Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ ПРЕДМЕТІВ ОДЯГУ НА ОСНОВІ СПАЙКІНГОВОЇ НЕЙРОМЕРЕЖІ

Виконала: студентка 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Чабан Я. К.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф.КН
Паночишин Ю.М.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н., проф. каф. КСУ
Биков М. М.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
«06» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Чабан Яні Костянтинівні

1. Тема роботи: Програмний модуль розпізнавання предметів одягу на основі спайкінгової нейромережі.
керівник роботи: Паночишин Юрій Миколайович, к.т.н., доц.
затверджені наказом вищого навчального закладу "20" березня 2025 року №97

2. Термін подання студентом роботи 30 травня 2025

3. Вихідні дані до роботи :
Формат вхідних файлів зображень – jpeg; кількість класів розпізнавання – 10; обсяг набору даних для навчання та тестування – не менше 1000 зображень; використання об'єктно-орієнтованої мови програмування.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
проаналізувати існуючі методи розпізнавання предметів одягу та вибрати найбільш ефективний;
вибрати нейронну мережу, спроектувати її структуру та метод навчання
розробити алгоритм роботи модуля розпізнавання предметів одягу;
спроектувати та реалізувати модуль розпізнавання предметів одягу за допомогою спайкінгової нейронної мережі;
протестувати роботу програми розпізнавання предметів одягу на основі спайкінгової нейронної мережі.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
схема алгоритму роботи програми
структура нейронної мережі
приклади зображень набору даних Fashion-MNIST
результати роботи програми

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Паночишин Ю.М., к.т.н., доц. каф. КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	05.05.25	20.05.25	
2	Обгрунтування методу розв'язання задачі, проектування програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі	11.05.25	13.05.25	
3	Розробка алгоритму роботи програмного модуля	14.05.25	19.05.25	
4	Програмна реалізація модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі	20.05.25	22.05.25	
5	Аналіз результатів тестування модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі	24.05.25	28.05.25	
6	Оформлення матеріалів до захисту БДР	02.06.25	04.06.25	

Студентка _____ **Чабан Я. К.**
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ **Паночишин Ю.М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Чабан Я. К. Програмний модуль розпізнавання предметів одягу на основі спайкінгової нейромережі. Бакалаврська кваліфікаційна робота скаладається з 71 сторінки формату А4, на яких є 16 рисунків, 3 таблиці, список використаних джерел містить 21 найменування.

Метою роботи є підвищення достовірності розпізнавання предметів одягу.

Розроблений програмний модуль призначений для розпізнавання предметів одягу як класичної задачі машинного навчання на наборі даних Fashion-MNIST. Було розроблено структуру спайкінгової нейронної мережі для розпізнавання предметів одягу, яка еквівалентна традиційній нейромережі, тобто містить два повнозв'язних шари спайкінгових нейронів у кількості відповідно 128 і 10, має 784 входи та 10 виходів. Модуль розроблено на мові програмування Python у середовищі програмування Colab та з використанням спеціалізованих бібліотек NumPy, Keras, TensorFlow та Matplotlib. Для навчання та тестування модуля взято набір зображень Fashion MNIST, який містить 70 000 зображень у градаціях сірого у 10 категоріях з роздільною здатністю 28 на 28 пікселів. Розроблений програмний модуль розпізнавання предметів одягу на основі спайкінгової нейронної мережі має достовірність розпізнавання на тестовому наборі 92,68%, а аналог – 90,60%. Тобто достовірність розпізнавання збільшена на 2,08%.

Ключові слова: розпізнавання, предмети одягу, Fashion-MNIST, спайкінгова нейронна мережа, достовірність розпізнавання.

ABSTRACT

Chaban Ya. K. Clothing recognition software module based on a spiking neural network. The bachelor thesis consists of 71 pages of A4 format, on which there are 16 figures, 3 tables, the list of used sources contains 21 names.

The aim of the work is to increase the reliability of clothing recognition.

The developed software module is designed for clothing recognition as a classic machine learning task on the Fashion-MNIST dataset. The structure of a spiking neural network for clothing recognition was developed, which is equivalent to a traditional neural network, i.e. it contains two fully connected layers of spiking neurons in the number of 128 and 10, respectively, and has 784 inputs and 10 outputs. The module was developed in the Python programming language in the Colab programming environment and using specialized libraries NumPy, Keras, TensorFlow and Matplotlib. The Fashion MNIST image set was used for training and testing the module, which contains 70,000 grayscale images in 10 categories with a resolution of 28 by 28 pixels. The developed software module for recognizing clothing items based on a spiking neural network has a recognition accuracy of 92.68% on the test set, and 90.60% on the analogue. That is, the recognition accuracy is increased by 2.08%.

Keywords: recognition, clothing items, Fashion-MNIST, spiking neural network, recognition accuracy.

ЗМІСТ

ВСТУП

Актуальність досліджень. Швидкий розвиток цифрового ринку сприяв впровадженню штучного інтелекту у програми та додатки, призначені для виконання різноманітних завдань. Нейронні мережі надають підприємствам необхідну гнучкість, щоб реагувати на нові тенденції та потреби клієнтів, одночасно підвищуючи їх продуктивність.

Машинне навчання відкриває практично безмежні можливості. На зростаючому конкурентному ринку, де нейронні мережі сприяють швидкому зростанню та підвищують ефективність бізнесу. В умовах дедалі жорсткішої ринкової конкуренції нейронні мережі допомагають швидко розвиватися та підвищувати ефективність бізнесу.

Розвиток нейронних мереж поступово стає світовим трендом, на який активно відгукуються компанії та підприємства різних сфер. Основними носіями додатків штучного інтелекту є: автоматизація, забезпечення безпеки мережі, відеоспостереження, управління ризиками, голосове управління та розробка ігор.

Останнім часом увага науковців та розробників все частіше переключається з дослідження традиційних нейронних мереж на дослідження нейромереж третього покоління, а саме – спайкінгових нейромереж. Спайкінгові нейронні мережі найбільш схожі на свої біологічні аналоги, тому що так само як біологічні нейромережі, використовують імпульсне кодування інформації. А тому очікувано від них бачити ширшу функціональність і більшу продуктивність. Але це треба доводити.

Довести переваги спайкінгових нейромереж можно тільки виконанням на них класичних завдань машинного навчання, причому з результатами, кращими за традиційні методи. Однією із таких класичних (еталонних) задач є задача класифікації предметів одягу на основі набору даних Fashion-MNIST.

Метою створення програмного застосунку є розробка спайкінгової нейромережі для розпізнавання предметів одягу на основі набору даних

Fashion-MNIST з тим, щоб порівняти результати її роботи з результатами традиційної нейромережі на цій задачі.

Метою дослідження є підвищення достовірності розпізнавання предметів одягу.

Об'єктом дослідження є процес комп'ютеризованого розпізнавання предметів одягу на основі нейромережі.

Предметом дослідження є алгоритми та програмні засоби розпізнавання предметів одягу на основі спайкінгової нейронної мережі та достовірність їх роботи.

Задачі дослідження:

1. Проаналізувати відомі методи розпізнавання предметів одягу та обрати напрямок досліджень;
2. Обгрунтувати вибір архітектури спайкінгової нейромережі;
3. Розробити структуру спайкінгової нейронної мережі для розпізнавання предметів одягу;
4. Розробити алгоритм роботи програмного модуля розпізнавання предметів одягу на основі спайкінгової нейромережі;
5. Здійснити програмну реалізацію модуля розпізнавання предметів одягу на основі спайкінгової нейромережі;
6. Провести тестування програмного модуля розпізнавання предметів одягу на основі спайкінгової нейромережі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗПІЗНАВАННЯ ПРЕДМЕТІВ ОДЯГУ

1.1 Постановка задачі

За своїми можливостями штучний інтелект можна порівняти з людським мозком. Ви надаєте йому інформацію про свою компанію та задаєте певні завдання. Технологія використовуватиме дані для навчання та адаптації, як навчаються люди. При цьому нейронні мережі працюють швидше і не допускають помилок через неуважність, втому тощо, що дозволяє економити гроші та час при наймі персоналу чи аутсорсингу для певних проектів.

Висока якість і точність результатів ШІ [1] не вимагає відпочинок чи сон і завжди працює з однаковою ефективністю. Нейронні мережі самостійно збирають і аналізують необхідні дані, забезпечуючи якісні і точні результати. Завдяки машинному навчанню штучний інтелект може виконувати обробку природної мови, розпізнавати шаблони та зображення, вирішувати проблеми та працювати безперебійно весь час.

Співробітники зможуть зосередитися на тому, що дійсно важливо, виконуючи повсякденну роботу, не відволікаючись на повсякденні завдання, які забирають багато часу та енергії, які будуть передані ШІ. Зрештою, це призводить до ефективнішої та швидшої роботи та прискорює процес зростання бізнесу.

У даній роботі ставиться задача розробити програмний модуль для розпізнавання предметів одягу із датасету Fashion-MNIST за допомогою спайкінгової нейронної мережі.

Дано: зображення у форматі jpeg, у градаціях сірого, з низькою роздільною здатністю 28 на 28 пікселів.

Потрібно: розпізнати на зображенні один із десяти предметів одягу.

Ідея: буде використано спайкінгову нейромережу для розпізнавання предметів одягу та порівняно достовірність її роботи з достовірністю традиційної нейромерережі.

1.2 Огляд відомих методів розпізнавання предметів одягу

Розвиток методів та алгоритмів штучного інтелекту для аналізу зображень характеризуються значним якісним стрибком останнім часом. Все це зумовлено великим попитом у різних сферах та мультизадачністю нейронних мереж, унаслідок чого вони все більше і більше впроваджуються в повсякденне життя та стають не від'ємною частиною нашого життя. Існує значна кількість нейронних мереж різних типів. Найдосвідченішою мережою є саме багатошаровий персептрон. Традиційним завданням, що пов'язане з обробкою зображень котре може бути вирішене за допомогою нейронних мереж, зокрема, персептрона, є завдання розпізнавання. Проте, можливості використання нейромережевих підходів не обмежуються лише розпізнаванням. Персептрон також використовують для боротьби із завадами, реконструкції і стиснення зображень. У той самий час персептрон рідко використовують для пошуку областей інтересу на зображеннях[2].

Незважаючи на високі якісні показники ефективності, яких можна досягти з використанням нейромережевих реалізацій, їхнім основним недоліком є значна кількість операцій обчислення, наслідком чого часто є недостатній рівень швидкодії і необхідність використання апаратних можливостей для швидких паралельних обчислень [2].

Хоча високу якість показників продуктивності можна досягти за допомогою реалізації на основі нейронних мереж, їх основним суттєвим недоліком є велика кількість обчислювальних операцій, результатом яких часто є недостатня швидкодія. А ще нейромережі вимагають використання апаратних можливостей для швидкого паралельного обчислення.

Задача розпізнавання предметів одягу є окремим випадком загальної задачі розпізнавання образів. Розглянемо різні методи розпізнавання образів і оберемо найкращий для нашої задачі.

Розпізнавання образів виконують різні методи, які використовуються для ідентифікації та класифікації закономірностей (рис. 1.1). Ці методи можна загалом розділити на статистичні, структурні та нейромережеві підходи [3]. Крім того, поширеними методами є зіставлення шаблонів та нечіткі моделі.

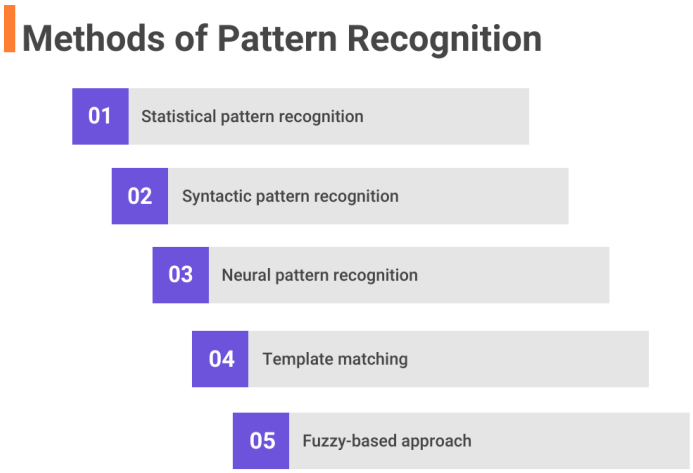


Рисунок 1.1 – Методи розпізнавання образів

Проведемо детальніший огляд цих методів:

1. Статистичне розпізнавання образів:

Цей метод розпізнавання образів використовує історичні статистичні дані, на яких навчаються алгоритми на основі прикладів та закономірностей. Статистичний метод накопичує спостереження та обробляє їх для визначення моделі. Отримана модель узагальнює зібрані спостереження та використовує правила до нових прикладів або наборів даних.

Концепція: Застосовує математичні та статистичні методи для класифікації образів на основі розподілів імовірностей.

Приклади: класифікація Байєса, лінійний дискримінантний аналіз (LDA), метод опорних векторів (SVM).

Кроки: Вилучення ознак, оцінка моделі, класифікація.

2. Синтаксичне (структурне) розпізнавання образів:

Синтаксичне розпізнавання образів містить складні шаблони, які ідентифікуються за допомогою ієрархічного алгоритму. Шаблони визначаються на основі того, як примітиви (напр., літери у слові) взаємодіють один з одним. Прикладом цього може бути випадок, коли примітиви об'єднуються у слова і речення.

Концепція: Зосереджується на зв'язках між ознаками для розпізнавання закономірностей. Особливо є корисним для складних структур.

Підхід: Ієрархічний алгоритм з категоризацією на підкласи.

3. Розпізнавання образів з використанням нейронних мереж:

У цьому методі використовуються штучні нейронні мережі (ШНМ) [4,5], які навчається на складних та нелінійних співвідношеннях між входом та виходом. Метод адаптується до даних та виявляє у них закономірності. Найпоширенішою та найефективнішою архітектурою у нейронних мережах є архітектура прямого розповсюдження. У цій архітектурі навчання відбувається методом зворотного поширення помилки. Це дуже схоже на те, як люди навчаються на своєму минулому досвіді та помилках. Модель на основі ШНМ оцінюється як найвитратніший метод розпізнавання шаблонів порівняно з іншими методами через обчислювальні ресурси, що задіяні в процесі.

Концепція: Використовує штучні нейромережі для розпізнавання образів.

Перевага: Гнучкий та ефективний в класифікації, особливо при використанні архітектур глибокого навчання.

4. Зіставлення шаблонів:

Зіставлення шаблонів - один з найпростіших методів розпізнавання образів. Тут подібність між двома образами визначається шляхом зіставлення невідомого зразка з еталонами. Такі методи часто використовуються в цифровій обробці зображень, де невеликі ділянки зображення зіставляються зі

збереженими зображеннями шаблонів. Деякі з реальних прикладів містять розпізнавання обличь, обробку медичних зображень та навігацію роботів.

Концепція: Порівнює вхідний образ зі збереженим шаблоном для визначення ступеня подібності.

Застосування: Використовується при роботі з подібними типами об'єктів, такими як криві або фігури.

5. Нечіткі моделі:

У нечіткому методі набір шаблонів розділяється на основі подібності ознак шаблонів. Коли унікальні ознаки шаблону виявлені правильно, дані легко можна класифікувати у цей відомий простір ознак. Навіть зорова система людини іноді не розпізнає певні компоненти, незважаючи на тривале розглядання об'єктів. Те саме стосується і цифрового світу, де алгоритми не можуть точно визначити природу об'єкта. Отже, нечіткий метод спрямований на класифікацію об'єктів по кількох подібних ознаках у шаблонах.

Концепція: Містить нечітку логіку для обробки неточних або неповністю визначених даних.

Застосування: Корисний для реальних задач розпізнавання, де поширена нечіткість.

Для реалізації модуля розпізнавання предметів одягу було обрано метод розпізнавання образів з використанням нейронних мереж.

1.3 Обгрунтування вибору аналогів розробленого модуля розпізнавання предметів одягу

Даний аналог міститься у вільному доступі в середовищі GitHab [6]. Ключові особливості це даний продукт розроблений за допомогою хмарних технологій з використанням Google Colab.

Із основних недоліків даного продукту можна виділити низьку швидкодію, високий поріг входу (тобто людина повинна вміти кодувати аби розібратися з даною програмою, також відсутність структуризації вносити свої

корективи). Основними елементами, що містить аналог, це підключення бібліотек наведено на рисунку 1.2.

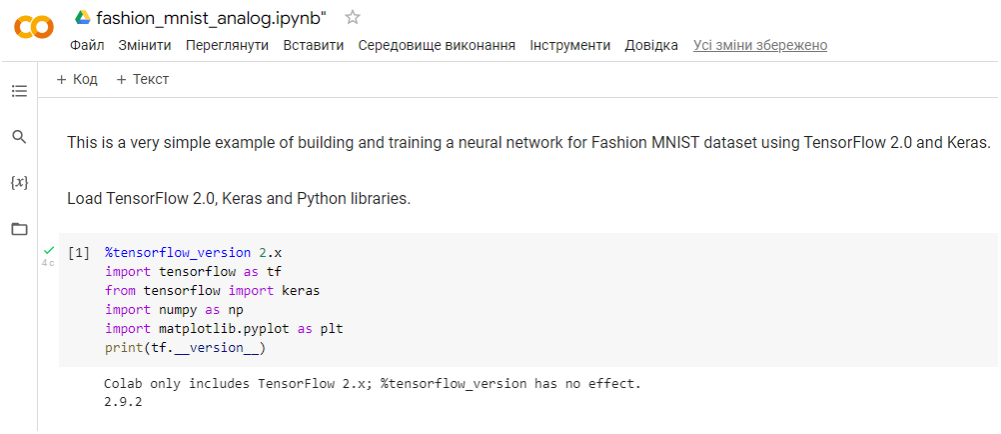


Рисунок 1.2 – Підключення бібліотек

Побудована нейромережа містить два шари в 128 та 10 вихідних нейронів котрі відповідають за клас одягу, що розпізнається наведено на рисунку 1.3.

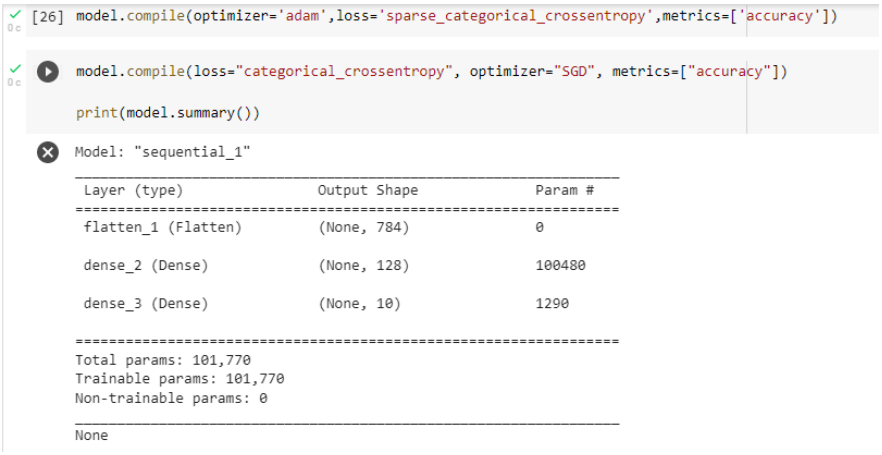


Рисунок 1.3 – Створена нейромережа аналога

При навчанні було з’ясовано що процес навчання займає дуже багато часу навіть попри відсутність інтерфейсу, та високий поріг входження нейронна мережа навчається вкрай повільно. Також не менш важливим є розпізнавання зображень (рис. 1.4). Аби його змінити в аналогові, потрібно змінювати код і це є досить поганим рішенням, адже не кожен користувач знає

що саме потрібно змінити.

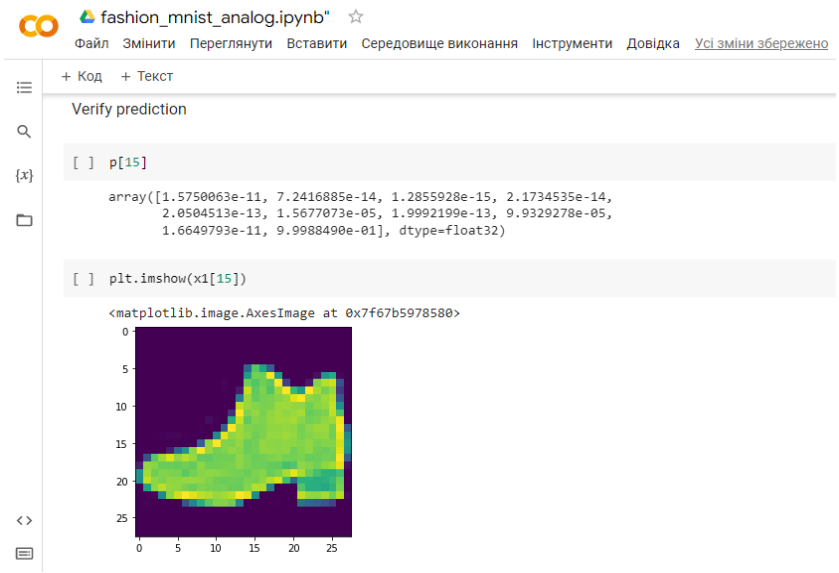


Рисунок 1.4 – Розпізнавання аналогу

Таким чином, основним недоліком аналогу є невисока достовірність розпізнавання предметів одягу: близько 90%. Звідси випливає мета бакалаврської кваліфікаційної роботи - підвищення достовірності розпізнавання предметів одягу.

1.4 Висновок до розділу 1

У розділі описано детальну постановку задачі розпізнавання предметів одягу. Було розглянуто такі методи розпізнавання образів: статистичні, синтаксичні, нейромережеві, зіставлення шаблонів та нечіткої логіки. Як найбільш перспективний і застосовний до даної задачі було обрано нейромережевий метод. Крім цього, було обгрунтовано вибір аналогу до розробленого модуля розпізнавання предметів одягу, який реалізовано на багатошаровому персептроні.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ ПРЕДМЕТІВ ОДЯГУ НА ОСНОВІ СПАЙКІНГОВОЇ НЕЙРОННОЇ МЕРЕЖІ

2.1 Аналіз роботи спайкінгової нейронної мережі для розпізнавання предметів одягу

Останнє десятиліття стало свідком зростання можливостей штучних нейронних мереж (ШНМ) від багатошарового персептрона [2] першого покоління (БШП) до багатьох найсучасніших методів глибоких нейронних мереж [7] другого покоління (ГНМ). Це досягнення значною мірою залежить від великої кількості анотованих даних і широкої доступності високопродуктивних обчислювальних пристроїв, а також графічних процесорів (GPU) загального призначення. Незважаючи на цей великий прогрес, ШНМ все ще відстають від біологічних нейронних мереж з точки зору енергоефективності та можливостей онлайн-навчання. Було зроблено багато спроб зменшити енергоспоживання традиційних моделей глибокого навчання.

Незважаючи на те, що розробку ШНМ/ГНМ історично надихнув людський мозок, вони принципово відрізняються за структурою, нейронними обчисленнями та правилами навчання порівняно з біологічною нейронною мережею. Тому останнім часом серед дослідників більш популярними стають спайкінгові нейронні мережі (SNN) [8], які часто називають третім поколінням нейронних мереж, які можуть стати проривом у вузьких місцях ШНМ. Варто згадати використання SNN на нейроморфних апаратних засобах [9], таких як TrueNorth, Loihi, SpiNNaker, NeuroGrid тощо, і є перспективним підходом до проблеми споживання енергії. У SNN, наприклад у біологічних нейронних мережах, нейрони спілкуються один з одним за допомогою дискретних електричних сигналів (імпульсів), які називаються спайками, і працюють у безперервному часі.

Завдяки своїй функціональній схожості з біологічними нейронними мережами, SNN можуть охоплювати розрідженість, виявлену в біології, і дуже сумісні з часовим кодом [10]. Хоча SNN все ще відстають від ГНМ з точки зору їх продуктивності, розрив зникає для деяких завдань, тоді як SNN зазвичай потребують набагато менше енергії для роботи. Однак SNN все ще важко навчити загалом, головним чином через їх складну динаміку нейронів і недиференційований характер спайкінгових операцій. Порівняння між біологічними нейронними мережами, ШНМ та SNN наведено в таблиці 2.1.

Таблиця 2.1 - Порівняння властивостей між біологічними нейронними мережами, традиційними ШНМ і SNN

Властивості	Біологічні НМ	ШНМ	SNN
Представлення інформації	Спайки	Скаляри	Спайки
Парадигма навчання	Синаптична пластичність	Зворотне поширення	Пластичність/ Зворотне поширення
Платформа	Мозок	VLSI	Нейроморфний VLSI

Нейрони є основними робочими одиницями нервової системи, які обробляють інформацію, поширюючи електрохімічні сигнали через потенціали дії. Нейрони не є електрично нейтральними відносно позаклітинної рідини через присутність у них іонів. Іони постійно входять і виходять з клітини через мембрану, яка може динамічно змінювати свою електричну проникність за допомогою зовнішніх електрохімічних сигналів. Потік іонів, що входять у клітину та виходять із неї, спричиняє віртуальний

струм, що протікає через мембрану, переважно пов’язаний з іонами Na+, K+ та Cl− [10].

На рисунку 2.1 показано типову структуру нейрона з чотирма основними компонентами [11]: дендритами, сомою, аксоном і синапсом. Дендрити — це короткі нервові закінчення, які можна розглядати як вхідні дані нейрона. Вони перетворюють хімічні сигнали, що передаються нейромедіаторами, що вивільняються з пресинаптичного нейрона, в електричні сигнали. Сома — це тіло клітини, де інтегровані мембранні потенціали, що поширюються від синаптичних входів, що в кінцевому рахунку визначає, чи запускає постсинаптична клітина потенціали дії перед передачею до аксона. Така взаємодія впливів називається нейронною інтеграцією. Аксон несе потенціал дії до інших нервових клітин. Для швидкого перенесення потенціалу дії на великі відстані без ослаблення деякі аксони вкриті мієліновою оболонкою.

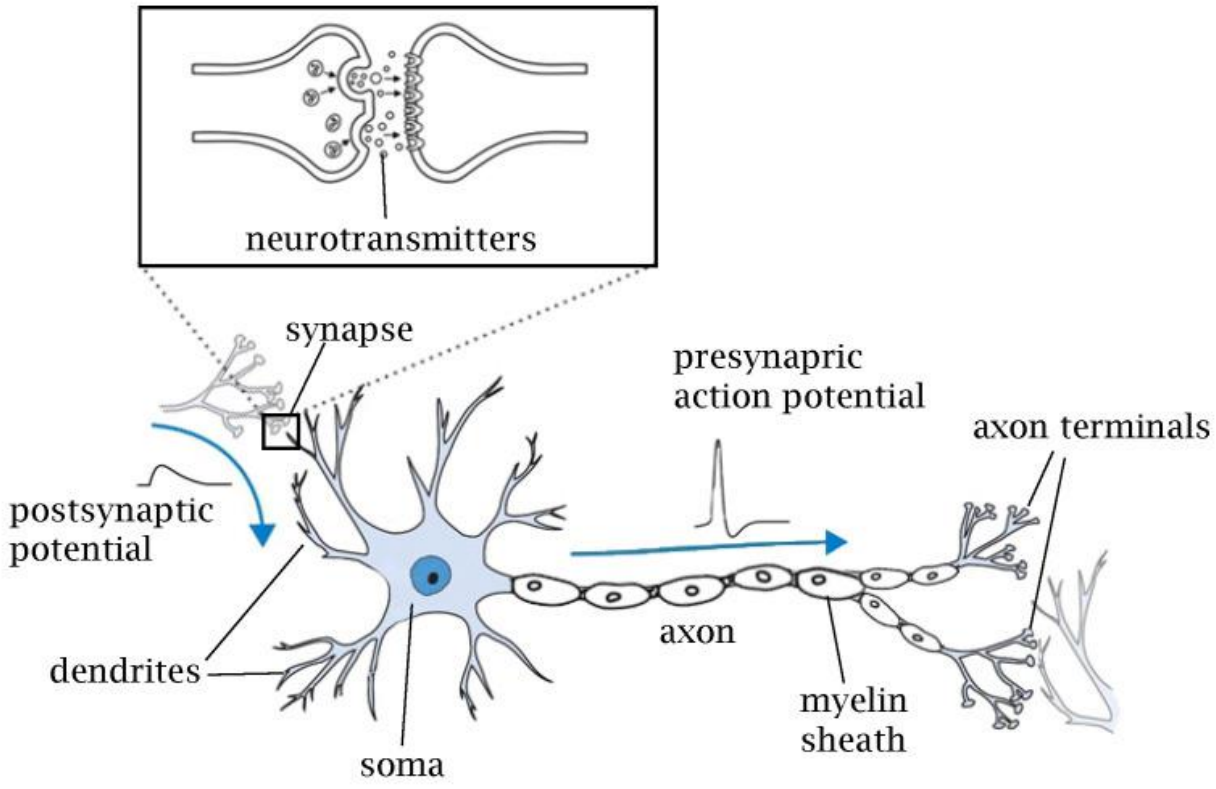


Рисунок 2.1 – Типова будова біологічного нейрона та синапсу

Синапси — це контактна структура для передачі інформації, яка з’єднує нейрони в нейронній мережі. Синапси можна умовно розділити на хімічні та електричні. У хімічних синапсах немає прямого контакту між пре- і постсинаптичними нейронами. Сигнал від пресинаптичного нейрона передається через нейромедіатори, що містяться в синаптичних гранулах, випущених у синаптичну щілину. Нейромедіатори зв’язуються з рецепторами в постсинаптичній клітині, безпосередньо змінюючи потенціал мембрани або активуючи внутрішньоклітинні вторинні месенджери для передачі інформації. Цей тип передачі повільний, але посилює сигнал і може подовжити вплив вхідного стрибка. Хімічні синапси можна підрозділити на збудливі та гальмівні синапси. Збудливі синапси — це синаптичні зв’язки, які деполяризують постсинаптичні клітини через синаптичну передачу та сприяють запуску потенціалів дії. Гальмівні синапси - це синаптичні зв'язки, які гіперполяризують постсинаптичні клітини шляхом синаптичної передачі та гальмують розвиток потенціалів дії.

Нейрон, заснований на частоті, моделює активність нейрона лише за макроскопічною ознакою, швидкістю спрацьовування r, незалежно від зміни мембранного потенціалу чи часу спалаху. Перший штучний нейрон зі швидкісним кодуванням, відомий як формальний нейрон або пороговий логічний блок. На основі формального нейрона в роботі представлено персептрон, використовуючи ступінчасту функцію Хевісайда як функцію активації. Ці нейрони першого покоління запускають двійкові сигнали, коли сума вхідних сигналів досягає порогу нейрона. Пізніше ця концепція була розширена для використання безперервних функцій активації, включаючи сигмоїду або функцію гіперболічного тангенса, для роботи з аналоговими входами та виходами; отже, це дозволило навчити нейронну мережу за допомогою потужного алгоритму зворотного поширення, який використовує градієнтний спуск. Через доведену здатність достатньо великої нейронної мережі штучних нейронів як завгодно добре апроксимувати будь-яку аналогову функцію (універсальна теорема про наближення стверджує, що

мережа прямого зв’язку з одним прихованим шаром із кінцевою кількістю нейронів може апроксимувати безперервні функції за припущень на неполіноміальній функції активації; сигмоїдальна функція активації та ReLU також доведено, що відповідають теоремі універсальної апроксимації), штучні нейронні мережі широко використовуються як потужний інструмент обробки інформації в машинному навчанні .

Зараз Rectified Linear Unit (ReLU) та його варіанти зазвичай використовуються як нелінійність, оскільки вони, як правило, демонструють кращу продуктивність збіжності, ніж сигмоподібна функція активації. Таке формулювання групи нейронів на основі швидкості часто називають повністю зв’язаним шаром. Сучасна архітектура нейронних мереж об’єднує варіант цього рівня для створення дуже глибоких мереж нейронів, які часто називають глибокими нейронними мережами (ГНМ).

Нейронні мережі зазвичай називають глибокими, якщо вони мають принаймні два прихованих шари обчислення нелінійних перетворень вхідних даних. Одним із часто використовуваних будівельних блоків ГНМ є згортковий шар [12]. Згортковий шар — це окремий випадок повністю зв’язаного шару, який реалізує розподіл ваги для обробки даних, що мають відому сіткову топологію, наприклад, зображень. Завдяки такому індуктивному зміщенню згорткові нейронні мережі (ЗНМ) можуть більш розумно використовувати просторову кореляцію сигналу. Репрезентативні властивості ранніх шарів у ЗНМ подібні до властивостей відповіді нейронів у первинній зоровій корі (V1), яка є першою корковою областю в зоровій ієрархії мозку приматів. ЗНМ володіють двома ключовими властивостями, які роблять їх надзвичайно корисними для застосування зображень: просторово розподілені ваги та просторове об’єднання. Цей тип мережі вивчає функції, які не змінюються, тобто фільтри, які корисні для всього зображення (через те, що статистика зображення є стаціонарною). Рівні об’єднання відповідають за зменшення чутливості вихідного сигналу до незначного вхідного зсуву та спотворень і збільшення поля прийому для наступних шарів. Починаючи з

2012 року, одним із найпомітніших результатів у глибокому навчанні є використання ЗНМ для досягнення значного вдосконалення проблеми класифікації ImageNet. На основі цього технологічного прориву в класифікації зображень були запропоновані різні вдосконалення для мережевих архітектур у моделях бачення. Хоча ШНМ були надзвичайно успішними в багатьох програмах, включаючи виявлення об'єктів, сегментацію зображення і розпізнавання дій, вони все ще обмежені в тому, як вони мають справу з часовою інформацією.

Моделі спайкінгових нейронів [10].

Здатність одночасно реєструвати активність кількох клітин привела до ідеї, що різниця в часі між спайками в різних нейронах і час самого спайку можуть мати функціональне значення. Оскільки модель частоти генерації спайків не може впоратися з проблемою цієї перспективи, була досліджена модель, що описує час спайків і зміну підпорогового мембранного потенціалу. Модель, яка обробляє генерацію таких спайків, відрізняється від моделі частоти генерації та називається моделлю спайків. Такі моделі нейронів зазвичай виражаються у формі звичайних диференціальних рівнянь. На рисунку 2.2 зображено відмінності між біологічним нейроном, штучним нейроном і спайкінговим нейроном.

Було запропоновано різноманітні моделі імпульсних нейронів, і вони відображають компроміс між біологічною точністю та обчислювальною здійсненністю (рис. 2.3). Вибір відповідної моделі залежить від вимог користувача. Моделі нейронів на основі спайків розглядаються щодо обчислювальної ефективності та біологічної правдоподібності.

Для прикладу розглянемо спайкінгову модель нейрона Leaky Integrate and Fire (LIF).

LIF – це модель, у якій вхідний струм інтегрується з часом, поки мембранний потенціал не досягне порогового значення без урахування поведінки біологічних іонних каналів. Ще її називають моделлю інтегрально-імпульсною (IF).

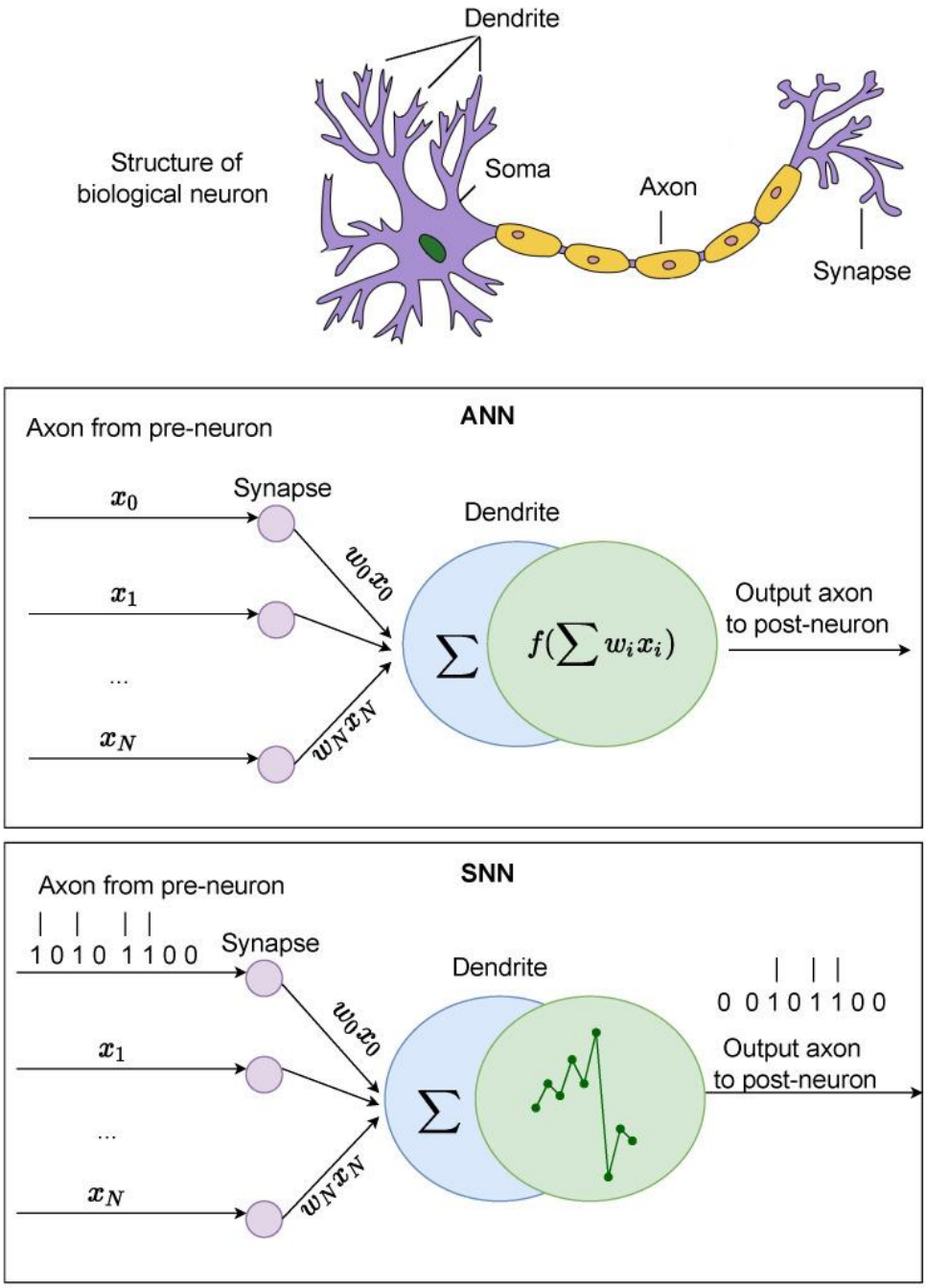


Рисунок 2.2 – Порівняння біологічного нейрона, штучного нейрона та спайкінгового нейрона

Наявність витоку інтегрально-імпульсної моделі (LIF) відображає дифузію іонів, яка відбувається через мембрану, коли певна рівновага не досягнута в клітині, вводячи термін «витік» до моделі IF. Завдяки своїй простоті та низькій обчислювальній вартості модель LIF та її варіанти є одним із широко використовуваних прикладів моделі спайк-нейрона.

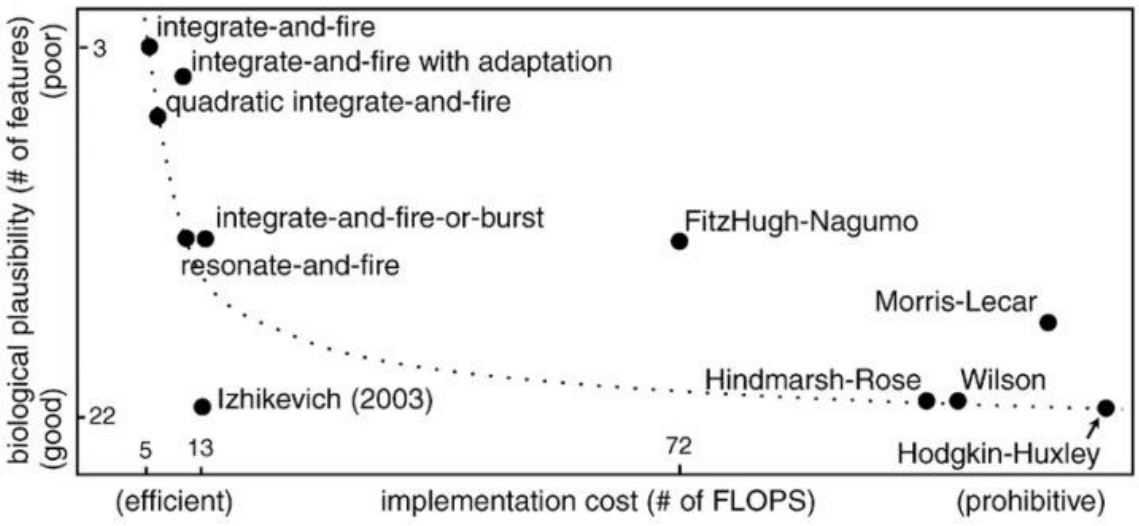


Рисунок 2.3 – Порівняння моделей спайкінгових нейронів з точки зору вартості реалізації та біологічної вірогідності

Динаміка моделі LIF представлена наступним рівнянням:

$$C_m \frac{dv_m}{dt} = -G_L\left(v_m - E_L\right) + I_{syn}\left(t\right)$$
$$\text{if} \quad v_m \geq v_\theta, \quad v_m \leftarrow v_{peak} \ \text{then} \quad v_m \leftarrow v_{reset} \tag{2.1}$$

де vθ — порогова напруга, vpeak — потенціал дії, а vreset — мембранний потенціал відновлення. Коли напруга досягає порогового значення vθ, яке зазвичай використовується для спрощення, нейрон випускає спайк, а потім напруга скидається до нуля протягом рефрактерного періоду τref, який обмежує частоту спрацьовування нейрона.

Коли синаптичний вхідний струм постійний (Isyn(t)=I) і vreset=0, ми можемо визначити мембранний потенціал наступним чином:

$$v_m\left(t\right) = R_m I\left(1 - \exp\left(-\frac{t}{\tau_m}\right)\right) \tag{2.2}$$

де Rm – опір мембрани (МОм), τm=RmCm – постійна часу мембрани. Оскільки нейрон запускає сплеск, коли мембранний потенціал досягає порогу, час першого сплеску t(1) можна знайти, встановивши vm(t)=vθ:

$$t^{(1)} = \tau_m \ln \frac{R_m I}{R_m I - v_\theta} \tag{2.3}$$

Таким чином, стабільну швидкість стрільби можна знайти як:

$$f = \left(\tau_{ref} + \tau_m \ln \frac{R_m I}{R_m I - v_\theta} \right)^{-1} \tag{2.4}$$

Теоретично можна навчити глибоку нейронну мережу, використовуючи рівняння (2.4) як статичну нелінійність і зробити розумну апроксимацію мережі в спайк-нейронах. Інтуїтивно зрозуміло, особливо коли τref=0,τm=1,Rm=1 і vθ=1, швидкість запуску нейрона, що відповідає вхідному струму, поводиться подібно до функції активації ReLU в ШНМ. Ця функція часто використовується для перетворення ШНМ на SNN.

Навчання в нейронних мережах передбачає модифікацію з’єднань нейронів. На відміну від ШНМ, які можна успішно навчити за допомогою стохастичного градієнтного спуску та зворотного поширення, SNN все ще не мають надійних методів навчання. Власні методи навчання SNN можна класифікувати на:

– контрольоване навчання з градієнтним спуском і спайкінговим зворотним поширенням,

– неконтрольоване навчання з локальним правилом навчання в синапсі (наприклад, пластичність, залежна від часу спайку), і

– навчання з підкріпленням із сигналом винагороди/помилки з використанням винагороди модульована пластичність.

Синаптична пластичність — це біологічний процес, за допомогою якого специфічні моделі синаптичної активності призводять до змін синаптичної сили. Синаптична пластичність була вперше запропонована як механізм навчання та пам'яті на основі теоретичного аналізу. Хоча правило локального навчання в синапсі вважається біологічно більш правдоподібним, ефективність навчання зазвичай нижча, ніж навчання під наглядом. Альтернативним підходом до непрямого навчання SNN є перетворення ШНМ у SNN. Серед цих методів найсучасніші результати в основному отримані в результаті перетворення моделі з ШНМ.

Зворотне поширення на основі спайків.

Подібно до алгоритму зворотного розповсюдження для ШНМ, SpikeProp призначений для визначення набору бажаних моментів часу запуску всіх вихідних нейронів на постсинаптичних нейронах для заданого набору вхідного шаблону. Методи, засновані на подіях, включаючи SpikeProp, мають похідний термін, визначений лише навколо часу спрацьовування, тоді як ігнорують часовий ефект імпульсного сигналу. Посилання пропонує вдосконалений метод SpikeProp [13] під назвою SuperSpike, який використовує похідну мембранного потенціалу замість спайку, що дозволяє навчати модель без спайків. SuperSpike використовує відстань Ван Россума між вихідною та бажаною серією спайків як функцію втрат, тоді як SpikeProp використовує похибку суми квадратів. Нижче показано функцію втрат для мережі в інтервалі часу $t \in [0,T]$.

$$L = \frac{1}{2} \int_0^T (\alpha \times (\mathbf{s}(t) - \hat{\mathbf{s}}(t)))^2 dt \tag{2.5}$$

де α — нормалізоване гладке часове ядро згортки, s — вихідна серія спайків, а $s^{\wedge}$ — цільова серія спайків. Тут ланцюг спайків представлений як $s(t)=\sum t_k<t \delta(t-t_k)$.

Під час обчислення похідної рівняння (2.1) щодо синаптичних ваг з’являється проблематичний терн ∂s/∂w, який містить дельта-функцію Дірака. Щоб уникнути цього терміну, ланцюг спайків апроксимується безперервною допоміжною функцією мембранного потенціалу моделі LIF.

$$\frac{\partial s}{\partial w} \approx \frac{\partial \sigma(v_m)}{\partial w} = \sigma'(v_m) \frac{\partial v_m}{\partial w} \tag{2.6}$$

де σ(x)=x/(1+|x|) представляє швидку сигмоїду. Тут ми додатково наближаємо ∂vm∂w≈ϵ×s нормалізованим плавним ядром часової згортки ϵ.

$$\frac{\partial L}{\partial w} = \int_0^T \alpha \times (\mathbf{s} - \hat{\mathbf{s}})\ \alpha \times (\sigma'(v_m)(\epsilon \times \mathbf{s}^{pre}))\, dt \tag{2.7}$$

де α×(s−sˆ) є сигналом помилки, а α×(σ′(vm)(ϵ×spre)) є слідом синаптичної відповідності.

SLAYER розподіляє кредит помилки назад у часі, щоб усунути недолік методів, заснованих на подіях. SLAYER передбачає апроксимацію стохастичного імпульсного нейрона для моделі IF з рефрактерною реакцією та може одночасно вивчати як синаптичні ваги, так і аксональні затримки.

$$\frac{\partial L}{\partial w} = \int_0^T \rho(t)\,(\alpha \odot e)\,(\alpha \times \mathbf{s}^{pre})\, dt \tag{2.8}$$

де ρ(t) — функція щільності ймовірності, яку можна сформулювати за допомогою функції швидкості виходу спайків, ⊙ — поелементна кореляція в часі, а e — оцінка похибки зворотного поширення.

Спайкінгове кодування.

Оскільки SNN використовують спайки та послідовності спайків для передачі інформації, кодування реальних даних у спайки є суттєвим кроком у створенні SNN. Хоча те, як інформація кодується в спайки в біології, є однією з найбільших невирішених проблем у нейронауці. Дві основні схеми кодування, частотне кодування та часове (імпульсне) кодування, можна знайти в багатьох видах літератури. Крім того, слід зазначити, що деякі датчики, такі як датчик динамічного бачення (DVS), можуть створювати необроблені послідовності спайків.

Частотне кодування.

Схема частотнлгл кодування базується на середній кількості спайків за час; інформація кодується кількома спайками, що генеруються протягом певного часового вікна. Залежно від різних схем усереднення, існує кілька способів визначення частоти, наприклад середнє значення за часом або середнє значення за кілька повторів.

Цю частоту спрацьовування можна використовувати як вхідні дані для моделей нейронів на основі частоти, тобто ШНМ, де функція активації представляє криву частота-струм (FI).

Часове кодування.

Схема часового кодування базується на точному часовому розрізі спайків, де більш помітна інформація кодується як перші сплески. Порівняно зі схемою кодування за частотою, часове кодування створює набагато рідші сплески, оскільки інформацію представляє час сплеску, а не частота сплеску. Хоча часовий код дозволяє представити характеристики вхідних даних за допомогою невеликих груп нейронів, він містить вразливість до вхідного шуму або часового тремтіння.

Під час кодування зображення кожне окреме значення пікселя в діапазоні від 0 до 255 може бути просто використано для створення часу сплеску, пропорційного яскравості пікселя. Наприклад, піксель із нормалізованою яскравістю 0,1 відповідає часу сплеску при t=0,1. У сірому

зображенні білі пікселі (яскравість дорівнює 1 або 255) не викликають спайки, оскільки можна вважати, що вони не несуть ніякої інформації.

2.2 Розробка структури спайкінгової нейронної мережі для розпізнавання предметів одягу

Спайкінгові нейронні мережі (SNN) – третє покоління нейромереж. Основною різницею з іншими поколіннями нейромереж є те, що нейрони комунікують один з іншим за допомогою імпульсів (спайків). Ідея полягає у тому, що нейрон не спрацьовує кожного разу, коли на нього надходить сигнал, а спрацьовує при досягненні потенціалом його мембрани (характеристика кожного нейрону) певного значення порогу «збудження». Коли нейрон активується, він надсилає імпульс усім наступним, пов’язаним з нам, нейронам [14].

Найважливішими перевагами SNN є:

1. SNN є динамічними, отже придатні для обробки динамічних процесів (напр., розпізнавання звуків);
2. SNN просто навчати, оскільки навчання потребує лише вихідний шар нейронів;
3. SNN донавчаються в процесі роботи.

Зазвичай спайкінгова нейромережа складається з 3 шарів нейронів: вхідний, прихований та вихідний [15]. Структура SNN наведена на рисунку 2.4.

У частотних SNN використовується сигнал, що приймає значення, залежне від частоти імпульсів певної групи нейронів (ваги нейронів, власне, і є формою подання частоти). Проте, середня частота імпульсів у послідовності є доволі поганим варіантом подання інформації, оскільки різні види стимуляції можуть приводити до однакової середньої частоти імпульсів.

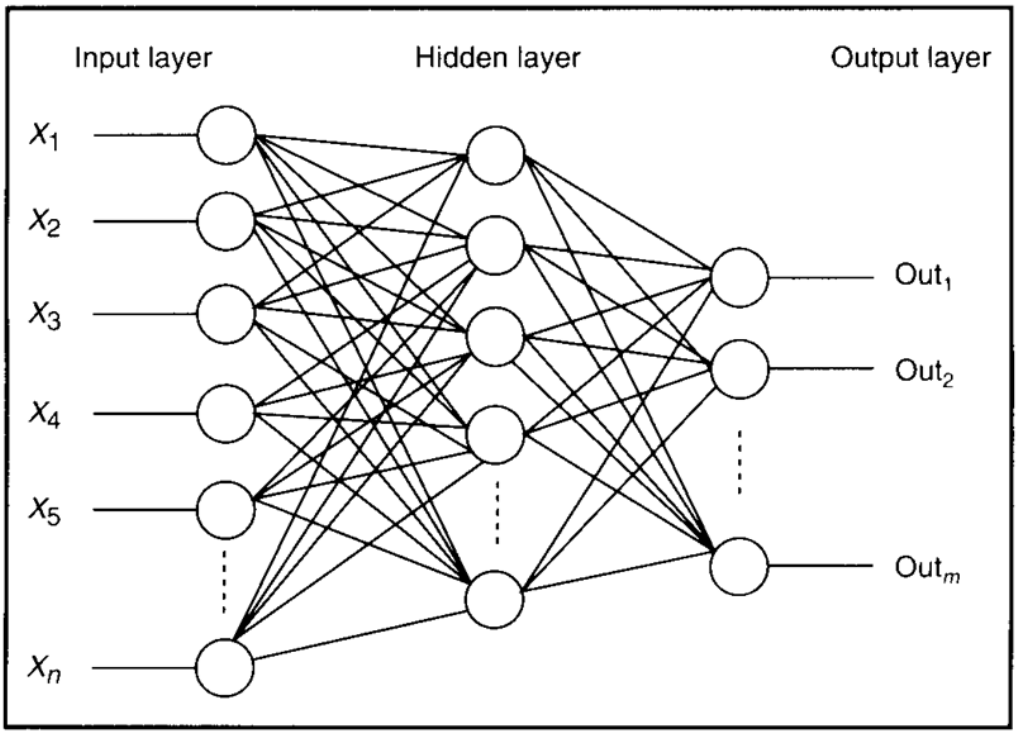


Рисунок 2.4 – Структура спайкінгової нейронної мережі

Для позбавлення від цього недоліку в SNN використовуються такі види подання інформації:

- фазовий (часовий) – інформація про сигнал задається точним (або в межах деякого вікна) положенням імпульсів у часі (щодо будь-якого загального опорного ритму головного мозку);
- синхронний (позиційний / просторовий / популяційний) – інформація про сигнал задається синхронною активністю різних груп нейронів, і, як наслідок, синхронним (або в межах певного вікна) появою спайків на окремих виходах мережі (наприклад, реагуючі на високі і низькі частоти слухові рецептори вуха равлики, знаходяться в різних зонах);
- час до першого імпульсу – інформація про сигнал задається часом появи першого імпульсу на будь-якому виході;
- порядковий – інформація про сигнал задається порядком отримання імпульсів на виходах нейромережі;

- інтервальний (затримковий) – інформація про сигнал задається відстанню у часі між імпульсами, які отримуються на виходах мережі;
- резонансний – інформація про сигнал задається щільною послідовністю імпульсів (чергою), яка приводить до виникнення резонансу (одиночні імпульси загасають і не вносять жодного внеску в передачу інформації).

На перший погляд, підхід SNN може здатися кроком назад – від безперервної, свого роду аналогової картини, до імпульсної, двійкової. Однак, перевага SNN полягає в тому, що імпульсний підхід дозволяє оперувати даними, з огляду на відстані між нейронами і тривалість поширення сигналу, тобто в контексті простору і часу. За рахунок цього SNN мережі набагато краще пристосовані для обробки даних від справжніх сенсорів.

Просторовий аспект відображає той факт, що нейрони в першу чергу з'єднані з найближчими сусідами, і тому фрагменти введення обробляються окремо.

Часовий аспект відповідає тому, що тренувальні імпульси приходять з різними затримками, і та інформація, яку ми «втрачаємо» при переході від безперервного сигналу до імпульсного, насправді зберігається в інформації про затримку імпульсів один відносно одного. Доведено, що спайкінгові нейрони є більш потужними обчислювальними елементами, ніж традиційні штучні нейрони.

З огляду на те, що SNN в теорії є більш потужними нейромережами, ніж мережі другого покоління, залишається дивуватися, чому не помітно їх широкого застосування. Основна проблема практичного використання SNN – навчання. Незважаючи на наявність методів неконтрольованого біологічного навчання (без учителя), таких як Hebbian і STDP, поки невідомі ефективні методи навчання SNN, які б забезпечували б більш високу продуктивність, ніж мережі другого покоління.

Через проблеми з диференціюванням імпульсів, SNN неможливо навчати, використовуючи градієнтний спуск, не втрачаючи точну часову інформацію про імпульси. Тому, щоб ефективно використовувати SNN для реальних завдань, необхідно розробити відповідні методи контрольованого навчання. Це важке завдання, враховуючи біологічний реалізм цих мереж, вона передбачає точне розуміння того, як вчиться людський мозок.

Для покращення часу та точності результату також використовують архітектуру рекурентних нейромереж. Це дозволяє оброблювати серії імпульсів послідовно, не чекаючи закінчення реагування на попередній. Це, в свою чергу, покращує функціонування нейронної мережі при обробці великих об’єктів, які розбиваються на менші імпульси.

2.3 Розробка алгоритму роботи програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі

Відповідно до мети роботи та постановки задачі було розроблено алгоритм програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі, представлений на рисунку 2.5.

Першим кроком в програмному модулі розпізнавання предметів одягу на основі спайкінгової нейронної мережі буде завантаження необхідних бібліотек (блок 1), а другим кроком буде завантаження набору даних Fashion MNIST (блок 2).

Далі відбувається нормалізація даних (блок 3), тобто діапазон яскравосте від 0 до 255 перетворюється у дапазон значень від 0 до 1, які необхіні для подачі на входи нейромережі.

Після цього відбувається створення та навчання простої традиційної (неспайкінгової) нейромережі (блок 4) для класифікації зображень Fashion MNIST. Потім оцінюється достовірність роботи традиційної нейромережі на тестовій вибірці (блок 5).

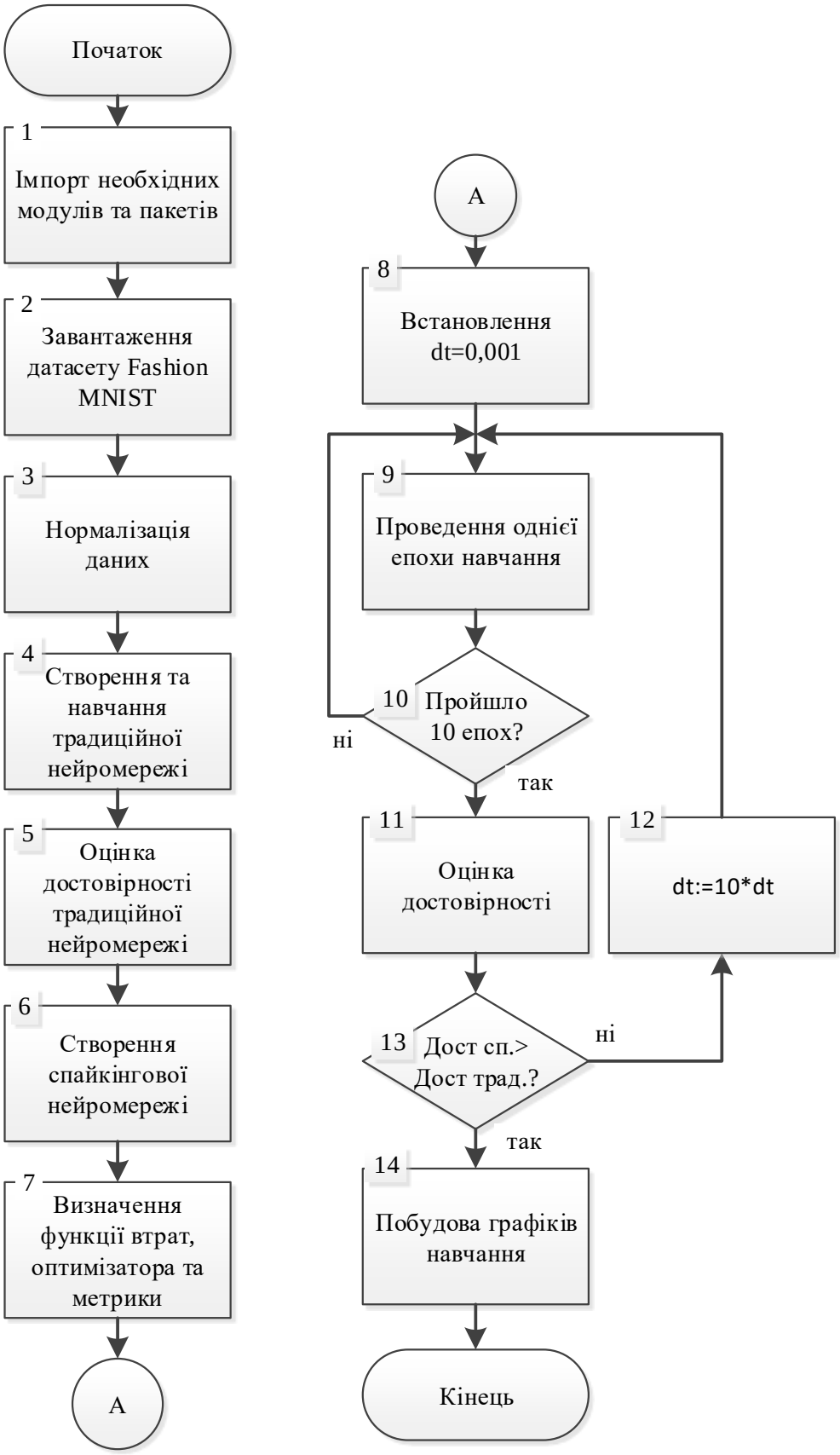


Рисунок 2.5 – Схема алгоритму роботи програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі

Далі створюється еквівалентна (по кількості шарів і нейронів) спайкінгова нейронна мережа (блок 6), для неї задаються функція втрат, оптимізатор та метрика (блок 7).

Потім встановлюється часовий крок роботи спайкінгової нейромережі dt=0.001 (блок 8). Після цього проводиться одна епоха навчання спайкінгової нейромережі (блок 9). Умовний блок 10 задає кількість епох навчання, протягом яких потрібно навчати спайкінгову нейромережу. Після цього оцінюється достовірність роботи спайкінгової нейромережі на тестовій вибірці (блок 11).

У блоці 13 перевіряється чи достовірність спайкінгової нейронної мережі більше достовірності традиційної нейромережі. Якщо так, то будуються графіки (блок 14) зміни параметрів в процесі навчання і алгоритм завершує свою роботу. А якщо ні, то параметр dt збільшується в 10 разів (блок 12) і знову відбувається процес навчання спайкінгової нейромережі.

Даний алгоритм боло програмно реалізовано та описано у п. 3.3.

2.4 Висновок до розділу 2

У розділі було проведено аналіз роботи спайкінгової нейронної мережі для розпізнавання предметів одягу. Розглянуто подібність спайкінгових нейромереж до біологічних нейромереж, розглянуто моделі спайкінгових нейронів та методи навчання спайкінгових нейромереж. Розроблено структуру спайкінгової нейронної мережі для розпізнавання предметів одягу, яка еквівалентна традиційній нейромережі, тобто містить два повнозв’язних шари спайкінгових нейронів у кількості відповідно 128 і 10, має 784 входи та 10 виходів. Також було розроблено алгоритм роботи програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ РОЗПІЗНАВАННЯ ПРЕДМЕТІВ ОДЯГУ НА ОСНОВІ СПАЙКІНГОВОЇ НЕЙРОННОЇ МЕРЕЖІ

3.1 Обґрунтування вибору мови програмування та спеціалізованих бібліотек

Для реалізації програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі було обрано мову програмування Python [16]. Такий вибір продиктовано тим, що ця мова зараз найпоширеніша при програмуванні систем штучного інтелекту і має багато спеціалізованих бібліотек для роботи з нейронними мережами.

Як середовище програмування було обрано Colab [17], або Colaboratory, який дає змогу створювати й виконувати код Python у веб-переглядачі, має зручний інтерфейс (рис.3.1) та такі переваги :

- не потрібні попередні налаштування,
- є доступ до графічних процесорів,
- не потрібна оплата за користування,
- має легкий спільний доступ.

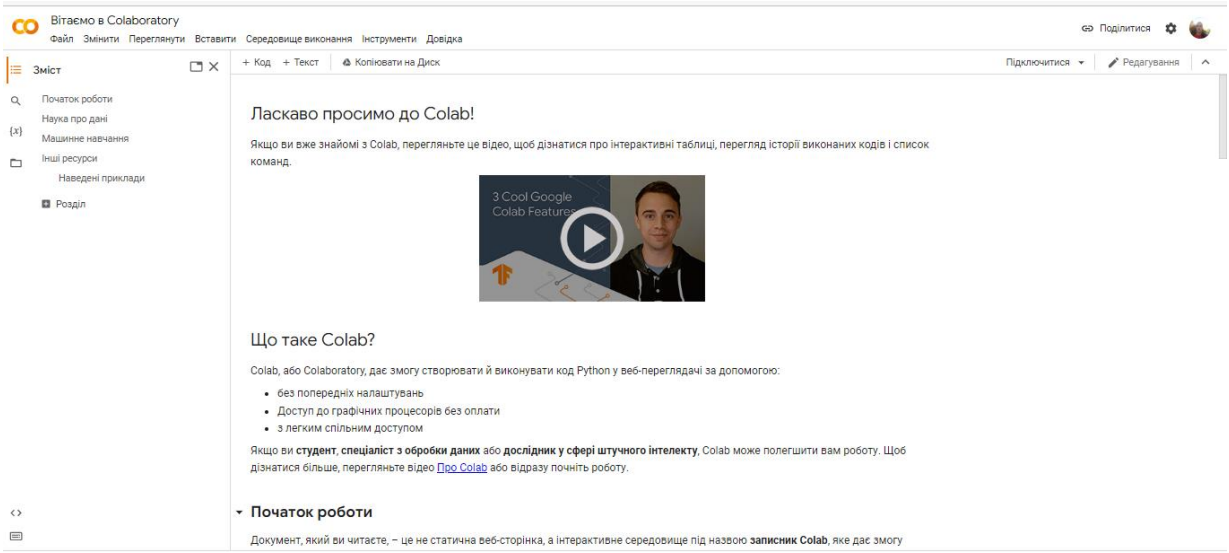


Рисунок 3.1 – Інтерфейс Google Colab

Підходить для студентів, спеціалістів з обробки даних або дослідників у сфері штучного інтелекту. Colab може полегшити роботу та додасть додатковий об’єм оперативної пам’яті (рис. 3.2). Розроблений проект не потребуватиме ніяких сторонніх завантажень бібліотек чи фреймів для вашого проекту. Також розроблений проект підійде для будь-якої операційної системи.

Почати роботу в Colab дуже легко - достатньо відкрити проект в хмарі. Документ, який ви читаєте це, по своїй суті, не статична веб-сторінка, а інтерактивне середовище під назвою блокнот (notebook) Colab, який дає змогу писати й виконувати потрібний вам код.

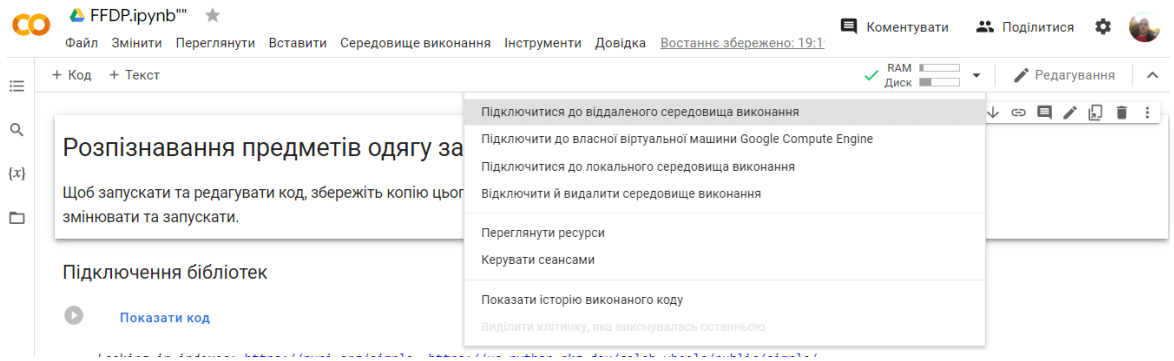


Рисунок 3.2 – Підключення до віддаленого середовища виконання

Блокноти Colab дозволяють поєднувати виконуваний код і форматований текст в один документ, а також додавати зображення, HTML, LaTeX тощо. Коли ви створюєте власні блокноти Colab, вони зберігаються у вашому обліковому записі Google Drive. Ви можете поділитися записниками Colab з колегами чи друзями, щоб вони могли коментувати та навіть редагувати.

За допомогою Colab користувачі можуть отримати доступ до всіх популярних бібліотек Python для аналізу та візуалізації даних. У наведених нижче комірках коду використовуються бібліотеки numpy (для створення довільних даних) і matplotlib (для візуалізації). Щоб змінити код, просто клацніть клітинку та почніть редагувати.

Ви можете імпортувати власні дані в блокноти Colab зі свого облікового запису Диска Google, включаючи електронні таблиці, а також GitHub і багато інших джерел.

У Colab ви можете імпортувати набір даних зображення, навчити на ньому класифікатор і оцінити модель за допомогою лише кількох рядків коду. Блокноти Colab запускають код на хмарних серверах Google, а це означає, що незалежно від того, наскільки потужним є ваш комп'ютер, ви можете скористатися перевагами апаратних можливостей Google, зокрема графічних і тензорних процесорів. Все, що потрібно, це веб-браузер. Colab широко використовується в машинному навчанні для таких завдань [17]:

- початок роботи з TensorFlow;
- розробка та навчання нейронних мереж;
- експериментувати з тензорними процесорами;
- поширення досліджень у сфері штучного інтелекту;
- створення навчальних посібників.

В Colab також може працювати з блокнотами:

- огляд спільних лабораторій,
- вказівки для розмітки,
- імпортуйте бібліотеки та встановлюйте залежності,
- зберігайте блокноти та завантажуйте їх на GitHub,
- інтерактивні таблиці,
- інтерактивні віджети, обробка даних,
- завантаження даних: диск, таблиці та Google Cloud Storage,
- діаграми: візуалізація даних,
- почати роботу з BigQuery використовувати апаратне прискорення,
- TensorFlow з GPU, TensorFlow з тензорними процесорам.

Оскільки штучні нейронні мережі використовують дуже багато операцій множення векторів та матриць, то у цій роботі стане у пригоді бібліотека

NumPy [18]. NumPy (скорочено від Numerical Python) — це бібліотека з відкритим кодом для мови Python. Вона має такі можливості:

- підтримка багатовимірних масивів (включаючи матриці);
- підтримка математичних функцій високого рівня, які призначені для роботи з багатовимірними масивами.

Для побудови різноманітних графіків нам знадобитьсч бібліотека Matplotlib [19].

Matplotlib - комплексна бібліотека для створення статичних, інтерактивних та анімованих візуалізацій на мові Python. Matplotlib має такі функціональні спроможності:

- Створення якісних сюжетів для публікацій.
- Створення інтерактивних фігур, котрі можна масштабувати, оновлювати, панорамувати.
- Налаштовування візуального макету і стилю.
- Експорт у велику кількість форматів файлів.
- Інтегрування у JupyterLab і у графічний користувацький інтерфейс.
- Використання широкого набору зовнішніх пакетів, побудованих на основі Matplotlib.

3.2 Опис набору даних зображень предметів одягу для навчання та тестування модуля

У цій роботі використовувався набір даних Fashion MNIST [20], який містить 70 000 зображень у градаціях сірого у 10 категоріях. Навчальний набір містить 60 000 прикладів та тестовий набір - 10 000 прикладів. Зображення показують окремі предмети одягу з низькою роздільною здатністю (28 на 28 пікселів), як показано на рисунку 3.3.

Кожному навчальному та тестовому прикладу присвоюється одна з міток, представлених у табл. 3.1

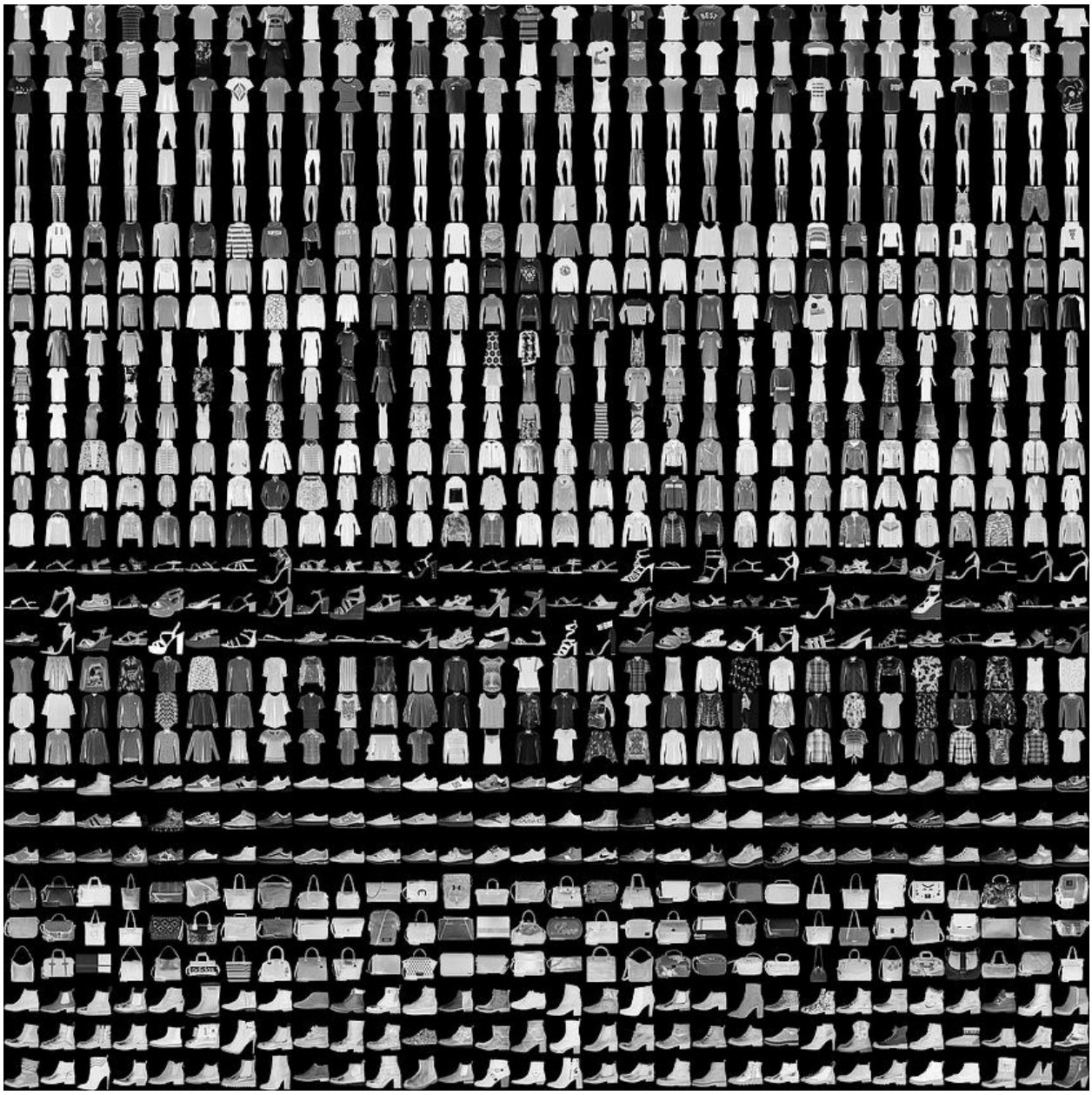


Рисунок 3.3 – Приклади зображень набору даних Fashion-MNIST

Все частіше Fashion-MNIST використовують як пряму заміну оригінального набору даних MNIST для бенчмаркінгу алгоритмів машинного навчання. Він має той самий розмір зображення та структуру розділень для навчання та тестування.

Оригінальний набір даних MNIST містить багато рукописних цифр. Члени спільноти штучного інтелекту/машинного навчання/науки про дані полюбляють цей набір даних і використовують його як еталон для перевірки своїх алгоритмів. Фактично, MNIST часто є першим, який випробовують

дослідники методів штучного інтелекту. «Якщо це не працює на MNIST, то взагалі не працюватиме», — кажуть вони. «Ну, якщо це спрацює на MNIST, то все одно може не спрацювати на інших».

Таблиця 3.1 – Мітки класів набору даних Fashion-MNIST.

Номер класу	**Назва класу**	**Переклад українською**
0	T-shirt/top	Футболка/топ
1	Trouser	Штани
2	Pullover	Светр
3	Dress	Сукня
4	Coat	Пальто
5	Sandal	Сандалі
6	Shirt	Сорочка
7	Sneaker	Кросівки
8	Bag	Сумка
9	Ankle boot	Чобітки

Ось кілька вагомих причин для заміни класичного MNIST на Fashion-MNIST:

- MNIST занадто простий. Згорткові мережі можуть досягти 99,7% на MNIST. Класичні алгоритми машинного навчання також можуть легко досягти 97%.
- MNIST надмірно використовується. Ще у 2017 році дослідник Google Brain та експерт з глибокого навчання Ян Гудфелло закликав відмовитися від використання MNIST.
- MNIST не може представляти сучасні завдання комп’ютерного зору. Так зазначав експерт з глибокого навчання/автор Keras Франсуа Шолле.

3.3 Програмна реалізація модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі

У цій роботі створено традиційну нейромережу для розпізнавання предметів одягу та спайкінгову нейромережу з метою порівняння продуктивності роботи обох нейромереж на цій класичній задачі. При створенні традиційної нейромережі використано можливості PyTorch зі створення та навчання стандартних неспайкінгових нейронних мереж. А при створенні спайкінгової нейромережі використано можливості PyTorchSpiking зі створення та навчання спайкінгових нейронних мереж, а також різні методи, які можна використовувати для точного налаштування продуктивності.

Першим кроком в програмному модулі розпізнавання предметів одягу на основі спайкінгової нейронної мережі є імпорт необхідних бібліотек:

```
import numpy as np
import keras_spiking
import tensorflow as tf
import matplotlib.pyplot as plt
```

Потім переходимо до завантаження набору даних Fashion MNIST:

```
train_images, train_labels = zip(
    *torchvision.datasets.FashionMNIST(".", train=True, download=True)
)
train_images = np.asarray([np.array(img) for img in train_images], dtype=np.float32)
train_labels = np.asarray(train_labels, dtype=np.int64)
test_images, test_labels = zip(
    *torchvision.datasets.FashionMNIST(".", train=False, download=True)
)
test_images = np.asarray([np.array(img) for img in train_images], dtype=np.float32)
test_labels = np.asarray(train_labels, dtype=np.int64)

# normalize images so values are between 0 and 1
train_images = train_images / 255.0
test_images = test_images / 255.0
```

```
class_names = [
    "T-shirt/top",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",
]
num_classes = len(class_names)

plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.axis("off")
    plt.title(class_names[train_labels[i]])
```

Далі ми створили та навчили просту неспайкінгову нейромережу для класифікації зображень Fashion MNIST:

```
model = torch.nn.Sequential(
    torch.nn.Linear(784, 128),
    torch.nn.ReLU(),
    torch.nn.Linear(128, 10),
)


def train(input_model, train_x, test_x):
    minibatch_size = 32
    optimizer = torch.optim.Adam(input_model.parameters())

    input_model.train()
    for j in range(10):
        train_acc = 0
        for i in range(train_x.shape[0] // minibatch_size):
            input_model.zero_grad()

            batch_in = train_x[i * minibatch_size : (i + 1) * minibatch_size]
            # flatten images
            batch_in = batch_in.reshape((-1,) + train_x.shape[1:-2] + (784,))
            batch_label = train_labels[i * minibatch_size : (i + 1) * minibatch_size]
            output = input_model(torch.tensor(batch_in))

            # compute sparse categorical cross entropy loss
            logp = torch.nn.functional.log_softmax(output, dim=-1)
            logpy = torch.gather(logp, 1, torch.tensor(batch_label).view(-1, 1))
            loss = -logpy.mean()

            loss.backward()
            optimizer.step()

            train_acc += torch.mean(
                torch.eq(torch.argmax(output, dim=1), torch.tensor(batch_label)).float()
            )
```

Як бачимо, це двохшаровий персептрон з 784 входами і 10 виходами, де перший шар містить 128 нейронів, а другий шар 10 нейронів, функція активації - ReLu, метод навчання ADAM, навчається 10 епох. Після виконання цього блоку коду буде виведено інформацію по точності процесу навчання:

```
Train accuracy (0): 0.819216668605804
Train accuracy (1): 0.862566649913787
Train accuracy (2): 0.875649988651275
Train accuracy (3): 0.88398331403732
Train accuracy (4): 0.891366660594940
Train accuracy (5): 0.896899998188018
Train accuracy (6): 0.900516688823701
Train accuracy (7): 0.905516684055328
Train accuracy (8): 0.908116638660430
Train accuracy (9): 0.911883354187011
Test accuracy: 0.906033337116241
```

Далі ми створили еквівалентну спайкінгову нейронну мережу. Тут є три важливі відмінності.

1. Додали часовий вимір до даних/моделі.

Спайкінгові мережі завжди функціонують у часі (тобто кожен прямий прохід через модель виконуватиметься протягом певної кількості часових кроків). Це означає, що нам потрібно додати часовий вимір до даних, тому замість форми (batch_size, ...) вони матимуть форму (batch_size, n_steps, ...). Принцип такий; спайкінговий нейрон приймає часові дані та обчислює з плином часу.

2. Замінили всі функції активації на pytorch_spiking.SpikingActivation.

pytorch_spiking.SpikingActivation може інкапсулювати будь-яку функцію активації та створюватиме еквівалентну спайкінгову реалізацію. Нейрони будуть видавати спайки зі швидкістю, пропорційною вихідному сигналу базової функції активації. Наприклад, якщо функція активації видає значення 10, то SpikingActivation видаватиме спайки з частотою 10 Гц (тобто 10 піків за 1 змодельовану секунду, де 1 змодельована секунда еквівалентна певній кількості часових кроків, що визначається параметром dt SpikingActivation).

3. Усереднили кількість вихідних імпульсів по часу.

Вихідні дані шару pytorch_spiking.SpikingActivation також є часовим рядом. Для класифікації нам потрібно якимось чином агрегувати цю часову інформацію, щоб створити остаточний прогноз. Усереднення вихідних даних з часом зазвичай є хорошим підходом (але не єдиним методом; ми також можемо, наприклад, подивитися на вихідні дані на останньому кроці часу або застосувати кодування «час до першого спайку»). Ми додаємо шар pytorch_spiking.TemporalAvgPool для усереднення за часовим виміром даних.

```
# repeat the images for n_steps
n_steps = 10
train_sequences = np.tile(train_images[:, None], (1, n_steps, 1, 1))
test_sequences = np.tile(test_images[:, None], (1, n_steps, 1, 1))
```

```
spiking_model = torch.nn.Sequential(
    torch.nn.Linear(784, 128),
    # wrap ReLU in SpikingActivation
    pytorch_spiking.SpikingActivation(torch.nn.ReLU(), spiking_aware_training=False),
    # use average pooling layer to average spiking output over time
    pytorch_spiking.TemporalAvgPool(),
    torch.nn.Linear(128, 10),
)

# train the model, identically to the non-spiking version,
# except using the time sequences as input
train(spiking_model, train_sequences, test_sequences)
```

Після запуску цього блоку коду буде виведено інформацію по точності процесу навчання:

```
Train accuracy (0): 0.819233357906341
Train accuracy (1): 0.862483322620391
Train accuracy (2): 0.874949991703033
Train accuracy (3): 0.883000016212463
Train accuracy (4): 0.888916671276092
Train accuracy (5): 0.894283354282379
Train accuracy (6): 0.899500012397766
Train accuracy (7): 0.9032333493232727
Train accuracy (8): 0.9077666401863098
Train accuracy (9): 0.9107499718666077
Test accuracy: 0.17543333768844604
```

Ми бачимо, що хоча точність навчання така ж висока, як ми очікували, точність тестування - ні. Це пов'язано з унікальною особливістю SpikingActivation; вона автоматично змінює поведінку спайкінгових нейронів під час навчання. Оскільки спайкінгові нейрони (загалом) не диференційовні, ми не можемо безпосередньо використовувати функцію активації спайків під час навчання. Натомість SpikingActivation використовуватиме базову (без спайків) активацію під час навчання та версію зі спайками під час логічного висновку. Отже, під час навчання вище ми бачимо точність неспайкінгової моделі, але під час тестування ми бачимо точність спайкінгової моделі.

Отже, питання полягає в тому, чому продуктивність спайкінгової моделі набагато гірша, ніж еквівалента без спайків, і що ми можемо зробити, щоб це виправити?

Щоб краще зрозуміти, що відбувається, було створено блок коду для візуалізації результатів роботи спайкінгової нейромережі:

```
def check_output(seq_model, modify_dt=None):  # noqa: C901
    """
    This code is only used for plotting purposes, and isn't necessary to
    understand the rest of this example; feel free to skip it
    if you just want to see the results.
    """

    # rebuild the model in a form that will let us access the output of
    # intermediate layers
    class Model(torch.nn.Module):
        def __init__(self):
            super().__init__()

            self.has_temporal_pooling = False
            for i, module in enumerate(seq_model.modules()):
                if isinstance(module, pytorch_spiking.TemporalAvgPool):
                    # remove the pooling so that we can see the model's output over time
                    self.has_temporal_pooling = True
                    continue

                if isinstance(
                    module, (pytorch_spiking.SpikingActivation, pytorch_spiking.Lowpass)
                ):
                    # update dt, if specified
                    if modify_dt is not None:
                        module.dt = modify_dt
                    # always return the full time series so we can visualize it
                    module.return_sequences = True

                if isinstance(module, pytorch_spiking.SpikingActivation):
                    # save this layer so we can access it later
                    self.spike_layer = module
```

```
    def forward(self, inputs):
        x = inputs

        for i, module in enumerate(self.modules()):
            if i > 0:
                x = module(x)

                if isinstance(module, pytorch_spiking.SpikingActivation):
                    # save this layer so we can access it later
                    spike_output = x

        return x, spike_output

func_model = Model()

# run model
func_model.eval()
with torch.no_grad():
    output, spikes = func_model(
        torch.tensor(
            test_sequences.reshape(
                test_sequences.shape[0], test_sequences.shape[1], -1
            )
        )
    )
output = output.numpy()
spikes = spikes.numpy()

if func_model.has_temporal_pooling:
    # check test accuracy using average output over all timesteps
    predictions = np.argmax(output.mean(axis=1), axis=-1)
else:
    # check test accuracy using output from last timestep
    predictions = np.argmax(output[:, -1], axis=-1)
accuracy = np.equal(predictions, test_labels).mean()
print(f"Test accuracy: {100 * accuracy:.2f}")
```

```
time = test_sequences.shape[1] * func_model.spike_layer.dt
n_spikes = spikes * func_model.spike_layer.dt
rates = np.sum(n_spikes, axis=1) / time

print(
    f"Spike rate per neuron (Hz): "
    f"min={np.min(rates):.2f} mean={np.mean(rates):.2f} max={np.max(rates):.2f}"
)

# plot output
for ii in range(4):
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 3, 1)
    plt.title(class_names[test_labels[ii]])
    plt.imshow(test_images[ii], cmap="gray")
    plt.axis("off")

    plt.subplot(1, 3, 2)
    plt.title("Spikes per neuron per timestep")
    bin_edges = np.arange(int(np.max(n_spikes[ii])) + 2) - 0.5
    plt.hist(np.ravel(n_spikes[ii]), bins=bin_edges)
    x_ticks = plt.xticks()[0]
    plt.xticks(
        x_ticks[(np.abs(x_ticks - np.round(x_ticks)) < 1e-8) & (x_ticks > -1e-8)]
    )
    plt.xlabel("# of spikes")
    plt.ylabel("Frequency")

    plt.subplot(1, 3, 3)
    plt.title("Output predictions")
    plt.plot(
        np.arange(test_sequences.shape[1]) * func_model.spike_layer.dt,
        torch.softmax(torch.tensor(output[ii]), dim=-1),
    )
    plt.legend(class_names, loc="upper left")
    plt.xlabel("Time (s)")
    plt.ylabel("Probability")
    plt.ylim([-0.05, 1.05])

    plt.tight_layout()
```

У розділі 4 п.4.1 описано процес тестування та візуалізації роботи саме спайкінгової моделі нейромережі і оптимізацію її параметрів для отримання кращих результатів точності розпізнавання предметів одягу на тестовій вибірці.

3.4 Висновок до розділу 3

У розділі було обґрунтовано вибір мови програмування Python, середовища програмування Colab та спеціалізованих бібліотек NumPy, Keras, TensorFlow та Matplotlib для програмної реалізації модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі. Проведено аналіз набору зображень Fashion MNIST, який містить 70 000 зображень у градаціях сірого у 10 категоріях, з яких 60 000 прикладів входять у навчальний набір, а 10 000 прикладів входять у тестовий набір. Зображення показують окремі предмети одягу з роздільною здатністю 28 на 28 пікселів. Описано основні етапи програмної реалізація та функціонування модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі, що складається з завантаження бібліотек, завантаження набору даних із зображеннями, запуску роботи моделі аналогу і запропонованої нейромережі, візуалізації результатів роботи, підрахунку показників достовірності аналога та розробленої нейромережі.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ ПРЕДМЕТІВ ОДЯГУ НА ОСНОВІ СПАЙКІНГОВОЇ НЕЙРОННОЇ МЕРЕЖІ

4.1 Тестування програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі

В процесі тестування проводилась візуалізація роботи спайкінгової моделі нейромережі і оптимізація її параметрів для отримання кращих результатів точності розпізнавання предметів одягу на тестовій вибірці.

В результаті навчання з параметрами, представленими в кінці п.3.3, ми отримали такі показники точності:

```
Test accuracy: 17.85
Spike rate per neuron (Hz): min=0.00 mean=0.61 max=100.00
```

Як бачимо, достовірність на тестовій вибірці складає всього 17, 85%. Це дуже низький показник. Для з’ясування причин такого поганого результату розглянемо деякі приклади візуалізації розпізнавання предметів одягу, які представлено на рисунку 4.1.

Із рисунка 4.1 видно безпосередню проблему цього явища: нейрони майже не мають піків. Так, на рисунку.4.1 видно по показнику «Probability» скільки генерує кожен із 10 нейронів спайків. Якщо «Probability» 0,1, значить генерується 1 спайк, а якщо «Probability» 1, то генерується 10 спайків. Так на рисунку 4.1а видно, що більшість нейронів генерують за час моделювання 0 або 1 спайк і тільки в момент часу 0.002 с генерується 10 спайків для класу «bag» (що, до речі є помилковим). А на рис. 4.1б видно, що 10 спайків генерується в 4 моменти часу: для класу «trouser» в момент часу 0.002 с, для класу «dress» в момент часу 0.003 с, для класу «ankle boot» в момент часу 0.005 с та для класу «sandal» в момент часу 0.009 с. Середня кількість спайків, які

ми отримуємо від кожного нейрона в нашому шарі SpikingActivation, дуже близька до нуля (`mean=0.61`), і в результаті вихідний сигнал здебільшого стабільнио нульовий без спайків.

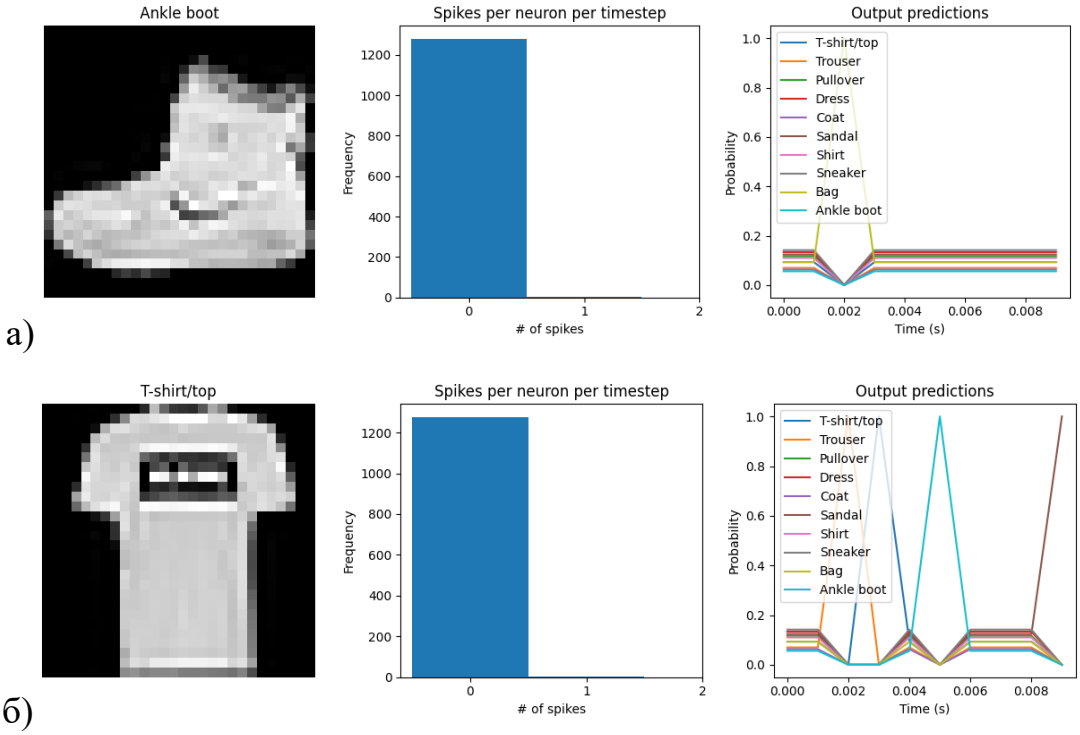


Рисунок 4.1 – Деякі приклади візуалізації розпізнавання предметів одягу при dt=0,001

Щоб зрозуміти, чому так відбувається, потрібно більше подумати про часову природу спайкінгових нейронів. Шар налаштовано таким чином, що коли базова функція активації видає значення 1, це еквівалентно видачі спайку з частотою 1 Гц (тобто один спайк за секунду). У наведеному на рисунку 4.1 прикладі, ми моделюємо протягом 10 часових кроків зі значенням dt за замовчуванням 0,001 с, тому моделювання загалом триває 0,01 с, а спайк може з’явитись протягом 1 с. Якщо нейрони не видають спайків одразу після початку моделювання дуже швидко, і ми моделюємо лише протягом 0,01 с, то не дивно, що за цей час ми не отримуємо жодних спайків у цьому часовому вікні.

Ми можемо збільшити значення dt, фактично довше запускаючи спайкові нейрони, щоб отримати точніше вимірювання вихідного сигналу нейрона. По суті, це дозволяє нам отримувати більше спайків з кожного нейрона, даючи кращу оцінку фактичної частоти спайкінгового нейрона. Ми можемо побачити, як змінюється кількість спайків та точність зі збільшенням dt, запустивши код:

```
# dt=0.01 * 10 timesteps is equivalent to 0.1s of simulated time
check_output(spiking_model, modify_dt=0.01)
```

І побачимо результат виконання цього коду:

```
Test accuracy: 64.42
Spike rate per neuron (Hz): min=0.00 mean=0.60 max=30.00
```

Тобто достовірність на тестовій вибірці збільшилась до 64,42% (була 17,85%). Це суттєве зростання. І дійсно, з рисунку 4.2 видно, що кількість генерованих імпульсів суттєво зросла.

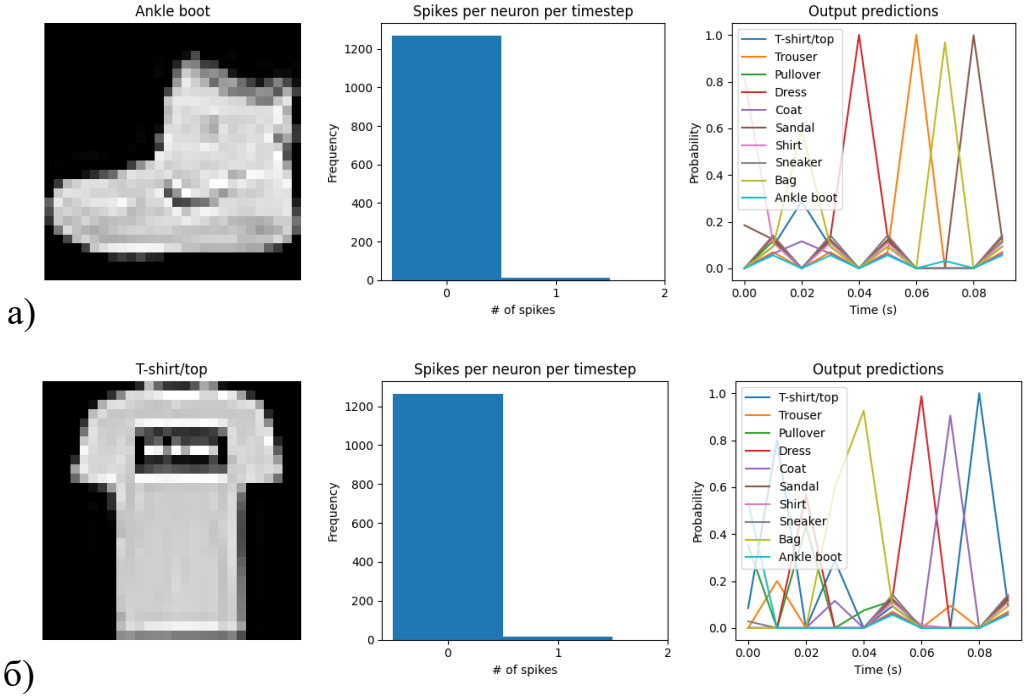


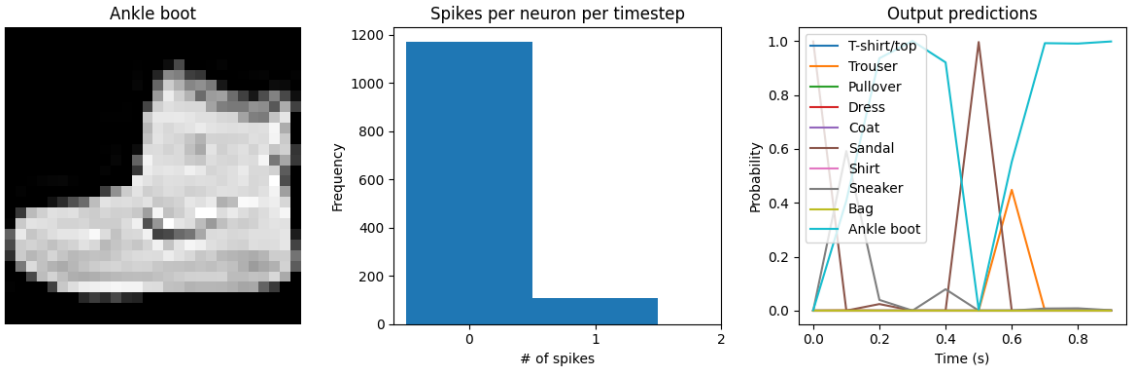
Рисунок 4.2 – Деякі приклади візуалізації розпізнавання предметів одягу при dt=0,01

Збільшимо ще значення dt до 0,1:

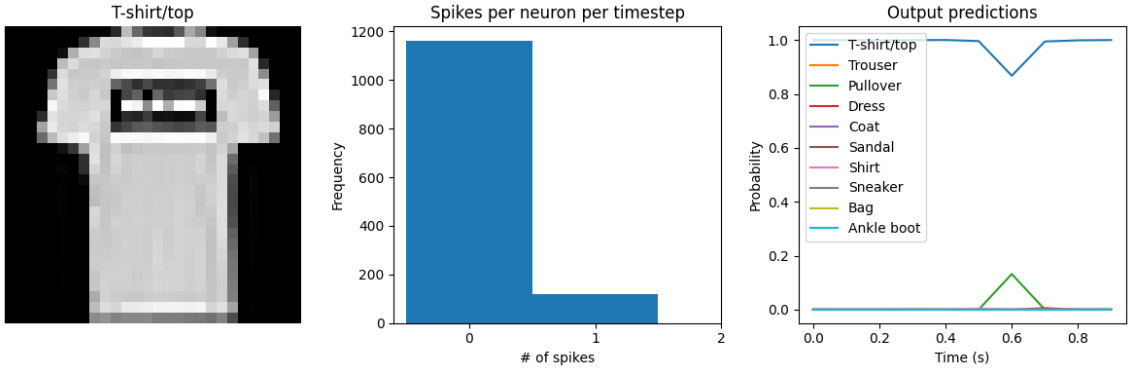
```
check_output(spiking_model, modify_dt=0.1)
```

І отримаємо такий результат (рис.4.3):

```
Test accuracy: 90.32
Spike rate per neuron (Hz): min=0.00 mean=0.61 max=24.00
```



а)



б)

Рисунок 4.3 – Деякі приклади візуалізації розпізнавання предметів одягу при dt=0,1

Якщо збільшити ще значення dt до 1:

```
check_output(spiking_model, modify_dt=1)
```

То отримаємо такий результат (рис.4.4):

```
Test accuracy: 92.68
Spike rate per neuron (Hz): min=0.00 mean=0.61 max=24.50
```

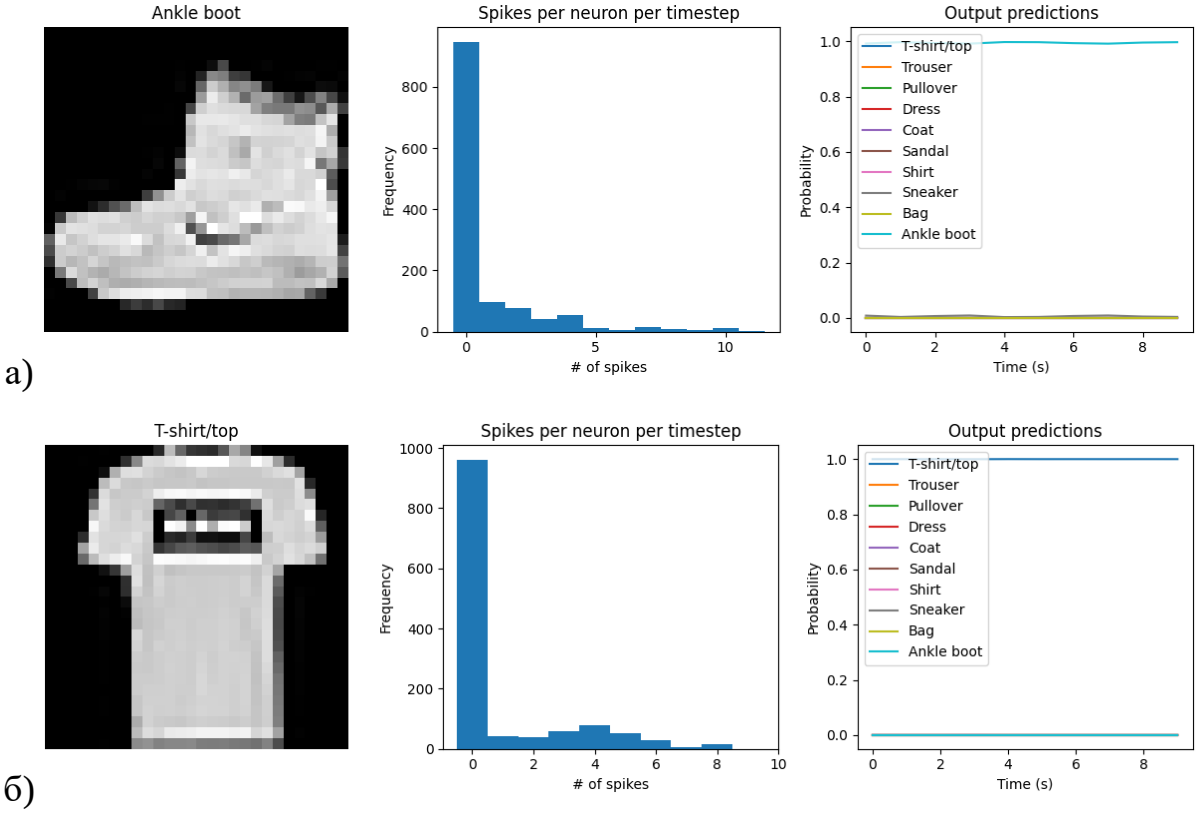


Рисунок 4.4 – Деякі приклади візуалізації розпізнавання предметів одягу при dt=1

Ми дослідили, що зі збільшенням dt достовірність спайкінгової нейромережі також збільшується. Крім того, зі збільшенням dt збільшується середня кількість спайків на один нейрон. Щоб пояснити, чому це покращує достовірність, пам'ятаємо, що хоча час моделювання збільшується, фактична кількість часових кроків все ще становить 10 у всіх випадках. Ми фактично об'єднуємо всі спайки, які виникають на кожному часовому кроці. Тож, зі збільшенням розмірів наших інтервалів (збільшенням dt), кількість спайків буде точніше наближатися до «справжнього» виходу базової функції активації у випадку неспайкінгової нейромережі.

Може здатися,ливо, що треба просто збільшити dt до дуже великого значення і таким чином отримаємо чудову продуктивність. Але коли ми це зробимо, то втратимо переваги спайкінгових нейромереж, які мотивували їх

досліджувати. Тобто ми втратимо у швидкодії намагаючись збільшити достовірність.

Таким чином, встановлення dt являє собою компроміс між точністю та швидкодією. Вибір відповідного значення залежатиме від вимог до розроблюваної програми.

4.2 Аналіз результатів роботи програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі

Як метрику оцінки ми будемо використовувати достовірність. Достовірність – це відношення правильно класифікованих екземплярів до їхньої загальної кількості.

Як зазначалось у п.1.3, аналогом було обрано повнозв’язну нейронну мережу з двома шарами – 128 нейронів та 10 нейронів відповідно у першому та другому шарах. Порівняння параметрів аналога та розробленого програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі наведено у таблиці 4.1.

Таблиця 4.1 - Порівняння параметрів аналога та розробленого програмного модуля розпізнавання предметів одягу

	Нейронів у першому шарі	Нейронів у другому шарі	Вид нейрона	Достовірність розпізнавання, %
Аналог	128	10	аналоговий	90,60
Розроблений модуль	128	10	спайкінговий	92,68

Із табл. 4.1 видно, що достовірність розпізнавання на тестовому наборі предметів одягу для розробленого модуля має величину 92,68%, а для аналога

– 90,60%. Тобто достовірність розпізнавання розробленої програми на 2,08% вища, ніж у програми аналогу.

Таким чином, можна зробити висновок, що розроблений програмного модуль розпізнавання предметів одягу на основі спайкінгової нейронної мережі має порівняно з аналогом збільшену на 2,08% т достовірність розпізнавання предметів одягу. Тобто мета роботи досягнута – достовірність розпізнавання предметів одягу підвищена.

4.3 Висновок до розділу 4

У розділі в результаті тестування програмного модуля було доведено його працездатність та відповідність поставленому завданню. Було проведено оптимізацію спайкінгової нейронної мережі по параметру часового кроку dt з метою отримання кращої достовірності розпізнавання. Розроблений програмний модуль розпізнавання предметів одягу на основі спайкінгової нейронної мережі має достовірність розпізнавання на тестовому наборі 92,68%, а аналог – 90,60%. Тобто достовірність розпізнавання розробленої програми на 2,08% вища, ніж у програми аналогу.

ВИСНОВКИ

У роботі було розглянуто задачу розпізнавання предметів одягу як окремий випадок загальної задачі розпізнавання образів. Розроблений програмний модуль призначений для розпізнавання предметів одягу як класичної задачі машинного навчання на наборі даних Fashion-MNIST. В ході аналізу предметної області було описано детальну постановку задачі розпізнавання предметів одягу. Було розглянуто такі методи розпізнавання образів: статистичні, синтаксичні, нейромережеві, зіставлення шаблонів та нечіткої логіки. Як найбільш перспективний і застосовний до даної задачі було обрано нейромережевий метод. Крім цього, було обгрунтовано вибір аналогу до розробленого модуля розпізнавання предметів одягу, який реалізовано на багатошаровому персептроні

Було проведено аналіз роботи спайкінгової нейронної мережі для розпізнавання предметів одягу. Розглянуто подібність спайкінгових нейромереж до біологічних нейромереж, розглянуто моделі спайкінгових нейронів та методи навчання спайкінгових нейромереж. Розроблено структуру спайкінгової нейронної мережі для розпізнавання предметів одягу, яка еквівалентна традиційній нейромережі, тобто містить два повнозв’язних шари спайкінгових нейронів у кількості відповідно 128 і 10, має 784 входи та 10 виходів. Також було розроблено алгоритм роботи програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі.

Було обґрунтовано вибір мови програмування Python, середовища програмування Colab та спеціалізованих бібліотек NumPy, Keras, TensorFlow та Matplotlib для програмної реалізації модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі. Проведено аналіз набору зображень Fashion MNIST, який містить 70 000 зображень у градаціях сірого у 10 категоріях, з яких 60 000 прикладів входять у навчальний набір, а 10 000 прикладів входять у тестовий набір. Зображення показують окремі предмети

одягу з роздільною здатністю 28 на 28 пікселів. Описано основні етапи програмної реалізація та функціонування модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі, що складається з завантаження бібліотек, завантаження набору даних із зображеннями, запуску роботи моделі аналогу і запропонованої нейромережі, візуалізації результатів роботи, підрахунку показників достовірності аналога та розробленої нейромережі.

У результаті тестування програмного модуля було доведено його працездатність та відповідність поставленому завданню. Було проведено оптимізацію спайкінгової нейронної мережі по параметру часового кроку dt з метою отримання кращої достовірності розпізнавання. Розроблений програмний модуль розпізнавання предметів одягу на основі спайкінгової нейронної мережі має достовірність розпізнавання на тестовому наборі 92,68%, а аналог – 90,60%. Тобто достовірність розпізнавання розробленої програми на 2,08% вища, ніж у програми аналогу

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Куссуль Н. М. Інтелектуальні обчислення: навчальний посібник (із грифом МОН України) / Н. М. Куссуль, А. Ю. Шелестов, А. Н. Лавренюк. - К.: «Наукова думка», 2006. — 186 с. ISBN 966-00-0592-X.

2 Персептрон [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/316106483_Multilayer_perceptron_as_the_primary_instrument_for_image_clusterin

3 Кутковецький В. Я. Розпізнавання образів : навчальний посібник / В. Я. Кутковецький. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2017. – 420 с. ISBN 978-966-336-384-4

4 Руденко О.В. Штучні нейронні мережі: Навчальний посібник / О.В.Руденко, Є.В.Бодянський. - Харків : ТОВ «Компанія СМІТ», 2006. — 404 с. - ISBN 966-8630-73-X.

5 Любунь З. М. Основи теорії нейромереж: текст лекцій / З. М. Любунь. – Львів : Видавничий центр ЛНУ ім. Івана Франка, 2006.

6 Аналог [Електронний ресурс]. Режим доступу: GitHub - sudhir2016/Google-Colab-1: Fashion_mnist.

7 Гудфеллоу Я. Глибоке навчання / пер. з анг. А. А. Слинкіна. 2-е вид., випр. К.: Наукова Думка, 2018. 652 с.

8 Maas, Wolfgang. Networks of spiking neurons: The third generation of neural network models (англ.) // Neural Networks : journal. — 1997. — Vol. 10. — P. 1659—1671. — doi:10.1016/S0893-6080(97)00011-7

9 Колесницький О. К. Аналітичний огляд апаратних реалізацій спайкових нейронних мереж / О. К. Колесницький // Математичні машини і системи. – 2015. – №1, С.3-19. ISSN 1028-9763 [Електронний ресурс]. Режим доступу - http://www.immsp.kiev.ua/publications/articles/2015/2015_1/01_2015_Kolesnytskyy.pdf

10 Gerstner, Wulfram and Kistler, Werner M. Spiking Neuron Models: Single Neurons, Populations, Plasticity. — Cambridge, U.K.: Cambridge university press, 2002

11 V. P. Kozemiako ; O. K. Kolesnytskyj ; T. S. Lischenko ; W. Wojcik and A. Sulemenov " Optoelectronic spiking neural network ", Proc. SPIE 8698, Optical Fibers and Their Applications 2012, 86980M (January 11, 2013); doi:10.1117/12.2019340; http://dx.doi.org/10.1117/12.2019340

12 The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3) [Online]. Available: https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

13 Sander M. Bohte and Joost N. Kok and Han La Poutre,SpikeProp: backpropagation for networks of spiking neurons, The European Symposium on Artificial Neural Networks}, 2000, url: https://api.semanticscholar.org/CorpusID:14069916

14 Neurocomputer architecture based on spiking neural network and its optoelectronic implementation / Oleh K. Kolesnytskyj; Vladislav V. Kutsman; Krzysztof Skorupski; Mukaddas Arshidinova, Proc. SPIE 11176, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019, 1117609 (6 November 2019); doi: 10.1117/12.2536607

15 Ponulak, Filip; Kasinski, Andrzej. Introduction to spiking neural networks: Information processing, learning and applications (англ.) // Acta neurobiologiae experimentalis[англ.] : journal. — 2010. — Vol. 71. — P. 409—433.

16 Python [Електронний ресурс]. Режим доступу:Newest 'python' Questions - Stack Overflow.

17 Google colab [Електронний ресурс]. Режим доступу:Вітаємо в Colaboratory - Colaboratory (google.com).

18 NumPy [Електронний ресурс] – Режим доступу: https://numpy.org/

19 Matplotlib: Visualization with Python [Електронний ресурс] – Режим доступу: https://matplotlib.org/

20 Fashion-MNIST [Електронний ресурс]. Режим доступу: https://github.com/zalandoresearch/fashion-mnist

21 Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності 122 - «Комп’ютерні науки» / Укладачі А.А.Яровий, О.В.Сілагін, І.В.Богач. – Вінниця: ВНТУ, 2022. – 47 с.

Додаток А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль розпізнавання предметів одягу на основі спайкінгової нейромережі

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 10,28%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Паночишин Ю.М.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Чабан Я. К.
(підпис) (прізвище, ініціали)

Додаток Б (обов'язковий)

Лістинг програми

```
# pylint: disable=redefined-outer-name

import matplotlib.pyplot as plt
import numpy as np
import torch
import torchvision

import pytorch_spiking

torch.manual_seed(0)
np.random.seed(0)

train_images, train_labels = zip(
    *torchvision.datasets.FashionMNIST(".", train=True, download=True)
)
train_images = np.asarray([np.array(img) for img in train_images],
dtype=np.float32)
train_labels = np.asarray(train_labels, dtype=np.int64)
test_images, test_labels = zip(
    *torchvision.datasets.FashionMNIST(".", train=False, download=True)
)
test_images = np.asarray([np.array(img) for img in train_images],
dtype=np.float32)
test_labels = np.asarray(train_labels, dtype=np.int64)

# normalize images so values are between 0 and 1
train_images = train_images / 255.0
test_images = test_images / 255.0

class_names = [
    "T-shirt/top",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",
]
num_classes = len(class_names)

plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.axis("off")
```

```
    plt.title(class_names[train_labels[i]])

model = torch.nn.Sequential(
    torch.nn.Linear(784, 128),
    torch.nn.ReLU(),
    torch.nn.Linear(128, 10),
)


def train(input_model, train_x, test_x):
    minibatch_size = 32
    optimizer = torch.optim.Adam(input_model.parameters())

    input_model.train()
    for j in range(10):
        train_acc = 0
        for i in range(train_x.shape[0] // minibatch_size):
            input_model.zero_grad()

            batch_in = train_x[i * minibatch_size : (i + 1) * minibatch_size]
            # flatten images
            batch_in = batch_in.reshape((-1,) + train_x.shape[1:-2] + (784,))
            batch_label = train_labels[i * minibatch_size : (i + 1) *
minibatch_size]
            output = input_model(torch.tensor(batch_in))

            # compute sparse categorical cross entropy loss
            logp = torch.nn.functional.log_softmax(output, dim=-1)
            logpy = torch.gather(logp, 1, torch.tensor(batch_label).view(-1, 1))
            loss = -logpy.mean()

            loss.backward()
            optimizer.step()

            train_acc += torch.mean(
                torch.eq(torch.argmax(output, dim=1),
torch.tensor(batch_label)).float()
            )

        train_acc /= i + 1
        print(f"Train accuracy ({j}): {train_acc.numpy()}")

    # compute test accuracy
    input_model.eval()
    test_acc = 0
    for i in range(test_x.shape[0] // minibatch_size):
        batch_in = test_x[i * minibatch_size : (i + 1) * minibatch_size]
        batch_in = batch_in.reshape((-1,) + test_x.shape[1:-2] + (784,))
        batch_label = test_labels[i * minibatch_size : (i + 1) * minibatch_size]
        output = input_model(torch.tensor(batch_in))

        test_acc += torch.mean(
            torch.eq(torch.argmax(output, dim=1),
torch.tensor(batch_label)).float()
```

```
        )

    test_acc /= i + 1

    print(f"Test accuracy: {test_acc.numpy()}")


train(model, train_images, test_images)

# repeat the images for n_steps
n_steps = 10
train_sequences = np.tile(train_images[:, None], (1, n_steps, 1, 1))
test_sequences = np.tile(test_images[:, None], (1, n_steps, 1, 1))

spiking_model = torch.nn.Sequential(
    torch.nn.Linear(784, 128),
    # wrap ReLU in SpikingActivation
    pytorch_spiking.SpikingActivation(torch.nn.ReLU(),
spiking_aware_training=False),
    # use average pooling layer to average spiking output over time
    pytorch_spiking.TemporalAvgPool(),
    torch.nn.Linear(128, 10),
)

# train the model, identically to the non-spiking version,
# except using the time sequences as input
train(spiking_model, train_sequences, test_sequences)

def check_output(seq_model, modify_dt=None):  # noqa: C901
    """
    This code is only used for plotting purposes, and isn't necessary to
    understand the rest of this example; feel free to skip it
    if you just want to see the results.
    """

    # rebuild the model in a form that will let us access the output of
    # intermediate layers
    class Model(torch.nn.Module):
        def __init__(self):
            super().__init__()

            self.has_temporal_pooling = False
            for i, module in enumerate(seq_model.modules()):
                if isinstance(module, pytorch_spiking.TemporalAvgPool):
                    # remove the pooling so that we can see the model's output
over time
                    self.has_temporal_pooling = True
                    continue

                if isinstance(
                    module, (pytorch_spiking.SpikingActivation,
pytorch_spiking.Lowpass)
                ):
                    # update dt, if specified
                    if modify_dt is not None:
```

```
                    module.dt = modify_dt
                # always return the full time series so we can visualize it
                module.return_sequences = True

            if isinstance(module, pytorch_spiking.SpikingActivation):
                # save this layer so we can access it later
                self.spike_layer = module

            if i > 0:
                self.add_module(str(i), module)

    def forward(self, inputs):
        x = inputs

        for i, module in enumerate(self.modules()):
            if i > 0:
                x = module(x)

                if isinstance(module, pytorch_spiking.SpikingActivation):
                    # save this layer so we can access it later
                    spike_output = x

        return x, spike_output

func_model = Model()

# run model
func_model.eval()
with torch.no_grad():
    output, spikes = func_model(
        torch.tensor(
            test_sequences.reshape(
                test_sequences.shape[0], test_sequences.shape[1], -1
            )
        )
    )
output = output.numpy()
spikes = spikes.numpy()

if func_model.has_temporal_pooling:
    # check test accuracy using average output over all timesteps
    predictions = np.argmax(output.mean(axis=1), axis=-1)
else:
    # check test accuracy using output from last timestep
    predictions = np.argmax(output[:, -1], axis=-1)
accuracy = np.equal(predictions, test_labels).mean()
print(f"Test accuracy: {100 * accuracy:.2f}")

time = test_sequences.shape[1] * func_model.spike_layer.dt
n_spikes = spikes * func_model.spike_layer.dt
rates = np.sum(n_spikes, axis=1) / time

print(
    f"Spike rate per neuron (Hz): "
```

```
        f"min={np.min(rates):.2f} mean={np.mean(rates):.2f}
max={np.max(rates):.2f}"
    )

    # plot output
    for ii in range(4):
        plt.figure(figsize=(12, 4))

        plt.subplot(1, 3, 1)
        plt.title(class_names[test_labels[ii]])
        plt.imshow(test_images[ii], cmap="gray")
        plt.axis("off")

        plt.subplot(1, 3, 2)
        plt.title("Spikes per neuron per timestep")
        bin_edges = np.arange(int(np.max(n_spikes[ii])) + 2) - 0.5
        plt.hist(np.ravel(n_spikes[ii]), bins=bin_edges)
        x_ticks = plt.xticks()[0]
        plt.xticks(
            x_ticks[(np.abs(x_ticks - np.round(x_ticks)) < 1e-8) & (x_ticks > -
1e-8)]
        )
        plt.xlabel("# of spikes")
        plt.ylabel("Frequency")

        plt.subplot(1, 3, 3)
        plt.title("Output predictions")
        plt.plot(
            np.arange(test_sequences.shape[1]) * func_model.spike_layer.dt,
            torch.softmax(torch.tensor(output[ii]), dim=-1),
        )
        plt.legend(class_names, loc="upper left")
        plt.xlabel("Time (s)")
        plt.ylabel("Probability")
        plt.ylim([-0.05, 1.05])

        plt.tight_layout()

check_output(spiking_model)

# dt=0.01 * 10 timesteps is equivalent to 0.1s of simulated time
check_output(spiking_model, modify_dt=0.01)

check_output(spiking_model, modify_dt=0.1)

check_output(spiking_model, modify_dt=1)

spikeaware_model = torch.nn.Sequential(
    torch.nn.Linear(784, 128),
    # set spiking_aware_training and a moderate dt
    pytorch_spiking.SpikingActivation(
        torch.nn.ReLU(), dt=0.01, spiking_aware_training=True
    ),
    pytorch_spiking.TemporalAvgPool(),
    torch.nn.Linear(128, 10),
)
```

```
train(spikeaware_model, train_sequences, test_sequences)

spikeaware_model = torch.nn.Sequential(
    torch.nn.Linear(784, 128),
    # set spiking_aware_training and a moderate dt
    pytorch_spiking.SpikingActivation(
        torch.nn.ReLU(), dt=0.01, spiking_aware_training=True
    ),
    pytorch_spiking.TemporalAvgPool(),
    torch.nn.Linear(128, 10),
)

train(spikeaware_model, train_sequences, test_sequences)

check_output(spikeaware_model)
# construct model using a generic Module so that we can
# access the spiking activations in our loss function
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()

        self.dense0 = torch.nn.Linear(784, 128)
        self.spiking_activation = pytorch_spiking.SpikingActivation(
            torch.nn.ReLU(), dt=0.01, spiking_aware_training=True
        )
        self.temporal_pooling = pytorch_spiking.TemporalAvgPool()
        self.dense1 = torch.nn.Linear(128, 10)

    def forward(self, inputs):
        x = self.dense0(inputs)
        spikes = self.spiking_activation(x)
        spike_rates = self.temporal_pooling(spikes)
        output = self.dense1(spike_rates)

        return output, spike_rates


regularized_model = Model()

minibatch_size = 32
optimizer = torch.optim.Adam(regularized_model.parameters())

regularized_model.train()
for j in range(10):
    train_acc = 0
    for i in range(train_sequences.shape[0] // minibatch_size):
        regularized_model.zero_grad()

        batch_in = train_sequences[i * minibatch_size : (i + 1) * minibatch_size]
        batch_in = batch_in.reshape((-1,) + train_sequences.shape[1:-2] + (784,))
        batch_label = train_labels[i * minibatch_size : (i + 1) * minibatch_size]
        output, spike_rates = regularized_model(torch.tensor(batch_in))

        # compute sparse categorical cross entropy loss
```

```
        logp = torch.nn.functional.log_softmax(output, dim=-1)
        logpy = torch.gather(logp, 1, torch.tensor(batch_label).view(-1, 1))
        loss = -logpy.mean()

        # add activity regularization
        reg_weight = 1e-3  # weight on regularization penalty
        target_rate = 20  # target spike rate (in Hz)
        loss += reg_weight * torch.mean(
            torch.sum((spike_rates - target_rate) ** 2, dim=-1)
        )

        loss.backward()
        optimizer.step()

        train_acc += torch.mean(
            torch.eq(torch.argmax(output, dim=1),
torch.tensor(batch_label)).float()
        )

    train_acc /= i + 1
    print(f"Train accuracy ({j}): {train_acc.numpy()}")

check_output(regularized_model)

dt = 0.01

filtered_model = torch.nn.Sequential(
    torch.nn.Linear(784, 128),
    pytorch_spiking.SpikingActivation(
        torch.nn.ReLU(), spiking_aware_training=True, dt=dt
    ),
    # add a lowpass filter on output of spiking layer
    # note: the lowpass dt doesn't necessarily need to be the same as the
    # SpikingActivation dt, but it's probably a good idea to keep them in sync
    # so that if we change dt the relative effect of the lowpass filter is
unchanged
    pytorch_spiking.Lowpass(units=128, tau=0.1, dt=dt, return_sequences=False),
    torch.nn.Linear(128, 10),
)

train(filtered_model, train_sequences, test_sequences)

check_output(filtered_model)
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ ПРЕДМЕТІВ ОДЯГУ НА ОСНОВІ СПАЙКІНГОВОЇ НЕЙРОМЕРЕЖІ»

Виконала: студентка 4 курсу,групи 4-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Чабан Я. К.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф.КН
Паночишин Ю.М.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

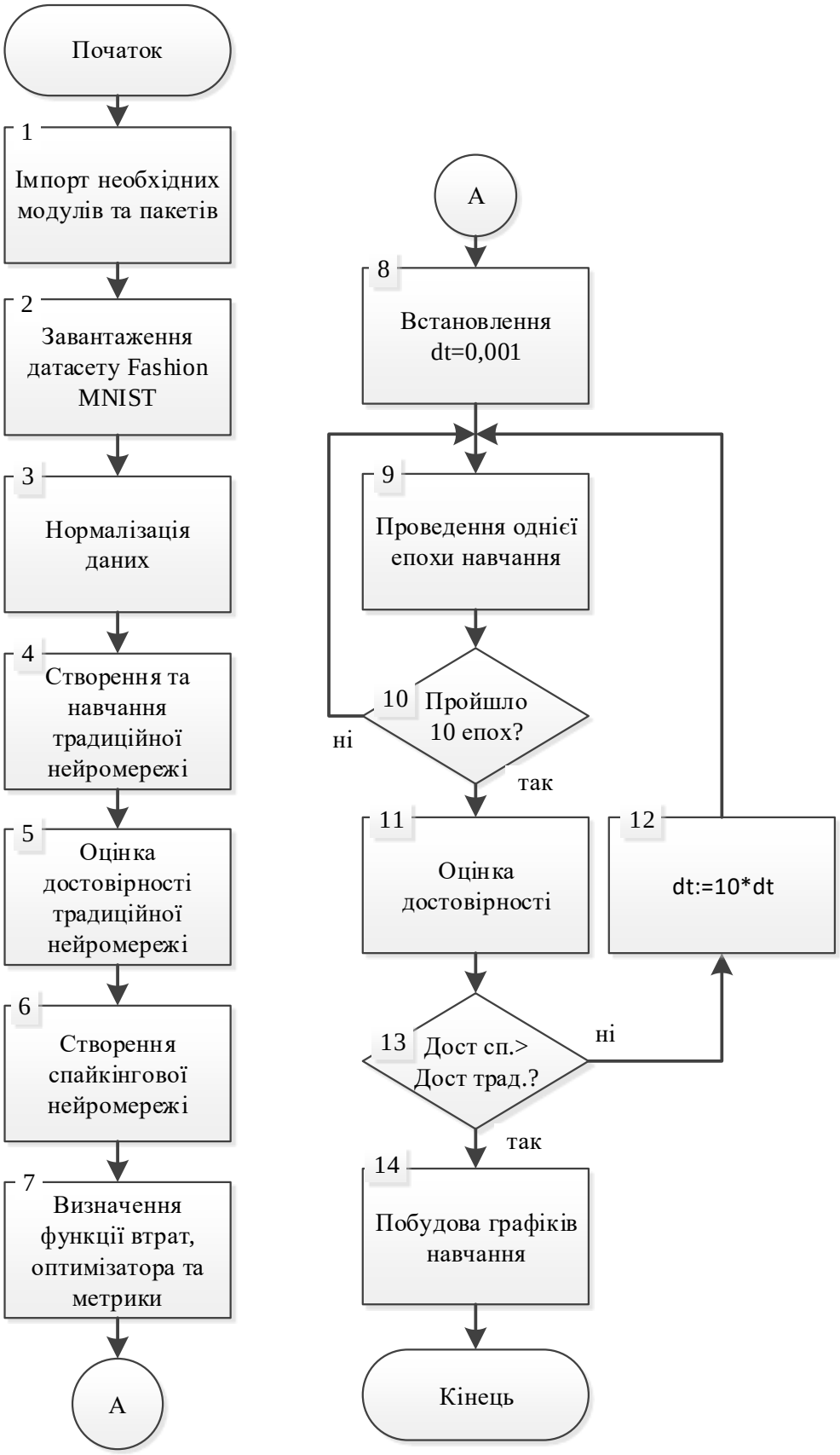


Рисунок В.1– Схема алгоритму роботи програмного модуля розпізнавання предметів одягу на основі спайкінгової нейронної мережі

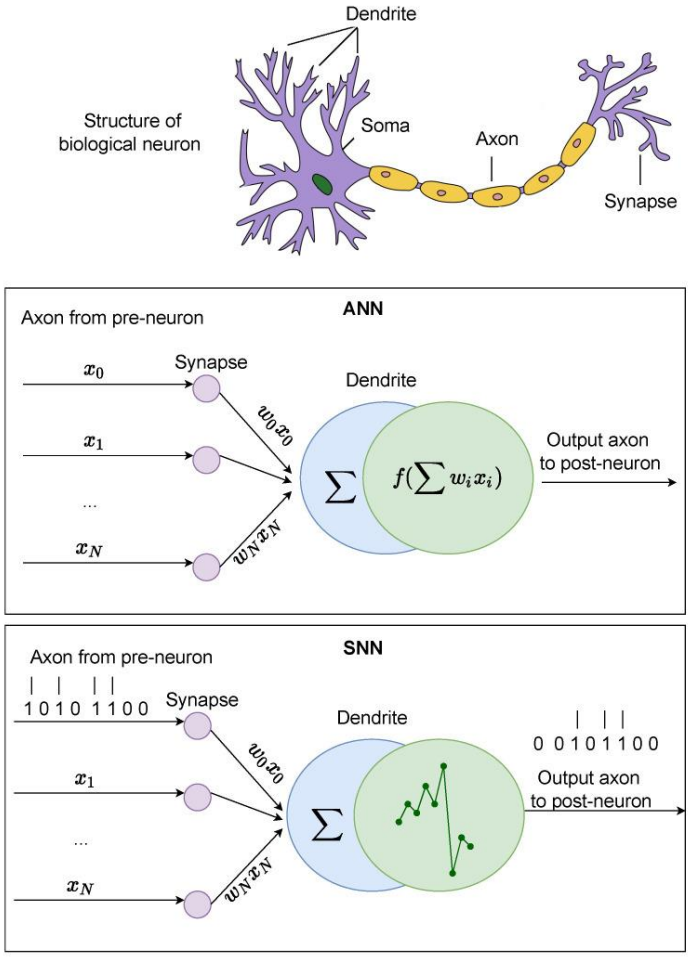


Рисунок В.2– Порівняння біологічного нейрона, штучного нейрона та спайкінгового нейрона

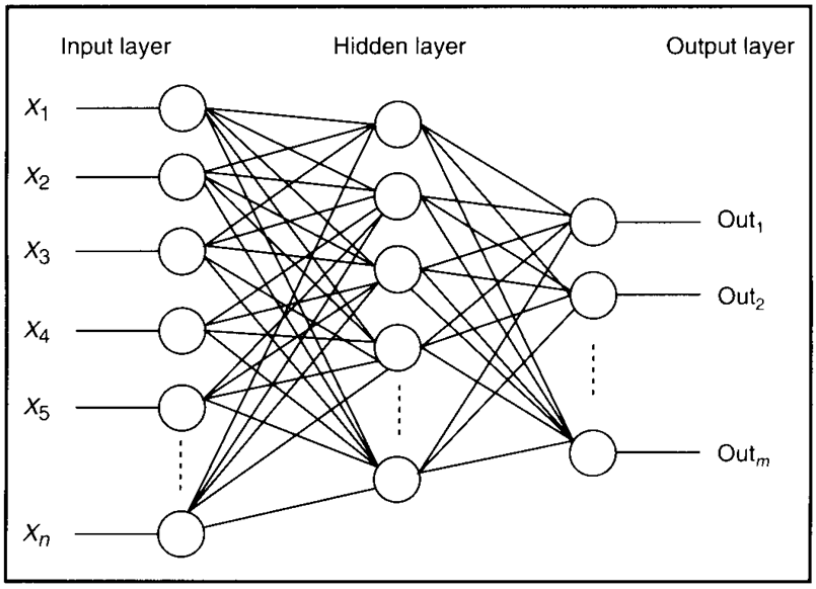


Рисунок В.3 – Структура спайкінгової нейронної мережі

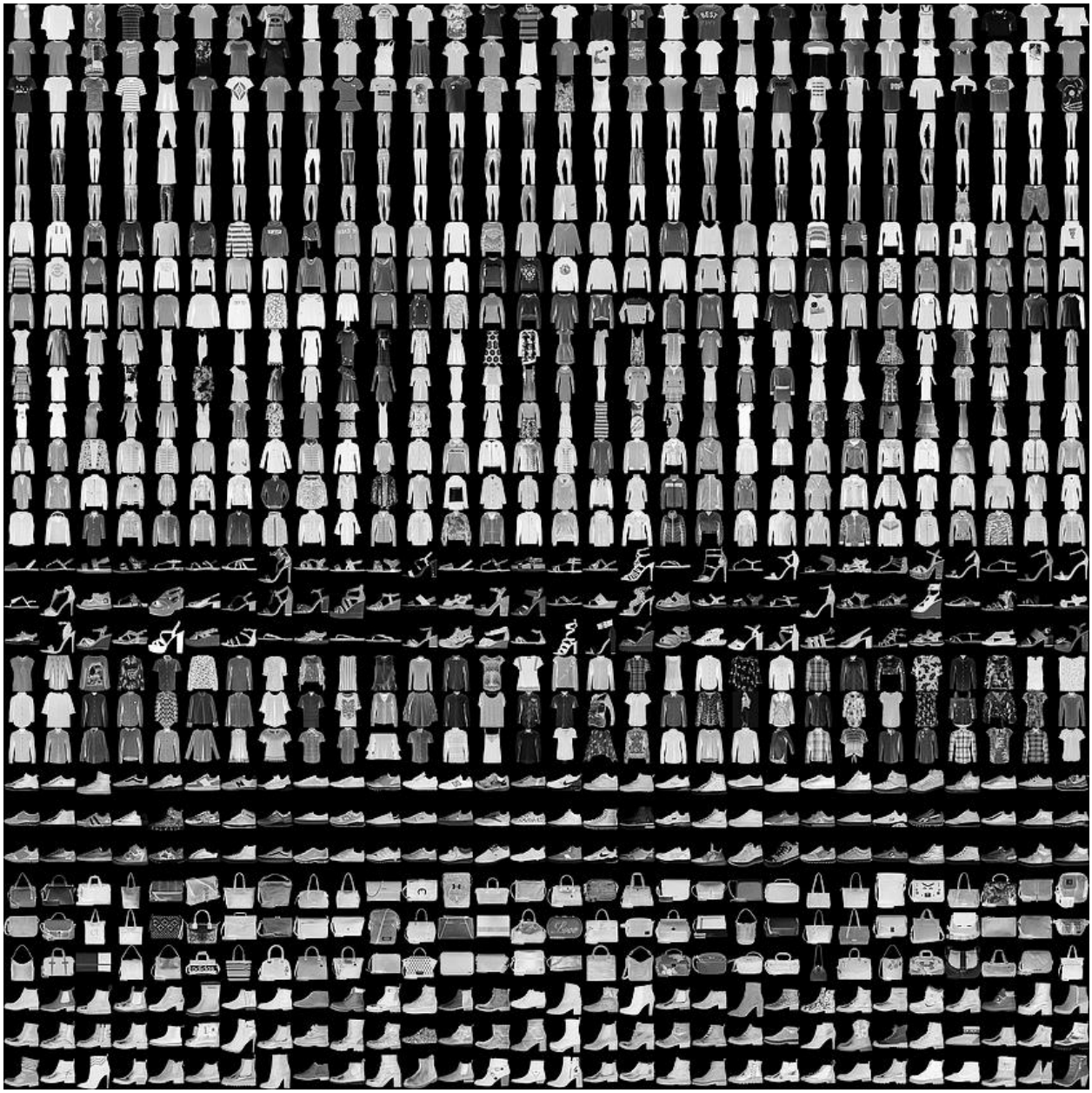
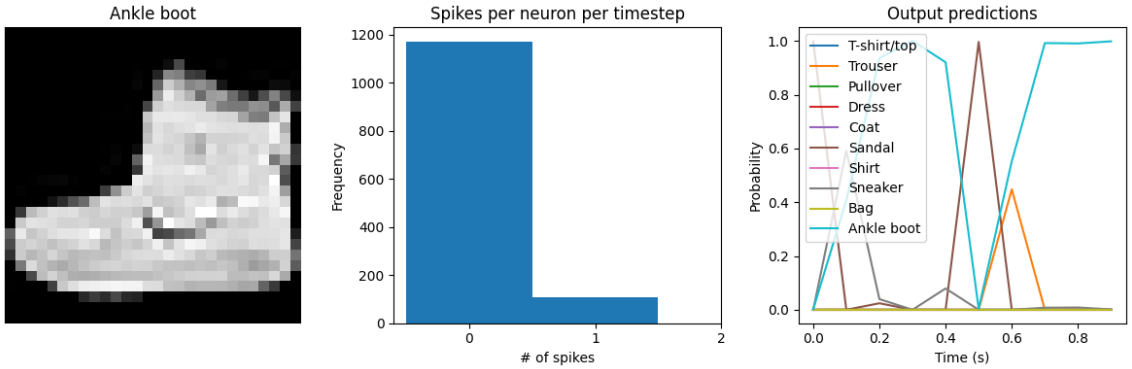
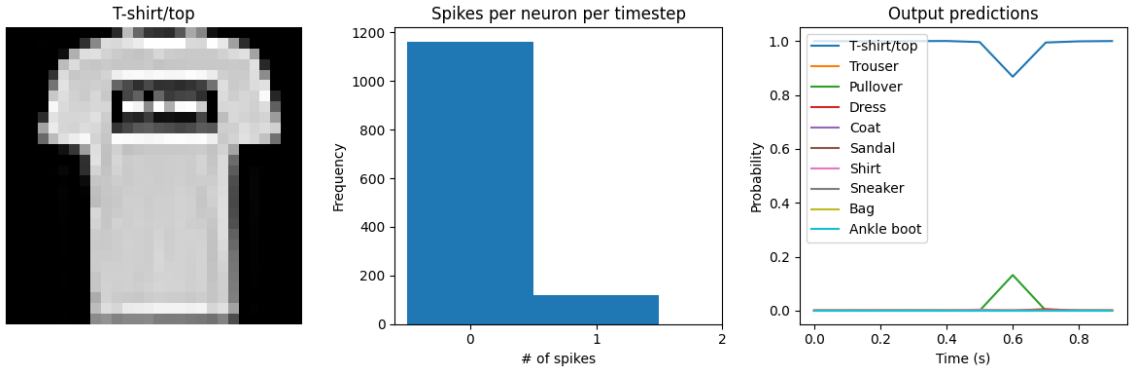


Рисунок В.4– - Приклади зображень набору даних Fashion-MNIST

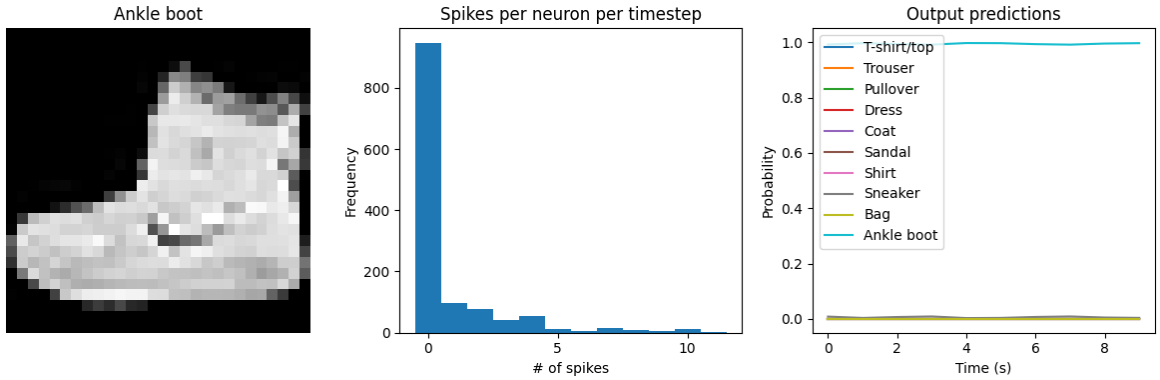


а)

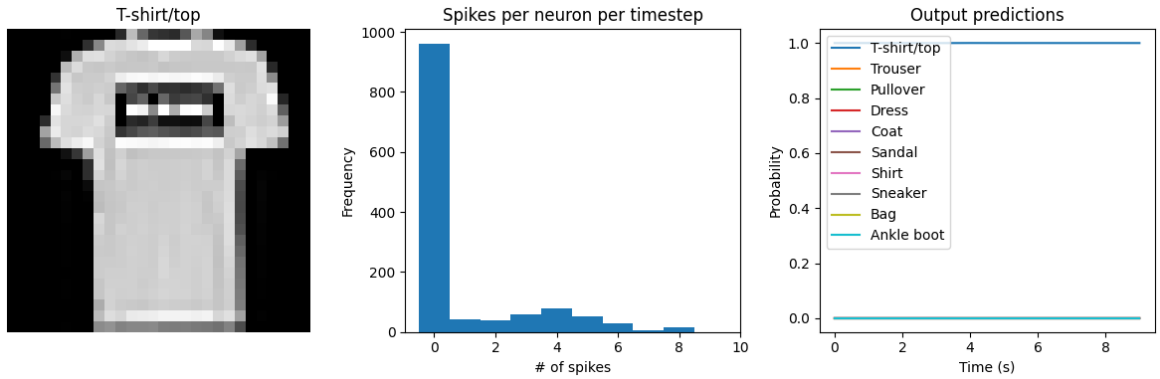


б)

Рисунок В.5 –Приклади візуалізації розпізнавання предметів одягу при dt=0,1



а)



б)

Рисунок В.6 – Приклади візуалізації розпізнавання предметів одягу при dt=1

Таблиця В.1 - Порівняння параметрів аналога та розробленого програмного модуля розпізнавання предметів одягу

	Нейронів у першому шарі	Нейронів у другому шарі	Вид нейрона	Достовірність розпізнавання, %
Аналог	128	10	аналоговий	90,60
Розроблений модуль	128	10	спайкінговий	92,68