіл даних про загальне прискорення є більш рівним, ніж дані про тіло, які є більш загостреними.

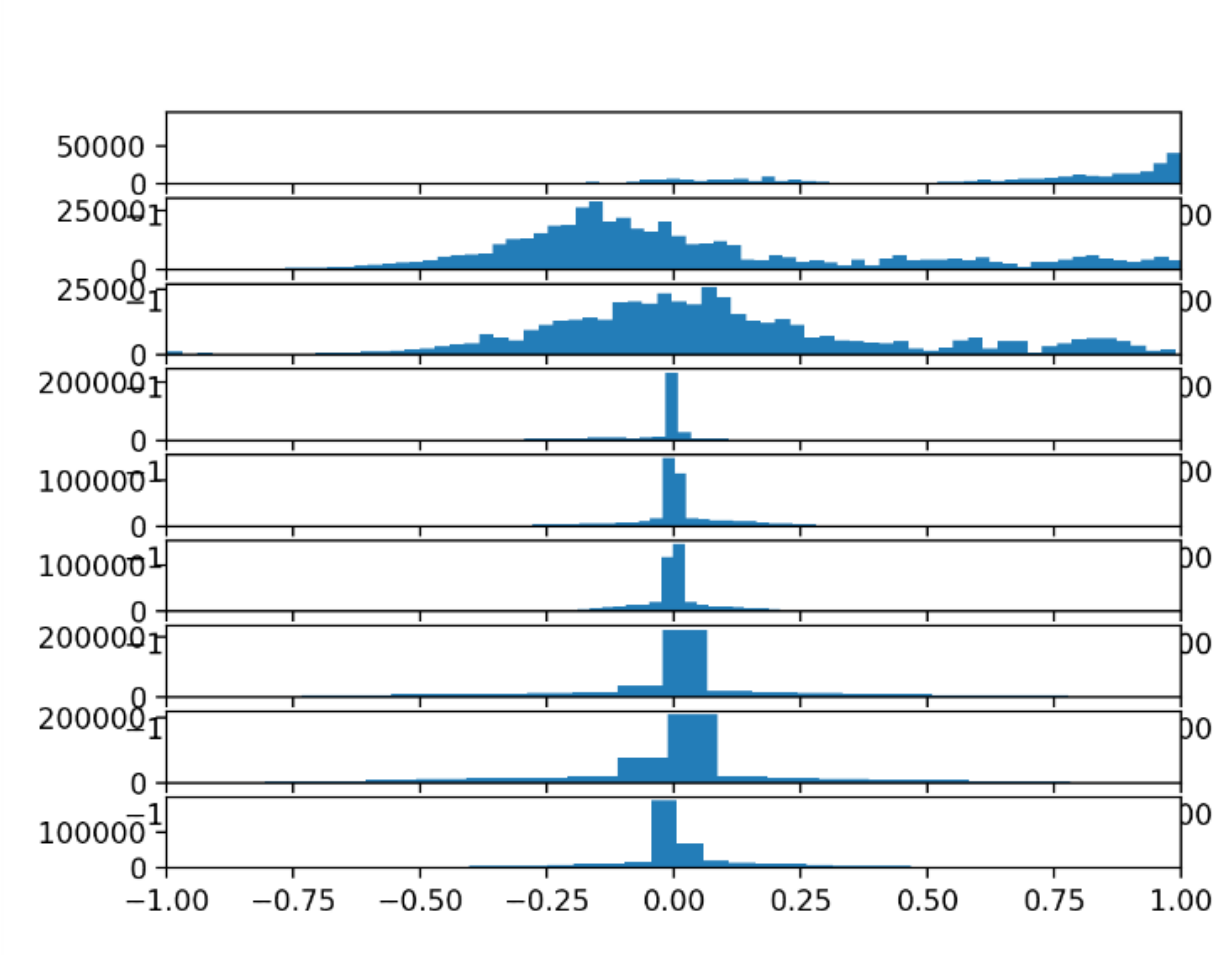

Рисунок 3.1 – Гістограми кожної змінної в наборі навчальних даних

Дані досить схожі на гаусови, щоб дослідити, чи перетворення стандартизації моделі допоможе отримати помітний сигнал з необроблених спостережень.

Функція scale_data() використовувалася для стандартизації даних до підгонки і оцінки моделі. Для виконання перетворення було використано клас StandardScaler scikit-learn. Спочатку він зіставлявся з даними навчання (наприклад, для визначення середнього і стандартного відхилення для кожної змінної), а потім застосовувався до наборів навчання і тестів.

Стандартизація необов'язкова, тому можна було застосувати процес і порівняти результати з тим же шляхом коду без стандартизації в контрольованому експерименті.

Лістинг 3.13 – Код для стандартизації

```
      # standardize data
def scale_data(trainX, testX, standardize):
      # remove overlap
      cut = int(trainX.shape[1] / 2)
      longX = trainX[:, -cut:, :]
      # flatten windows
      longX = longX.reshape((longX.shape[0] * longX.shape[1], longX.shape[2]))
      # flatten train and test
      flatTrainX = trainX.reshape((trainX.shape[0] * trainX.shape[1],
trainX.shape[2]))
      flatTestX = testX.reshape((testX.shape[0] * testX.shape[1],
testX.shape[2]))
      # standardize
      if standardize:
            s = StandardScaler()
            # fit on training data
            s.fit(longX)
            # apply to training and test data
            longX = s.transform(longX)
            flatTrainX = s.transform(flatTrainX)
            flatTestX = s.transform(flatTestX)
      # reshape
      flatTrainX = flatTrainX.reshape((trainX.shape))
      flatTestX = flatTestX.reshape((testX.shape))
      return flatTrainX, flatTestX
```

Було оновлено функцію evaluate_model(), щоб прийняти параметр, а потім використовувати цей параметр, щоб вирішити, чи слід виконувати стандартизацію.

Лістинг 3.14 – Використовувати нового параметру

```
      # fit and evaluate a model
def evaluate_model(trainX, trainy, testX, testy, param):
      verbose, epochs, batch_size = 0, 10, 32
      n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
trainy.shape[1]
      # scale data
      trainX, testX = scale_data(trainX, testX, param)
      model = Sequential()
      model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(n_timesteps,n_features)))
      model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
      model.add(Dropout(0.5))
      model.add(MaxPooling1D(pool_size=2))
      model.add(Flatten())
      model.add(Dense(100, activation='relu'))
      model.add(Dense(n_outputs, activation='softmax'))
      model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
      # fit network
```

```
	model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
	# evaluate model
	_, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose=0)
	return accuracy
```

Також було оновлено run_experiment (), щоб повторити експеримент 10 разів для кожного параметра; в цьому випадку було оцінено тільки два параметри [False, True] для відсутності стандартизації і стандартизації відповідно.

Лістинг 3.15 – Оновлена функція run_experiment()

```
	# run an experiment
	def run_experiment(params, repeats=10):
		# load data
		trainX, trainy, testX, testy = load_dataset()
		# test each parameter
		all_scores = list()
		for p in params:
			# repeat experiment
			scores = list()
			for r in range(repeats):
				score = evaluate_model(trainX, trainy, testX, testy,
p)
				score = score * 100.0
				print('>p=%d #%d: %.3f' % (p, r+1, score))
				scores.append(score)
			all_scores.append(scores)
		# summarize results
		summarize_results(all_scores, params)
```

Це призвело до двох зразків результатів, які можна порівняти.

Також було оновлено функцію summarize_results(), щоб підсумувати вибірку результатів для кожного параметра конфігурації і створити прямокутну діаграму для порівняння кожної вибірки результатів.

Лістинг 3.16 – Оновлена функція summarize_results()

```
	# summarize scores
	def summarize_results(scores, params):
		print(scores, params)
		# summarize mean and standard deviation
		for i in range(len(scores)):
			m, s = mean(scores[i]), std(scores[i])
```

```
        print('Param=%d: %.3f%% (+/-%.3f)' % (params[i], m, s))
    # boxplot of scores
    pyplot.boxplot(scores, labels=params)
    pyplot.savefig('exp_cnn_standardize.png')
```

Ці оновлення дозволили безпосередньо порівнювати результати підгонки моделі, як і раніше, і підгонки моделі до набору даних після його стандартизації.

Продуктивність друкувалась для кожної оцінюваної моделі. В кінці виконання продуктивність кожної з протестованих конфігурацій підсумовувалась із зазначенням середнього значення і стандартного відхилення.

Стандартизація набору даних перед моделюванням призвела до невеликого підвищення продуктивності з точності приблизно 90,4 % до точності приблизно 91,5 %.

Лістинг 3.17 – Оцінка вибірки зі стандартизацією і без неї

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
(7352, 128, 9) (7352, 6) (2947, 128, 9) (2947, 6)

>p=False #1: 91.483
>p=False #2: 91.245
>p=False #3: 90.838
>p=False #4: 89.243
>p=False #5: 90.193
>p=False #6: 90.465
>p=False #7: 90.397
>p=False #8: 90.567
>p=False #9: 88.938
>p=False #10: 91.144
>p=True #1: 92.908
>p=True #2: 90.940
>p=True #3: 92.297
>p=True #4: 91.822
>p=True #5: 92.094
>p=True #6: 91.313
>p=True #7: 91.653
>p=True #8: 89.141
>p=True #9: 91.110
>p=True #10: 91.890

[[91.48286392941975, 91.24533423820834, 90.83814048184594,
89.24329826942655, 90.19341703427214, 90.46487953851374,
90.39701391245333, 90.56667797760434, 88.93790295215473,
91.14353579911774], [92.90804207668816, 90.93993892093654,
92.29725144214456, 91.82219205972176, 92.09365456396336,
```

```
91.31319986426874, 91.65252799457076, 89.14149983033593,
91.10960298608755, 91.89005768578215]] [False, True]

Param=False: 90.451% (+/-0.785)
Param=True: 91.517% (+/-0.965)
```

Також був створений графік результатів у вигляді прямокутника і вусів.

Це дозволило порівнювати дві вибірки результатів непараметричним способом, показуючи медіану і середні 50% кожної вибірки.

Розподіл результатів зі стандартизацією сильно відрізняється від розподілу результатів без стандартизації.

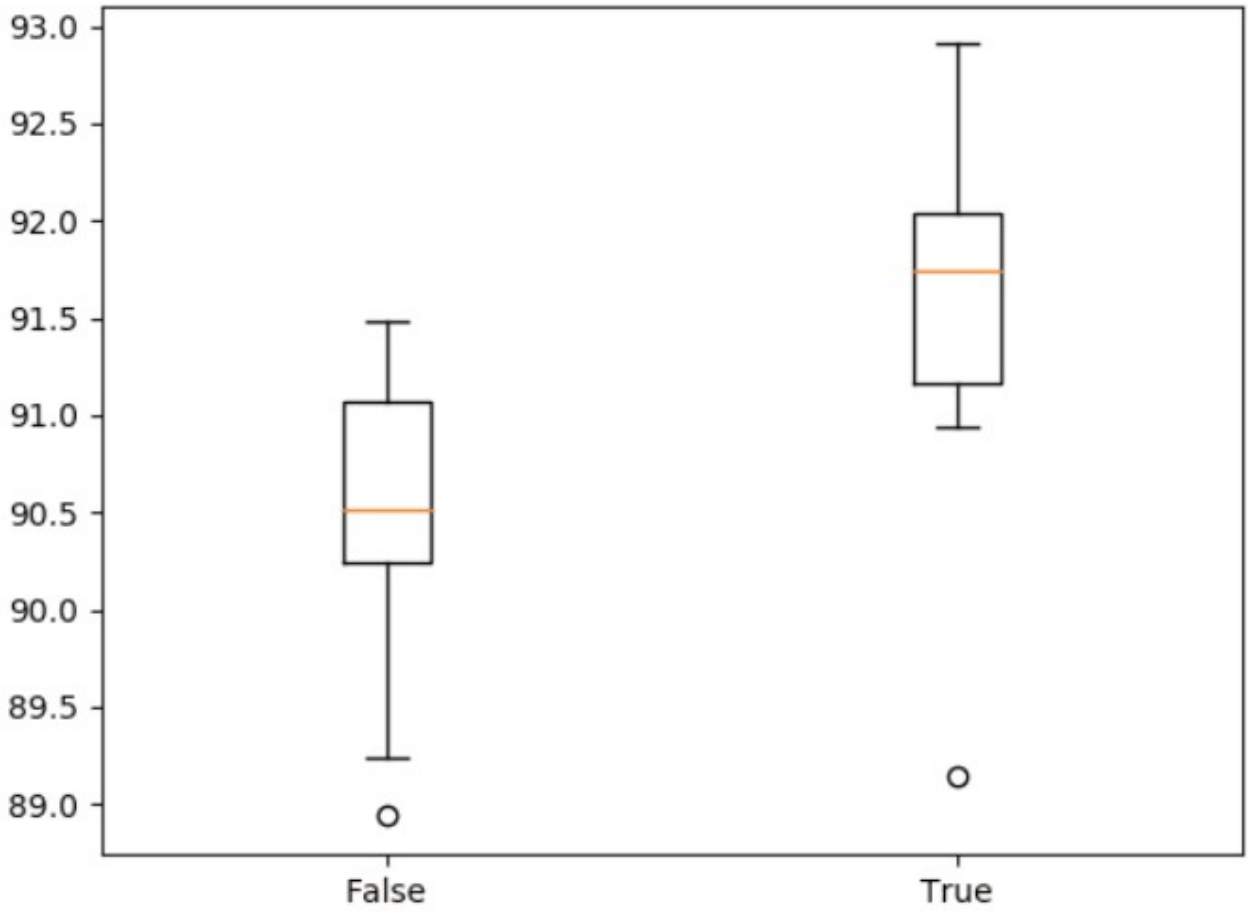

Рисунок 3.2 – CNN зі стандартизацією і без неї

Після отримання експериментальної база, було досліджено інші

гіперпараметри моделі.

Важливим гіперпараметром для CNN є кількість карт фільтрів. Були виконані експерименти з діапазоном різних значень, від меншого до набагато більшого, ніж 64, використані в першій розробленої моделі.

Зокрема були протестовані карти об'єктів 8, 16, 32, 64, 128 та 256.

Код був використаний той же, що і раніше з невеликими змінами у функції evaluate_model(), щоб використовувалися надані параметри як кількість фільтрів у шарах Conv1D. Також була оновлена функція summarize_results(), щоб зберегти boxplot як exp_cnn_filters.png.

Виконання тестування повторювало експеримент для кожного із зазначеної кількості фільтрів.

Спостерігалась тенденція до збільшення середньої продуктивності зі збільшенням кількості карт фільтрів. Різниця залишилась досить постійною, і 128 карт об'єктів можуть бути гарною конфігурацією для мережі.

Лістинг 3.18 – Оцінка вибірки з різною кількістю карт фільтрів

```
...
Param=8: 89.148% (+/-0.790)
Param=16: 90.383% (+/-0.613)
Param=32: 90.356% (+/-1.039)
Param=64: 90.098% (+/-0.615)
Param=128: 91.032% (+/-0.702)
Param=256: 90.706% (+/-0.997)
```

Також був створений графік результатів у вигляді прямокутника і вусів, що дозволило порівнювати розподіл результатів з кожною кількістю фільтрів.

На графіку спостерігається тенденцію до зростання з точки зору середньої точності класифікації (помаранчева лінія на полі) зі збільшенням кількості карт об'єктів. Видно падіння на 64 картах об'єктів (за замовчуванням або базової лінії в експериментах), що дивно, і плато точності на 32, 128 і 256 картах фільтрів. Використання 32 було більш стабільною конфігурацією.

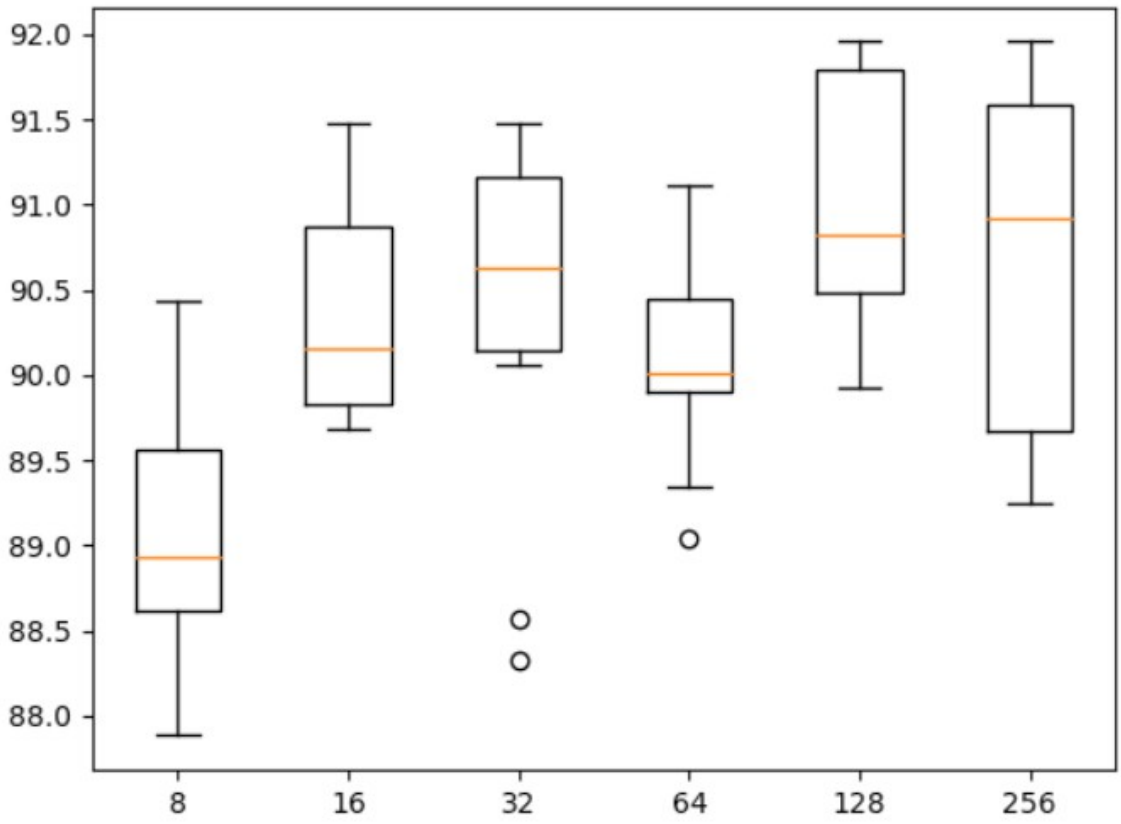

Рисунок 3.3 – CNN з різною кількістю карт фільтрів

Розмір ядра є ще одним важливим гіперпараметром CNN для налаштування.

Розмір ядра визначає кількість тимчасових кроків, що враховуються при кожному “зчитуванні” вхідної послідовності, яка потім проектується на карту об'єктів (за допомогою згорткового процесу).

Великий розмір ядра означає менш суворе зчитування даних, але може привести до більш узагальненого знімку вхідних даних.

Було використано одну і ту ж експериментальну установку і протестовано набір ядер різних розмірів на додаток до трьох тимчасових кроків за замовчуванням. До список значень потрапили 2, 3, 5, 7 та 11 ядер.

Виконання тестування перевіряло кожен розмір ядра по черзі.

Результати підсумовувалися в кінці пробігу. Спостерігалося загальне збільшення продуктивності моделі зі збільшенням розміру ядра.

Результати показали, що розмір ядра 5 може бути хорошим при середньому навику близько 91,8 %, але розмір 7 або 11 також може показати хороші результати при меншому стандартному відхиленні.

Лістинг 3.19 – Оцінка вибірки з різною кількістю розмірів ядер

```
...
Param=2: 90.176% (+/-0.724)
Param=3: 90.275% (+/-1.277)
Param=5: 91.853% (+/-1.249)
Param=7: 91.347% (+/-0.852)
Param=11: 91.456% (+/-0.743)
```

Також був створений графік результатів.

Результати показали, що більший розмір ядра призводить до більшої точності, і що розмір ядра 7 забезпечує хороший баланс між гарною продуктивністю та низькою дисперсією.

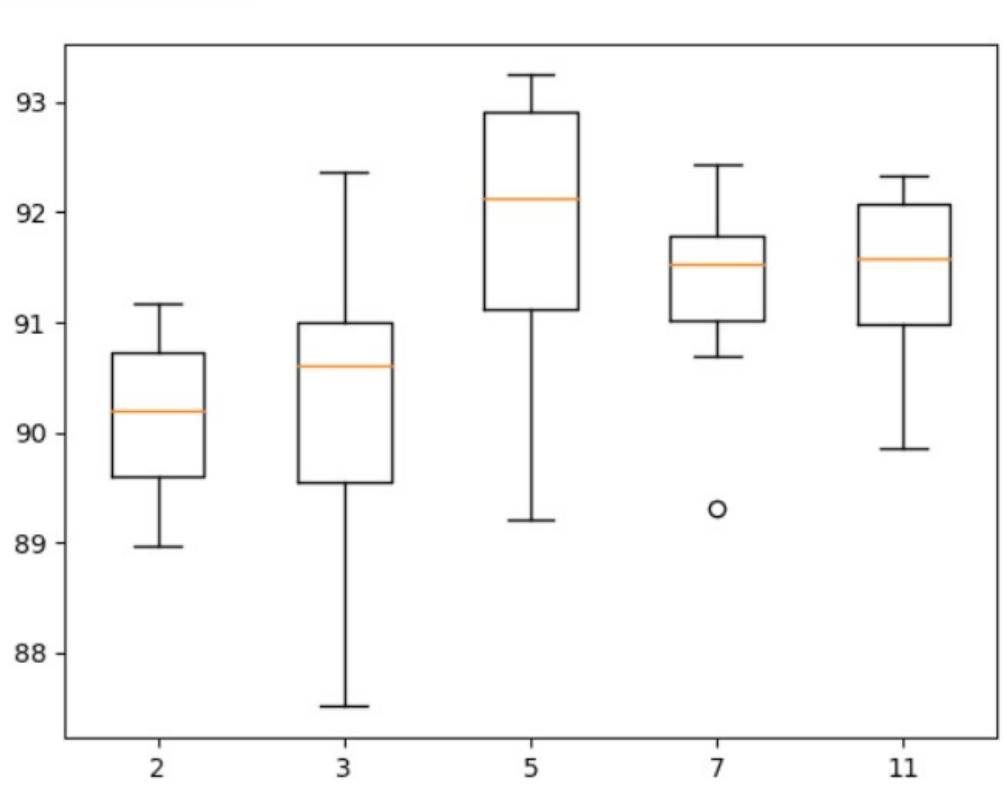

Рисунок 3.4 – CNN з різною кількістю розмірів ядер

На цьому налаштування та тестування моделі було завершено, хоча це лише найбільш важливі налаштування при необхідності можна спробувати підвищити продуктивність за рахунок комбінації інших налаштувань. Наприклад, збільшити кількість повторів від 10 до 30 або більше, щоб, можливо, досягти більш стабільних результатів.

3.3 Перенесення нейронної мережі на мікроконтролер

Для портування нейронної мережі, що попередньо була навчена на класичному комп'ютері було виконано квантуванням у форматі 8-бітних чисел з фіксованою точкою, оскільки генератор коду X-CUBE-AI підтримує тільки прості масиви вхідних і вихідних даних:

– 4-розмірна форма: група, висота, ширина, канал (формат «channel-last»);
– 32-бітові дані з плаваючою точкою і 8-ми бітові дані з фіксованою точкою.

Тому, для моделей Keras потрібен був файл переформатованої моделі (h5) і спеціалізований конфігураційний файл (json). Згенеровані моделі на С були повністю оптимізовані для ядер STM32 Arm Cortex-M4/M7 C FPU і DSP.

Генератор коду перетворив значення вагів і зміщення, а також відповідні активації з формату 32-бітних чисел з плаваючою точкою в формат 8-бітних чисел з фіксованою точкою. Метою цієї операції було зменшення розміру моделі для збільшення швидкості обчислень, забезпечуючи при цьому лише незначне зниження точності.

Наступним кроком було підготовка проекту для налагоджувальної плати. Для цього було необхідно встановити програмне забезпечення. Спочатку потрібно було встановити STM32CubeMX версії 5.1.0 або новіше. STM32CubeMX доступний як для операційної системи Windows, так і для Linux. Також було потрібно завантажити пакет розширення X-CUBE-AI і

середовище STM32CubeIDE, яке було необхідне для компіляції згенерованого коду.

Після завершення установки STM32CubeMX пакет X-CUBE-AI встановлюється наступним чином.

Запускається STM32CubeMX. В меню вибирається пункт «Help», далі «Manage embedded software packages», або безпосереднім натисканням кнопку «INSTALL/REMOVE» (рисунок 3.5).

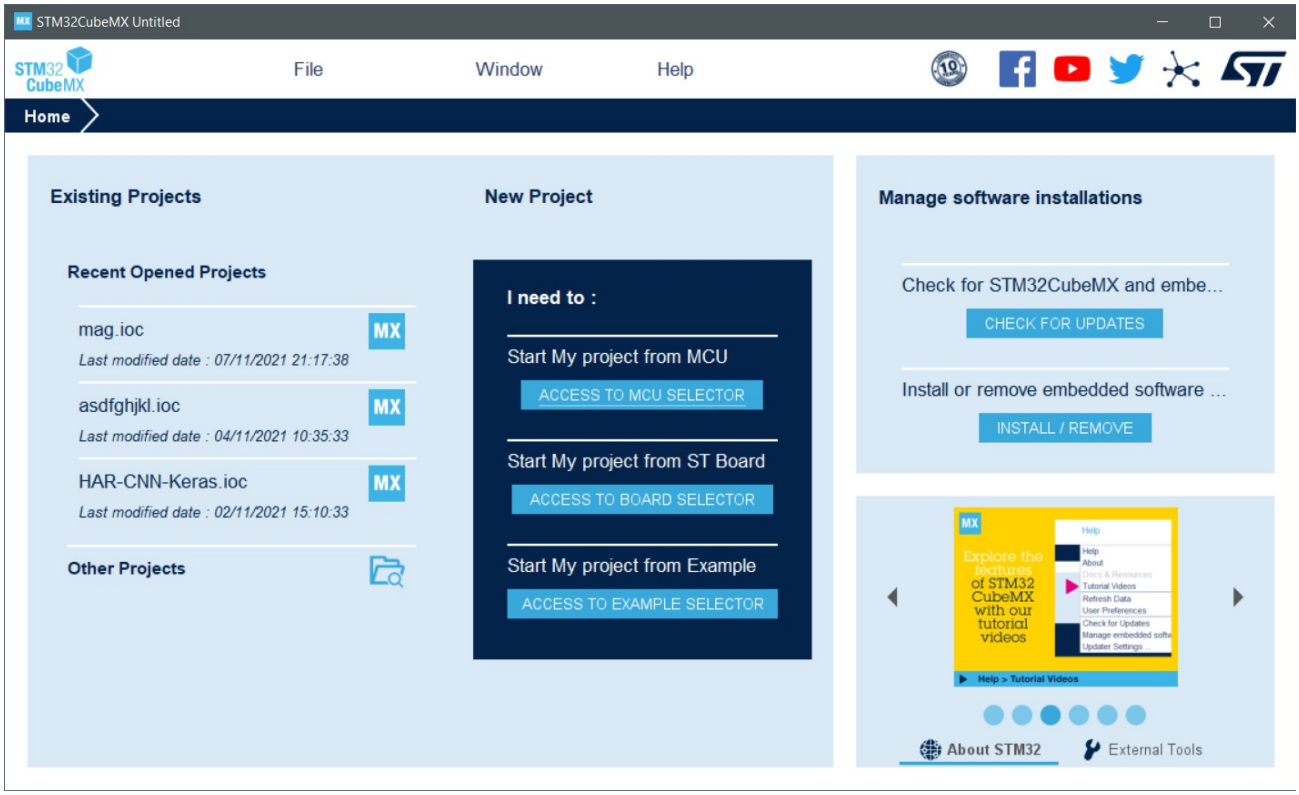

Рисунок 3.5 – Головне вікно в STM32CubeMX

У вікні «Embedded Software Packages Manager» необхідно було натиснути кнопку «Refresh» для отримання оновленого списку додаткових модулів. Для пошуку X-CUBE-AI обиралася вкладка «STMicroelectronics» (рисунок 3.6).

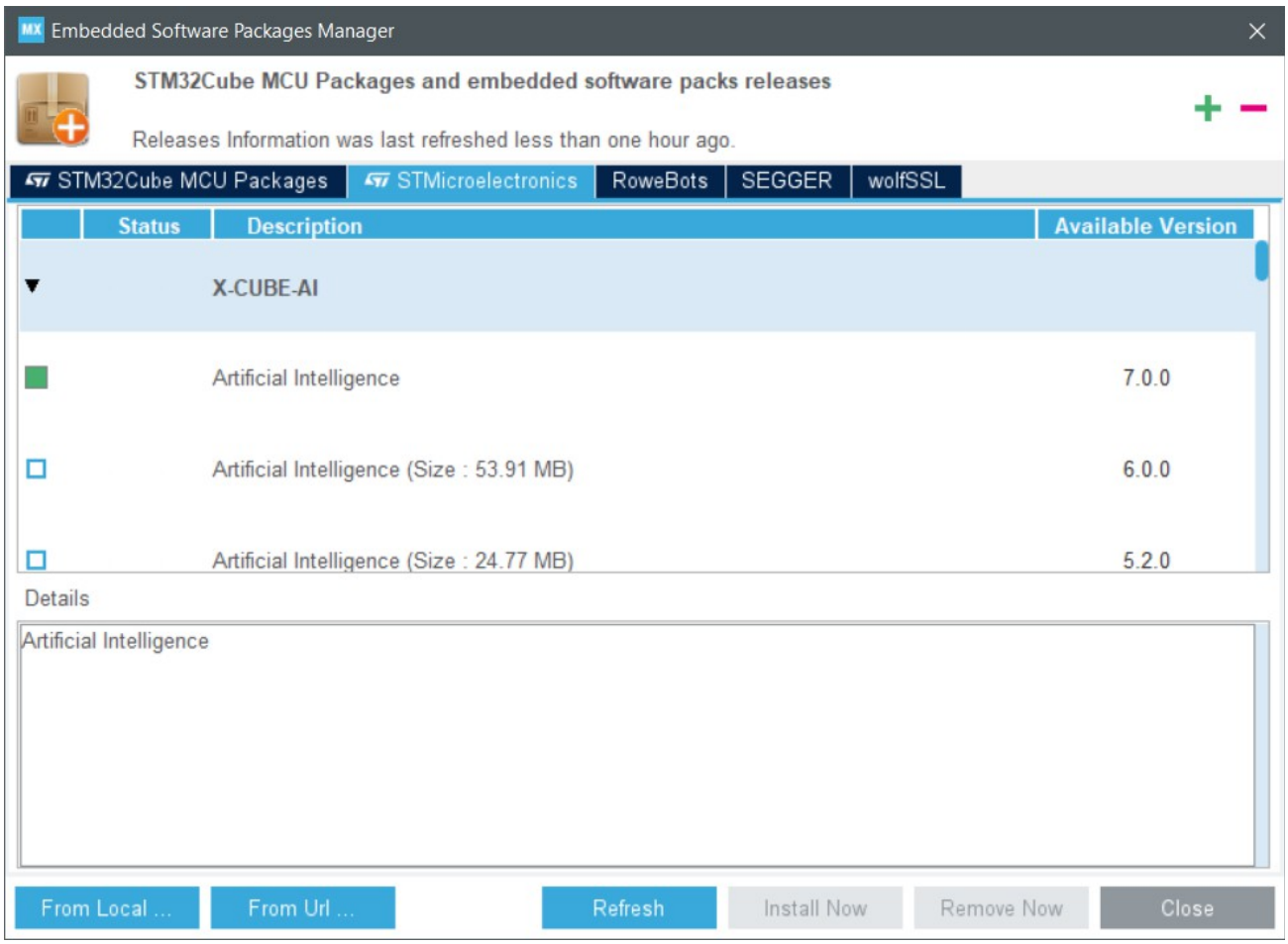

Рисунок 3.6 – Встановлення X-CUBE-AI в STM32CubeMX

Необхідна версія пакета вибиралась і встановлювалась шляхом натискання кнопки «Install Now». Після завершення установки квадрат поруч з необхідною версією пакета став зеленим, після чого можна було натиснути кнопку «Close».

Після установки і запуску програми STM32CubeMX необхідно було обарти пункт «ACCESS TO BOARD SELECTOR».

У вікні, в поле «Commercial Part Number» був введнений номер необхідного продукту «NUCLEO-G474RE», після чого в списку «Board List» виділялося необхідна налагоджувальну плату (рисунок 3.7).

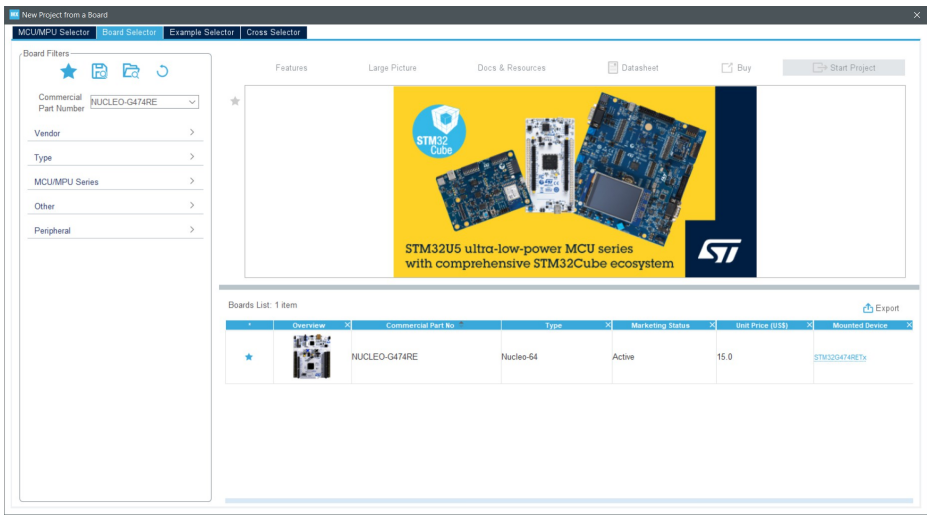

Рисунок 3.7 – Ініціалізація проекту

Подвійним кліком був запущений проект. При запуску якого з'являлось повідомлення «Initialize all peripherals with their Default Mode?». При відповіді «Yes» було виконана ініціалізація всі периферійних пристроїв в режимі за замовчуванням.

Виходи контролера також були автоматично налаштовані відповідно до трасування налагоджувальної плати NUCLEO-G474RE.

Тактову частоту проекту можна було перевірити і змінити у вкладці «Clock configuration» (рисунок 3.8).

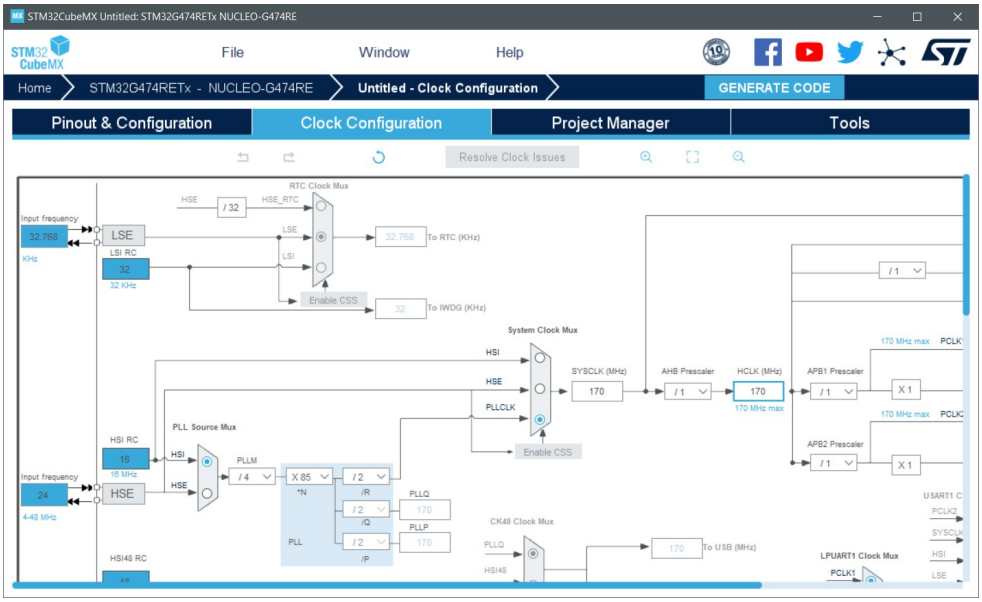

Рисунок 3.8 – Налаштування тактової частоти

Для налагоджувальної плати NUCLEO-G474RE максимальна частота ядра - 170 МГц. Саме це значення (параметр «HCLK (MHz)») і було вибрано.

Для обміну даними з комп'ютером використовувався порт LPUART1, основні налаштування якого можна подивитися, вибравши «Categories», «Connectivity», «LPUART1» (рисунок 3.9).

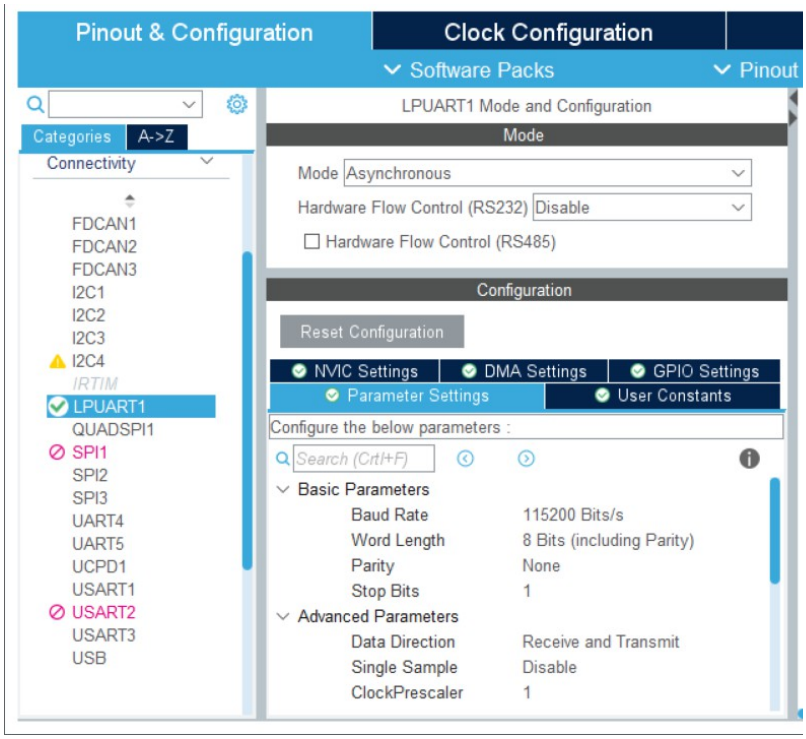

Рисунок 3.9 – Налаштування порту UART

Далі необхідно було підключити до проекту модуль X-CUBE-AI. Для цього обиралось в меню «Software Packs» далі «Select components». Потім у вікні був вибираний «STMicroelectronics.X-CUBE-AI» далі «Artificial Intelligence X-CUBE-AI» і «Core» та у меню «Application» пункт «Application template» (рисунок 3.10).

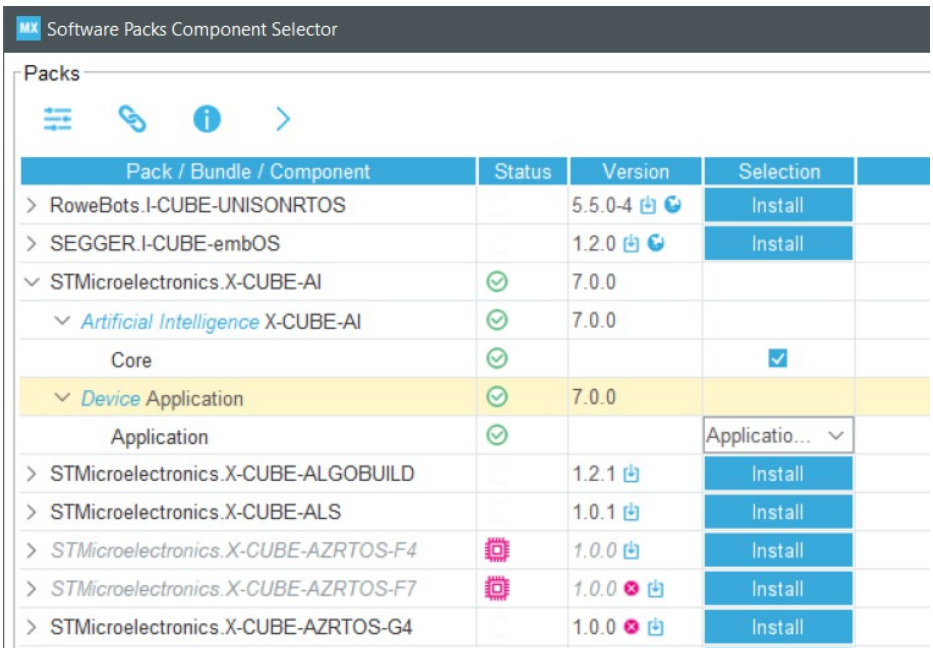

Рисунок 3.10 – Вибір модуля X-CUBE-AI

Існує кілька типів додатків, які можна обрати:

– System performance – для перевірки продуктивності реалізацій різних нейромереж на цільовій платформі;
– Validation – для перевірки продуктивності нейромереж і порівняння результатів обчислень;
– Application template – шаблон, що дозволяє створювати призначені для користувача програми.

Наступним кроком було вибирано із меню «Categories» пункт «STMicroelectronics.X-CUBE-AI». У вікні «Configuration», «Model inputs» обиралася назва для моделі. Тип мережі «Keras», було обрано «Saved model», тобто використовалась заздалегідь збережена мережа. Шлях вказувався до файлу нейромережі .h5, який був раніше завантажений. Обиралася ступінь стиснення моделі – в даному випадку 8, і активована кнопка «Analyze» (рисунок 3.11). Після закінчення аналізу було перевірено обчислювальні ресурси, необхідні для реалізації даної мережі із заданим ступенем стиснення. Необхідно було 366.65 кбайт Flash-пам'яті і 25.08 кбайт пам'яті

RAM. У дужках поруч вказані розміри пам'яті обраного мікроконтролера (рисунок 3.12).

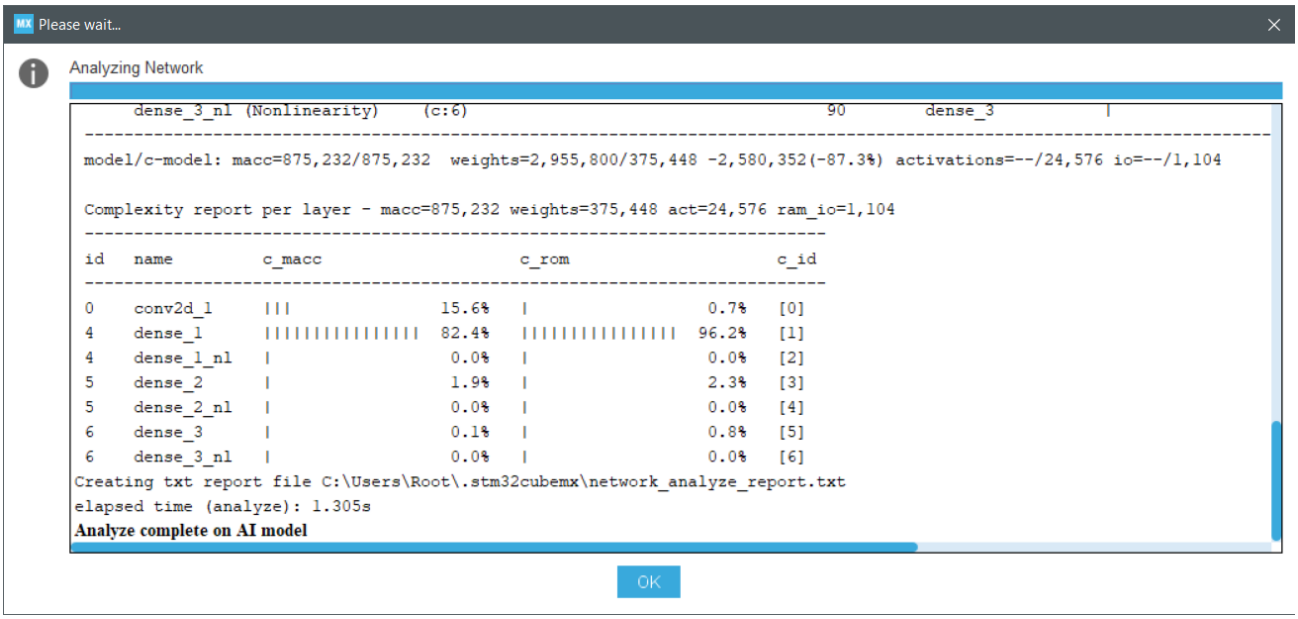

Рисунок 3.11 – Результат аналізу моделі

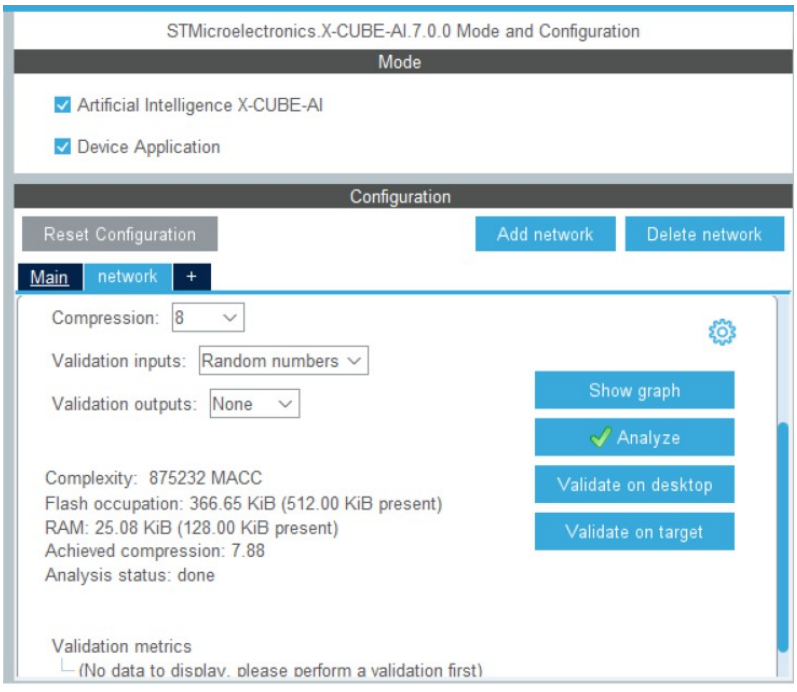

Рисунок 3.12 – Розрахунок обчислювальних ресурсів

Коефіцієнт стиснення 8 був обраний через обмежений обсягу пам'яті, так як модель з коефіцієнтом 4 вже не вмістилася б у Flash-пам'ять обраного мікроконтролера.

Було виконано перевірку, наскільки точно код, згенерований для мікроконтролера, відповідає оригінальній моделі нейромережі. Для цього існує два види тестів: на комп'ютері і на пристрої STM32.

Модель нейромережі, згенерована на мові C, запускається на виконання на процесорі x86 або STM32, і результат виконання порівнюється з оригінальною моделлю. При помилці L2 менше 0,01 результат побудови c-моделі вважається успішним. Великий вплив на помилку надає ступінь стиснення.

Спочатку була виконана перевірка моделі на комп'ютері в середовищі x86. Після натискання кнопки «Validate on desktop» відкривається вікно, що відображає процес перевірки. В даному випадку в якості вхідних даних були використані випадкові числа (рисунок 3.13).

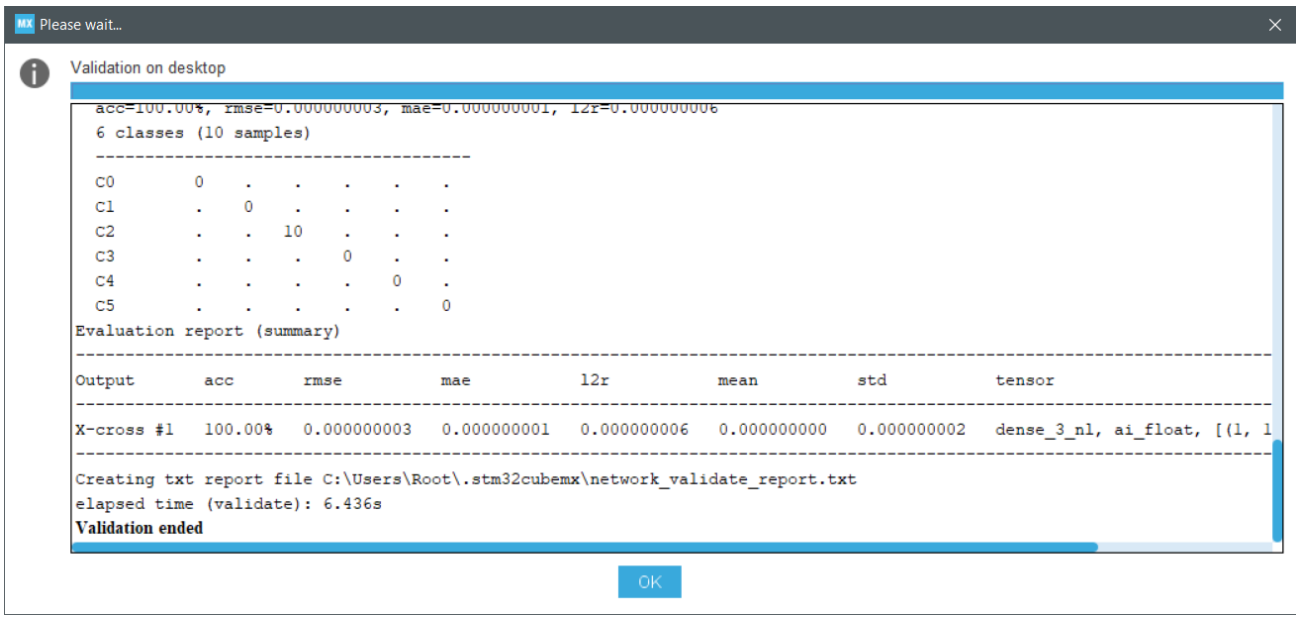

Рисунок 3.13 – Результат перевірки моделі на комп'ютері

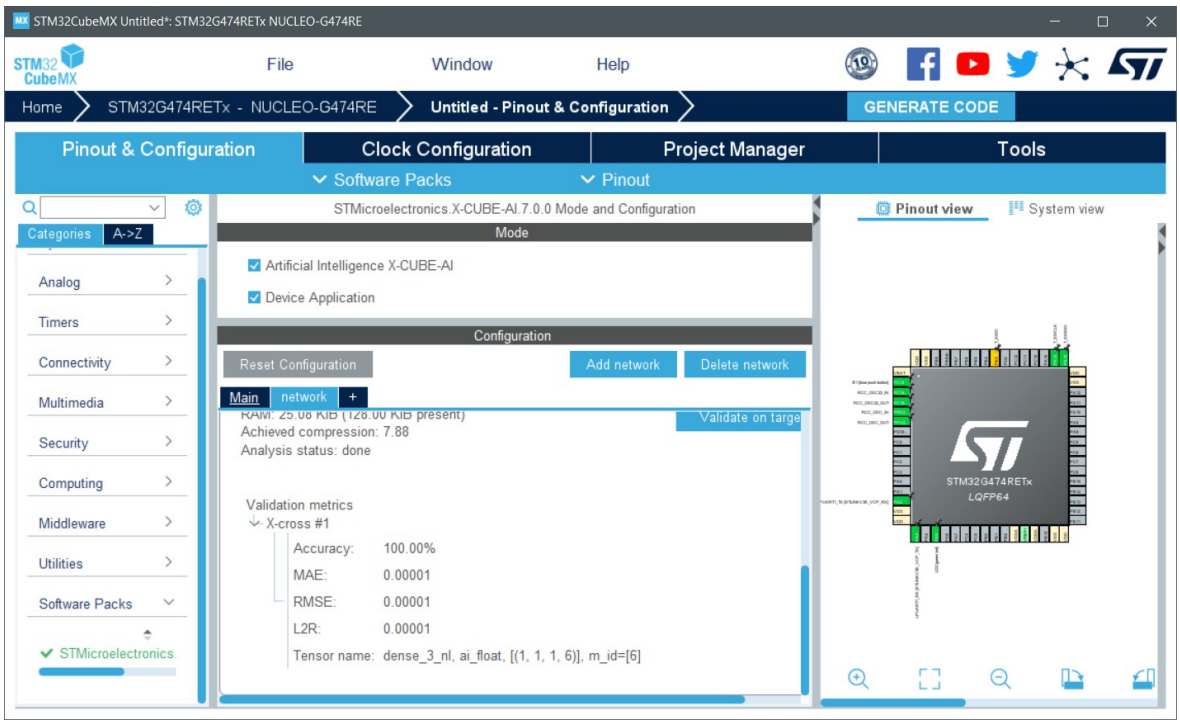

Рисунок 3.14 – Спрощені результати перевірки моделі на комп'ютері

Помилка L2 = 6 e-09, що значно менше допустимої похибки 0,01. Також у цьому вікні можно було дізнатися відсоток використання обчислювальних ресурсів різними шарами нейромережі.

Для перевірки моделі на STM32 активувалася кнопку «Validate on target». При цьому відкривалося вікно конфігурації перевірки (рисунок 3.15).

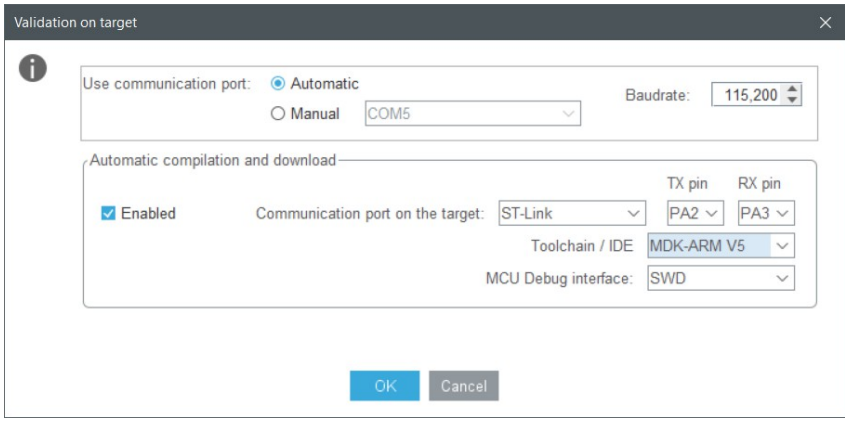

Рисунок 3.15 – Налаштування перевірки моделі в STM32

Ім'я порту залежить від операційної системи. В даному прикладі використовувалася система Windows 10, де порт для зв'язку з

налагоджувальної платою отримав автоматичну назву COM5. Швидкість 115200 відповідає налаштуванням проекту. Для перевірки моделі в мікроконтролері необхідне було середовище (Toolchain / IDE) для компіляції вихідних кодів, згенерованих за допомогою STM32CubeMX. В даному випадку для компіляції використовувалось середовище MDK-ARM V5, яке було попередньо встановлено на комп'ютер.

Після виконання необхідних налаштувань натискалося конпка «OK» і очікувалось виконання тесту, який включав в себе компіляцію проекту, прошивку плати, виконання тесту та аналіз отриманих результатів (рисунок 3.16).

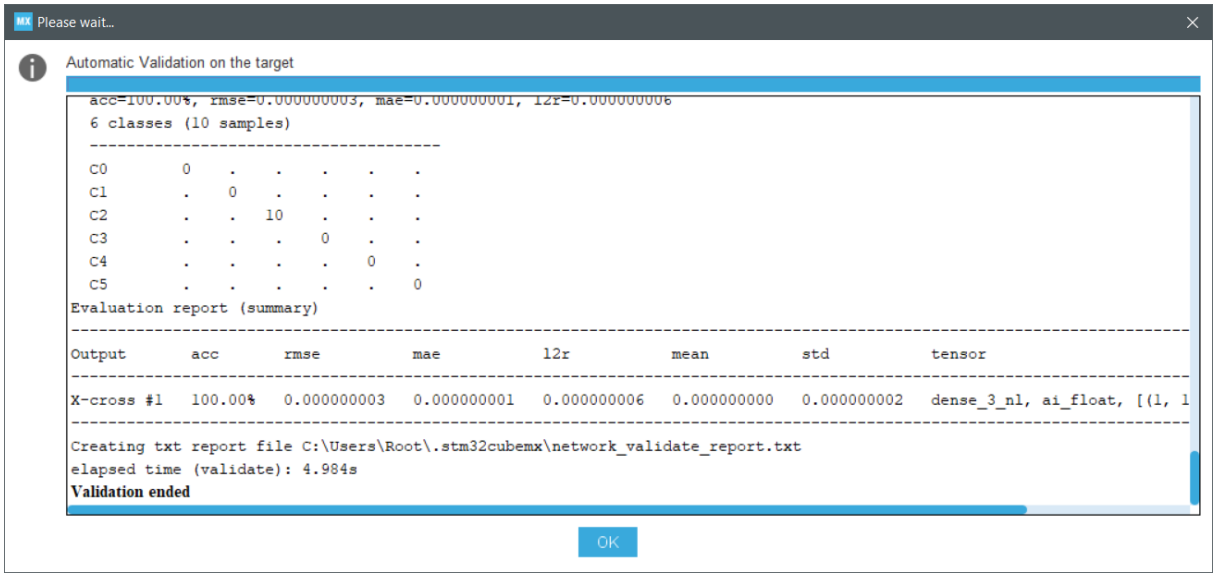

Рисунок 3.16 – Результат виконання тесту в STM32

Отримані результати повністю відповідали попереднім тестам на ПК.

Настпним кроком перевірки працездатності було безпосередне підключення до плати с з мікроконтроллером за допомногою попередньо завантаженої програми PuTTY (рисунок 3.17).

Після натискання на мікроконтролері кнопки «RESET» до консолі надходили результати роботи з данними нейрнної мережі, включаючи головні дані, що поступали на вхід та результат їх опрацювання для виходу (рисунок 3.18). При побудові більш складної систме, що включала би

приймач та візуалізатор даних схожий результат міг би бути перетвореним у зрозумілий для людей результат, але при використанні отрманої мережі без сторонніх програмних додатків, отриманий результат є цілковитим успіхом.

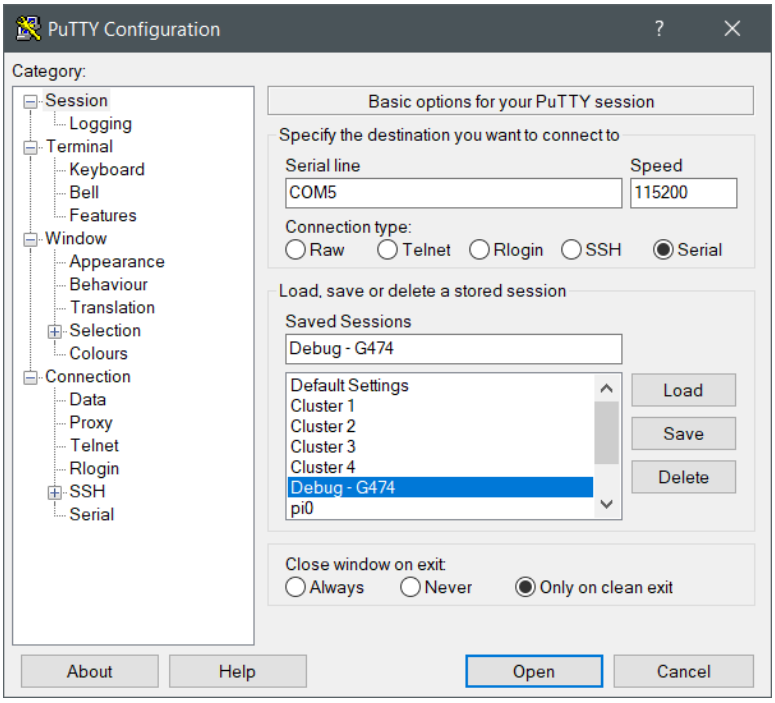

Рисунок 3.17 – Налаштування підключення у PuTTY

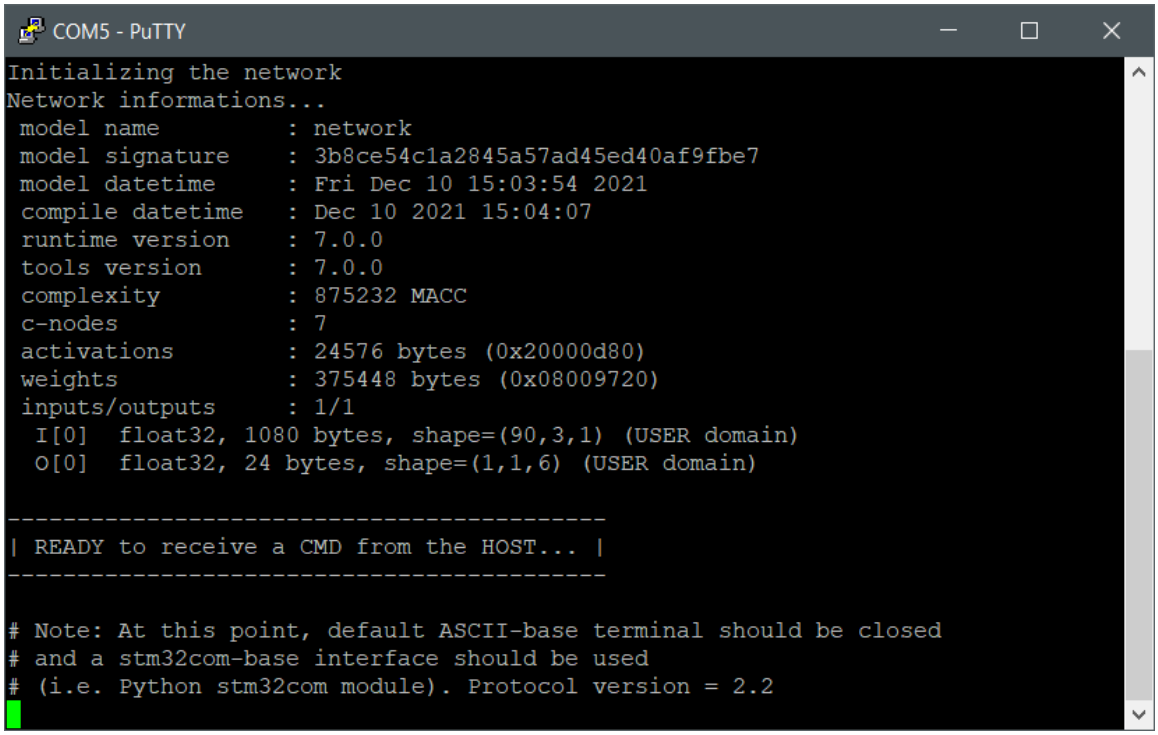

Рисунок 3.18 – Результати роботи мікроконтролера

ВИСНОВКИ

В рамках даної кваліфікаційної роботи було розглянуто можливість портування на мікроконтролер нейронної мережі, навченої на класичній ЕОМ. Для цього було створено прототип програмної системи для перенесення нейронної мережі та її запуску на мікроконтролері. В якості мікроконтролера використовувалася налагоджувальної плати NUCLEO-G474RE.

Для проведення експерименту була навчена нейронна мережа, яка виконувала розпізнавання людської діяльності. Для навчання був використаний набір даних, що має вільне розповсюдження «Human Activity Recognition Using Smartphones Data Set, UCI Machine Learning Repository», який було створено 2012 році вченими університету Генуї на смартфонах з використанням багатокласової апаратно-орієнтованої векторної машини підтримки. Нейронна мережа створена з використанням платформи TensorFlow і бібліотеки Keras. У готовій нейронній мережі генератором коду було перетворено значення вагів і зміщення, а також відповідні активації з формату 32-бітних чисел з плаваючою точкою в формат 8-бітних чисел з фіксованою точкою, після чого отриману модель було пристосовано для портування на мікроконтролер та виконано прошивку. На готовому мікроконтролері було виконано перевірку працездатності. Розглянутий приклад виконує поставлені задачі, має достатню швидкість обчислення та невелику вартість реалізації.

Таким чином мікроконтролери є життєздатною платформою для запуску програм глибокого навчання, за умови можливістю подолання моделью обмеження, що накладаються обмеженою пам'яттю і сховищем. У майбутньому цей проект може бути застосований для відстеження активності при заннятях спортом з врахуванням додаткової розробки візуального інтерфесу, що дозволив би отримувати результати у зрозумілій формі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. McCulloch W., Pitts W. A Logical Calculus of the Ideas Immanent in Nervous Activity (1943). Ideas That Created the Future. 2021. P. 79—88. [Електронний ресурс] – Режим доступу: URL: https://doi.org/10.7551/mitpress/12274.003.0011.
2. Hebb D. O. The Organization of Behavior: A Neuropsychological Theory [Текст] : Lawrence Erlbaum, 2002. 336 p.
3. Zell A. Simulation Neuronaler Netze [Текст] : Oldenbourg, 2003. 624 p.
4. The Improved Training Algorithm of Back Propagation Neural Network with Self-adaptive Learning Rate / Y. Li et al. 2009 International Conference on Computational Intelligence and Natural Computing (CINC), Wuhan, China, 6—7 June 2009. 2009. [Електронний ресурс] – Режим доступу: URL: https://doi.org/10.1109/cinc.2009.111.
5. Aleksander, Igor & De Gregorio, Massimo & França, Felipe & Lima, Priscila & Morton, Helen. (2009). A brief introduction to Weightless Neural Systems. URL: https://www.researchgate.net/publication/221166159_A_brief_introduction_to_Weightless_Neural_Systems
6. Ojha V. K., Abraham A., Snášel V. Metaheuristic design of feedforward neural networks: A review of two decades of research. Engineering Applications of Artificial Intelligence. 2017. Vol. 60. P. 97—116. [Електронний ресурс] – Режим доступу: URL: https://doi.org/10.1016/j.engappai.2017.01.013.
7. Kleene S. C. Representation of Events in Nerve Nets and Finite Automata. Automata Studies. (AM-34) / ed. by C. E. Shannon, J. McCarthy. Princeton, 1956. P. 3—42. [Електронний ресурс] –

Режим доступу: URL: https://doi.org/10.1515/9781400882618-002.

8. Hoskins J. C., Himmelblau D. M. Process control via artificial neural networks and reinforcement learning. Computers & Chemical Engineering. 1992. Vol. 16, no. 4. P. 241—251. [Електронний ресурс] – Режим доступу: URL: https://doi.org/10.1016/0098-1354(92)80045-b.
9. Bozinovski S., Bozinovska L. SELF-LEARNING AGENTS: A CONNECTIONIST THEORY OF EMOTION BASED ON CROSSBAR VALUE JUDGMENT. Cybernetics and Systems. 2001. Vol. 32, no. 6. P. 637—669. [Електронний ресурс] – Режим доступу: URL: https://doi.org/10.1080/01969720118145.
10. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997. Vol. 9, no. 8. P. 1735—1780. [Електронний ресурс] – Режим доступу: URL: https://doi.org/10.1162/neco.1997.9.8.1735.
11. Zissis D., Xidias E. K., Lekkas D. A cloud based architecture capable of perceiving and predicting multiple vessel behaviour. Applied Soft Computing. 2015. Vol. 35. P. 652—661. [Електронний ресурс] – Режим доступу: URL: https://doi.org/10.1016/j.asoc.2015.07.002.
12. Measuring systematic changes in invasive cancer cell shape using Zernike moments / E. Alizadeh et al. Integrative Biology. 2016. Vol. 8, no. 11. P. 1183—1193. [Електронний ресурс] – Режим доступу: URL: https://doi.org/10.1039/c6ib00100a.
13. Ghosh, Reilly. Credit card fraud detection with a neural-network. Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences, Wailea, HI, USA, 4—7 January 1994. 1994. [Електронний ресурс] – Режим доступу: URL: https://doi.org/10.1109/hicss.1994.323314.