Міністерство освіти і науки України
Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»

Факультет систем управління літальними апаратами

Кафедра систем управління літальних апаратів

Пояснювальна записка

до дипломної роботи
магістра
(освітньо-кваліфікаційний рівень)

на тему Дослідження методів обробки відеопотоків з рухомими об'єктами у системах моніторингу і контролю

ХАІ.301.362а.20О.151.153051

Виконав: студент 6 курсу, групи 362а спеціальності

151 "Автоматизація та комп'ютерно-інтегровані технології

Освітньо-професійна програма "Комп'ютерні системи технічного зору"

Черватюк В. В.
(прізвище та ініціали студента)

Керівник Гавриленко О. В.
(прізвище та ініціали)

Рецензент Кузнєцов Ю. А.
(прізвище та ініціали)

м. Харків – 2020 рік

ЗМІСТ

ВСТУП

Сучасний рівень розвитку інформаційних технологій алгоритмів і методів, сприяє появі нових інтерфейсів взаємодії користувачів та пристроїв. Одним з перспективних варіантів є розробка і використання інтерфейсів заснованих на розпізнаванні об’єктів. Технічна система, яка здійснює збір, попередню обробку інформації на низькому рівні, виділяє характерні особливості і закономірності, готує зведений звіт і візуалізує його для людини, називається системою моніторингу. Моніторинг відео потоку передбачає виявлення і відстеження руху об’єктів на відео даних.

Одним з важливих завдань, що вирішуються в системах відео моніторингу, є задача розпізнавання рухомих об'єктів і обчислення параметрів їх руху, що має особливе значення в охоронних системах, системах контролю дорожнього руху і т.п.

На сьогодні існує багато методів для виявлення об’єктів в відео, які застосовуються в залежності від цільової області. У цій сфері, як і в інших напрямках використання IT-технологій, багато безпосередньо залежить від вибору методу вирішення проблеми. Одним із самих поширених є метод розпізнавання за допомогою нейронних мереж. Він є досить поширеним, і використовує близько десятка різних алгоритмів. Найбільш досконалим на даний момент є методи пост-обробки, рекурентних нейронних мереж і оптичних потоків які дозволяють проводити швидке та ефективне розпізнавання, що дозволяють швидко класифікувати отримане зображення.

1. АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРИ І ІСНУЮЧИХ МЕТОДІВ ВИРІШЕННЯ ПРОБЛЕМИ

1.1.Аналіз технічного завдання

У ТЗ наведені вимоги до дослідницького проекту і сформовані питання, що підлягають розробці в проектувальній, експериментальній та реалізаційній частинах. Наведені вимоги до структури та функціонування системи, показників якості, конфігурації обчислювальної техніки. Також в ТЗ передбачені умови експлуатації системи. За допомогою всіх вимог наведених в ТЗ можливо визначити напрямки в яких потрібно провести дослідження, технічні та програмні засоби, підходи до розробки необхідного ПО, його структуру, тестове середовище.

Наведені в ТЗ пункти дозволяють декомпозувати предметну область на відносно не складні задачі які можуть, виконання яких забезпечить чітке слідування цілі даного дослідження. Серед цих пунктів можна виділити необхідність проводити дослідження мінімум на трьох різних моделях, з різними відеоданими за різних умов. Серед цих умов зазначаються погода, освітлення, кількість рухомих об’єктів, це дозволить провести тестування системи в різних умовах та визначити якість роботи кожного методу, що в свою чергу дозволить дати загальну оцінку тому чи іншому методу.

Практично всі операції з відео потребують відносно великих обчислювальних потужностей, тому, однією з самих головних вимог є необхідність проведення всіх обчислень на комп’ютері з потужністю центрального процесора від 400 GFLOPS, або відеокарти з технологією CUDA, що дасть змогу виконувати обробку відеопотоку «на ходу».

Головною вимогою до системи є можливість збору статистичних даних про розпізнані об’єкти в відео. До цього переліку входить ідентифікація

транспортних засобів та пішоходів і збереження їх в структуру даних, що дозволить проаналізувати дорожній трафік відносно часу.

Дотримання всіх вимог дозволить визначити найкращий метод розпізнавання за співвідношенням затрат на функціонування до точності ідентифікації рухомих об’єктів в відеопотоці.

1.2. Аналіз методів обробки відеопотоків з рухомими об’єктами у системах моніторингу і контролю

За останнє десятиліття було зроблено помітну роботу в галузі машинного навчання, особливо в галузі комп'ютерного зору. Починаючи від вдосконалених алгоритмів класифікації, таких як Inception від Google, до новаторської роботи Яна Гудфеллоу над Generative Adversarial Networks для генерування даних із шумів. Цікаво, що в першій половині десятиліття новаторські роботи в галузі комп'ютерного зору здебільшого стосувались обробки зображень, таких як класифікація, виявлення, сегментація та генерація, тоді як сфера обробки відео була менш глибоко досліджена.

Однією з явних причин незначного дисбалансу є те, що відео по суті є послідовністю зображень(кадрів) разом. Однак це визначення не може інкапсулювати повну картину того, що таке обробка відео, і це тому, що обробка відео додає проблемі нову властивість: час. Відео – це не лише послідовність зображень, це скоріше послідовність зв'язаних зображень. Хоча здається що різниці майже немає, дослідники можуть використати цю властивість безліччю способів, які не стосуються окремих зображень. Крім того, через складність відеоданих, розмір та дорогі обчислення навчання та аналізу, проводити дослідження в цій галузі було важче. Однак нещодавно, з випуском ImageNet VID та іншими масивними наборами відеоданих протягом другої половини десятиліття, з'являється все більше дослідницьких робіт,

пов'язаних з відео. У цій роботі здебільшого буде досліджене питання, розпізнавання відео, точніше, як дослідники можуть дослідити рух об’єктів.

1.2.1. Методи пост-обробки

Першим методом, що з’явився, була модифікація, застосована після етапу виявлення об’єкта. Даний метод не вимагає внесення змін в процес розпізнавання зображення, фактично ми можемо взяти будь який існуючий метод розпізнавання зображення і додати пост-обробку. Методи пост-обробки виконуються після розпізнавання кожного кадру, і тому не можуть підвищити продуктивність(обробка може зайняти трохи більше часу). Однак це дозволяє досягти значно більшої точності розпізнавання зображення.

Одним з таких методів є Seq-NMS суть якого полягає в використанні об’єктів із сусідніх кадрів розпізнаних з малою точністю, щоб збільшити кількість слабо розпізнаних об’єктів у тому самому наборі вхідних даних. Обробка відео за допомогою цього алгоритму мінімізує кількість помилкових розпізнань між кадрами або випадкових стрибків розпізнавання і стабілізує вихідний результат, однак використання цього методу потребує більше обчислювальних потужностей. Крім того, для роботи метод вимагає майбутніх кадрів для обробки, що унеможливлює його використання у системах які повинні забезпечувати миттєву реакцію на вхідні дані.

Алгоритм для Seq-NMS складається з трьох частин: вибір послідовності, переоцінка послідовності, подавлення кадрів з високою точністю розпізнавання. По аналогії для одного кадру, перший крок складається з находження максимальної послідовності кадрів з відповідною сумою оцінок в кожному кадрі. Щоб послідовність була належної довжини використовується правило яке відкидає кадри з високим рівнем оцінки, зазвичай математично це означає відсікання кадрів з середнім рівнем оцінки більше 0.5.

Для реалізації даного алгоритму зазвичай використовують динамічне програмування, це дозволяє скоротити час на реалізацію. Другим кроком виконується переоцінка всіх кадрів послідовності за допомогою функції, яка використовує середню оцінку, максимальну оцінку, і найкращу оцінку для одиночного кадру, це в ідеалі, але на практиці зазвичай використовують лише середню оцінку, так як інші параметри не додають суттєвого покращення в порівняння з кількістю затрачених ресурсів. Останнім кроком є заглушення оцінок в кадрах які сильно перекривають оцінку кадрів в послідовності, це робиться уникнення повторного розпізнавання.

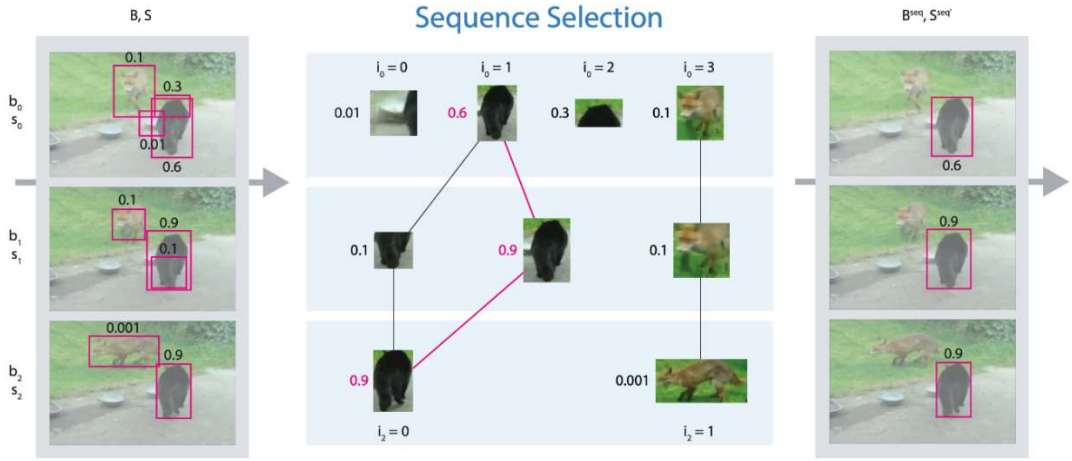

Рисунок 1.1 – Схема функціонування Seq-NMS

1.2.2. Методи рекурентних нейронних мереж

Для отримання кращих результатів у порівнянні з методами пост-обробки, можна схилятися до думки про використання рекурентної нейронної мережі, такої як LSTM. RNN – це спеціальні типи мереж, створені для обробки послідовних даних, включаючи тимчасові дані. Сфера, яка отримала велику

користь від цієї архітектури, – це обробка мови. Можна зауважити, що AWD-LSTM працює нарівні з найсучаснішою моделлю перетворювача BERT, маючи набагато менше параметрів. Було досягнуто певного прогресу в галузі виявлення об’єктів на мобільних пристроях, що говорить нам про те, що даний метод потребує не великих обчислювальних ресурсів.

Ідеєю цього методу є поєднання великих та тяжких нейронних мереж які забезпечують високу точність розпізнавання, та легких варіантів які на основі даних від тяжкої нейронної мережі виконують розпізнавання на основній частині відеопотоку. Це ефективно створює довгострокову пам'ять для архітектури з ключового кадру, який фіксує "суть", яка направляє малу мережу на те, що потрібно виявити. Все це дозволяє отримати багатообіцяючі результати, такі як 70 кадрів в секунду на мобільному пристрої, при цьому все ще досягаючи високої точності для невеликих нейронних мереж на ImageNet VID.

Одним із ключових моментів є те, що архітектура є наскрізною, що означає, що вона робить зображення і виводить масковані дані, також це означає що навчання повинно бути проведено на всій архітектурі. Оскільки ми маємо справу з відеоданими, модель повинна бути навчена великому обсягу даних.

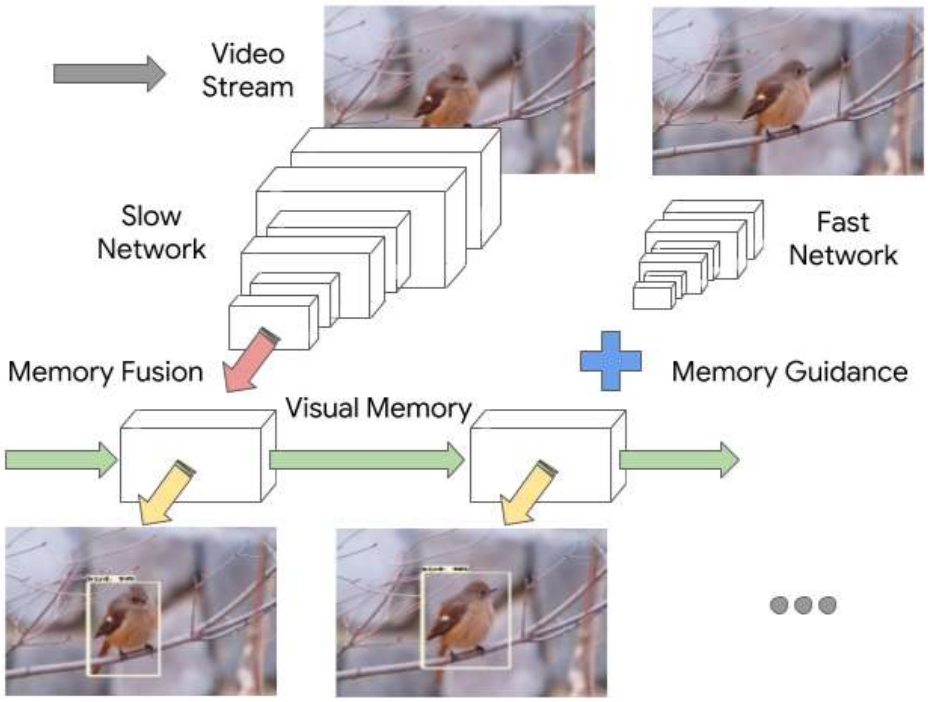

Рисунок 1.2 – Схема функціонування рекурентної нейронної мережі

1.2.3. Метод оптичних потоків

Іншим можливим способом обробки відеопотоків є застосування сучасних детекторів зображень, таких як YOLOv3, або детекторів обличчя, таких як RetinaFace та DSFD, до кожного кадру відеофайлу. Кожен окремий кадр буде використовуватися як вхід для моделі, а результати відео можуть бути настільки точними, як їх середня точність на зображеннях. Однак є два аспекти з безпосереднім застосуванням цих детекторів на кожному окремому кадрі відеофайлу:

– застосування одиночних детекторів зображення на всіх відеокадрах не ефективно, тому що, це спричиняє багато зайвих обчислень, оскільки часто два послідовних кадри з відеофайлу не сильно відрізняються;

– точність також страждає від таких проявів відео, які не часто спостерігаються на нерухомих зображеннях, такі як розмиття в русі, розфокусування.

Тому застосування детекторів до кожного файлу не є ефективним методом вирішення проблеми розпізнавання відео. Однак, існують різні можливі методи, які ми можемо застосувати для вирішення однієї або обох проблем.[1]

Оскільки мережа оптичного потоку може бути порівняно малою, час обробки та обчислювальна потужність, необхідні для таких мереж, менші, ніж об'єктні детектори. Отже, конвеєр функціонує як цикл з n кадрів. Перший кадр називається ключовим. Це кадр, який виявляє детектор об’єктів. Оскільки зараз детектори дають точне виявлення всіх предметів, виявлення будуть підпорядковуватися алгоритмам оптичного потоку. Після отримання векторів переміщення відомі виявлення наступних n-1 кадрів, і цикл повторюється.

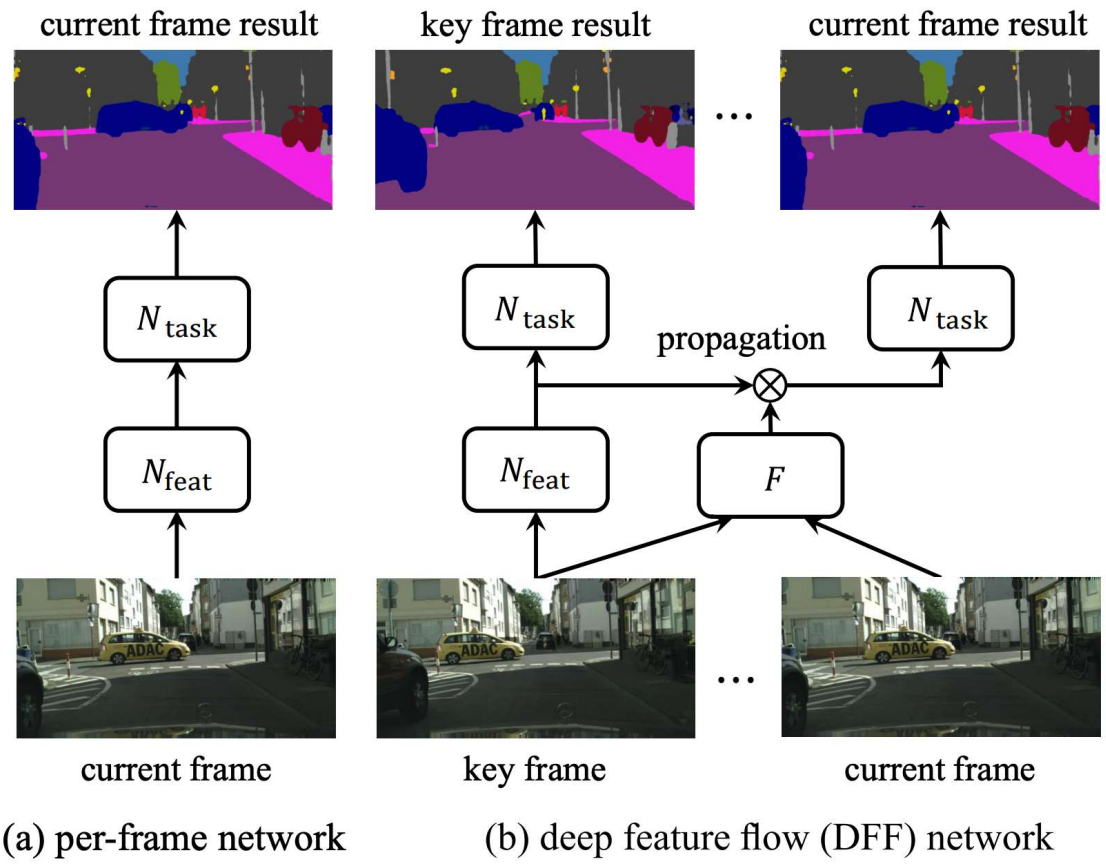

Рисунок 1.3 – Схема функціонування мережі оптичного потоку

Для підвищення точності розпізнавання відео застосовують агрегування декількох кадрів в один. Існують різні способи його реалізації, але всі вони обертаються навколо однієї ідеї: точні обчислюване розпізнавання кадру, та одночасна агрегація сусідніх кадрів до поточного кадру із зваженим усередненням. В результаті поточний кадр отримає додаткові дані від попередніх кадрів, а також деяких наступних кадрів для кращого розпізнавання. Також це може вирішити проблеми з рухом та обрізаними предметами з відеокадру.

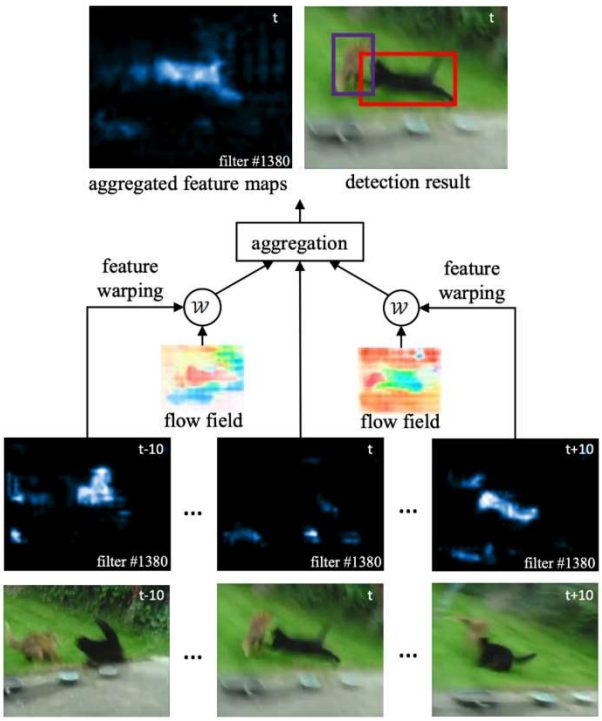

Рисунок 1.4 – Схема агрегування декількох кадрів в один

1.3. Аналіз існуючих програмних засобів

Перш за все розглянемо існуючі програмні реалізації для методів які були перечисленні у попередньому пункті. Для цього проведемо оцінювання за такими показниками як складність застосування, швидкість роботи, можливість для модифікації зі сторони, можливість налаштувань, ціна.

1.3.1. Програмні засоби для Seq-NMS

Аналізуючи доступні програмні рішення для даного методу, перш за все потрібно зазначити, що алгоритм доволі простий і найчастіше використовую

свою власну реалізацію для даного методу в поєднання з так званими детекторами зображення. Але не дивлячись на це були знайденні такі реалізації для цього методу:

– Реалізація irghust
– Реалізація tmoopen

Спочатку розглянемо реалізацію від irghust. Данна реалізація Seq-NMS передбачає використання Faster-RCCN як основного алгоритму для виявлення розпізнавання зображень. З однієї сторони це дозволяє з легкістю використовувати дану реалізацію з вище згаданим алгоритмом розпізнавання зображень. Для того щоб скористатися даною технологією потрібно лише замінити один файл в паці з Faster-RCCN, тому складність застосування є практично нульовою, крім того є інструкція що до того як налаштувати Faster-RCCN для роботи з даною реалізацію методу Seq-NMS. Модифікація даної реалізації потребує додаткових зусиль так як вихідний код має погану структуру, що в свою чергу потребує додаткових зусиль на те щоб розібратися як це все працює. Швидкість роботи є середньою, через те, що дану програму реалізовано на Python, а як відомо ця мова програмування не є швидкою порівняно з мовами програмування більш низького рівня такі як C або C++. Налаштувань практично немає, для того щоб використовувати дану реалізацію потрібно вносити зміни в код, що є не дуже хорошою практикою. Використовувати дану реалізацію може кожен бажаючий вона є повністю безплатною.

Реалізація від tmoopen представляє більш широкі можливості ніж її попередник. Вона не передбачає використання з будь якою із реалізацій алгоритму розпізнавання зображення, натомість користувачеві даної технології необхідно самостійно реалізувати «стик» даної бібліотеки з алгоритмами розпізнавання зображень, що однозначно віддалить початок її

використання на деякий час. Але якщо розглянути дану технологію в контексті того, що ми не будемо використовувати реалізацію від irghust з Faster-RCCN, то тоді однозначно перевага на стороні даної технології, так як її впровадження буде значно скоріше ніж вище описаної технології. Швидкість виконання залишається незмінною в обох випадках так як реалізація виконана на мові програмування Python. У даної технології набагато більші можливості модифікації ніж у її попередника, так як вона добре структурована, що дозволяє просто замінити реалізацію одного з її моделей на свою власну. Налаштування виконуються як і в попередньо описаної технології що однозначно погано. Програма є безкоштовною, тому кожен бажаючий може модифікувати її під свої потреби.

1.3.2. Реалізації для методів рекурентних нейронних мереж

Першою і напевно самою популярною реалізацією даної технології є Keras. Keras – це API для глибокого навчання, написаний на Python, запущений поверх платформи машинного навчання TensorFlow. Він був розроблений з акцентом на швидкі експерименти. Можливість якнайшвидшого переходу від ідеї до результату є ключовим фактором для проведення хороших досліджень.

Keras – це API, розроблений для людей, а не для машин. Keras дотримується найкращих практик зменшення когнітивного навантаження: пропонує послідовні та прості API, мінімізує кількість дій користувача, необхідних для використання, а також забезпечує чіткий та ефективний зворотній зв’язок про помилки користувача.

Це робить Keras простим у вивченні та зручним у використанні. Він дозволяє користувачеві бути більш продуктивним, дозволяє швидше випробувати більше ідей що, в свою чергу, допомагає досягати кращих

результатів. Ця простота використання не обумовлена зниженням гнучкості: оскільки Keras глибоко інтегрується з низькорівневою функціональністю TensorFlow, це дозволяє розробляти робочі процеси, де будь-яку частину функціональності можна налаштувати.

Нижче наведений графік на якому показано як часто вибирають Keras на основі даних з змагання по машинному навчанню Kaggle.

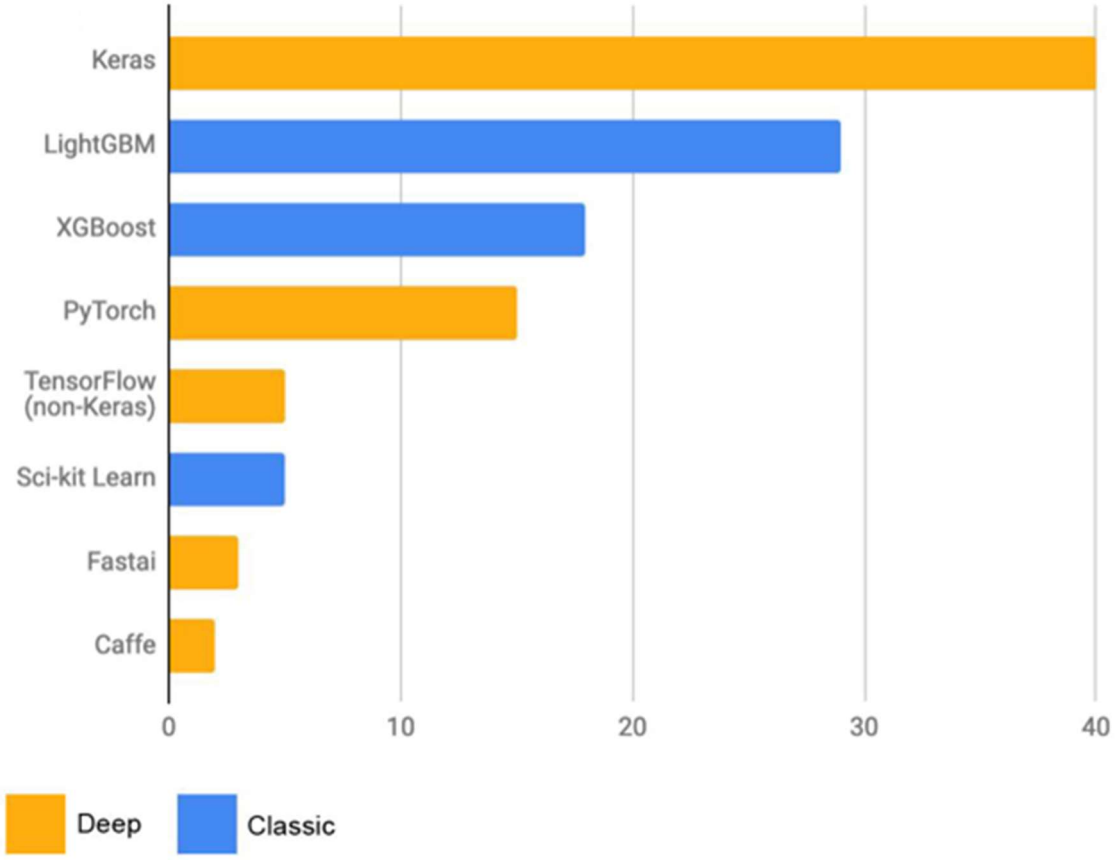

Рисунок 1.5 - Порівняльна характеристика частоти використання Keras командами переможцями Kaggle до кожного змагання

Наступною розглянемо TensorFlow яка може використовуватися для навчання та роботи глибоких нейронних мереж для рукописної класифікації цифр, розпізнавання зображень, вбудовування слів, рекурентних нейронних мереж, обробки природних мов та моделювання на основі PDE (рівняння

часткових диференціальних змінних). Даний продукт підтримує центральні процесори та графічні процесори Nvidia. Він може працювати на Ubuntu Linux, MacOS, Android, iOS та (краще, ніж раніше) Windows. Може використовуватися для прогнозування за допомогою тих самих моделей, що використовуються для навчання. Пропонує найкращу підтримку для використання з мовою програмування Python.

Починаючи з r0.10, TensorFlow має багато удосконалень та додаткових можливостей. Наприклад, підтримку бібліотеки CUDA та cuDNN, що підвищило продуктивність, за допомогою оптимізованого коду для останніх графічних процесорів Nvidia. Є підтримка HDFS (файлова система Hadoop), та язика програмування Go. Також випущено XLA, який є доменним компілятором для графіків TensorFlow, що покращує продуктивність, і відлачик TensorFlow. У той же час TensorFlow є сумісним з стандартною інфраструктурою Python, такою як підтримка інструментів PyPI та pip, пакету NumPy, який широко використовується для обчислень.

Є підтримка RNN (періодичні нейронні мережі, які часто використовуються для обробки природних мов), та нової інтеграції Intel MKL (Math Kernel Library) для покращення продуктивності навчання на центральному процесорі. Для зручності програмування в бібліотеку були додані регресори та класифікатори, бібліотеки для статистичного розподілу, обробки сигналів та диференційованої передискретизації зображень.

Є бібліотека наборів навчальних даних також реалізована зворотна сумісність; це корисно для розробки нових моделей для стандартних навчальних наборів даних.

Для полегшення навчання TensorFlow є навчальні матеріали та вдосконалила навчальні посібники. Зараз є багато друкованих книг про TensorFlow та декілька онлайн-курсів.

Кілька нових розділів бібліотеки TensorFlow пропонують інтерфейси, які вимагають менше програмування для створення та навчання моделей. Сюди входять tf.estimator, який надає ряд високоякісних засобів для роботи з моделями – як регресори, так і класифікатори для лінійних глибоких нейронних мережі (DNN), а також комбіновані лінійні та DNN, плюс базовий клас, з якого ви можете створити власні оцінювачі. Крім того, API набору даних дозволяє створювати складні вхідні конвеєри з менших простих.

TensorFlow Lite – це полегшене рішення TensorFlow для мобільних та вбудованих пристроїв, яке забезпечує роботу нейронних мереж на пристроях(але не навчання) з малими обчислювальними можливостями. TensorFlow Lite також підтримує апаратне прискорення за допомогою API нейронних мереж Android. Моделі TensorFlow Lite досить малі, щоб працювати на мобільних пристроях, і можуть використовуватися в автономному режимі.

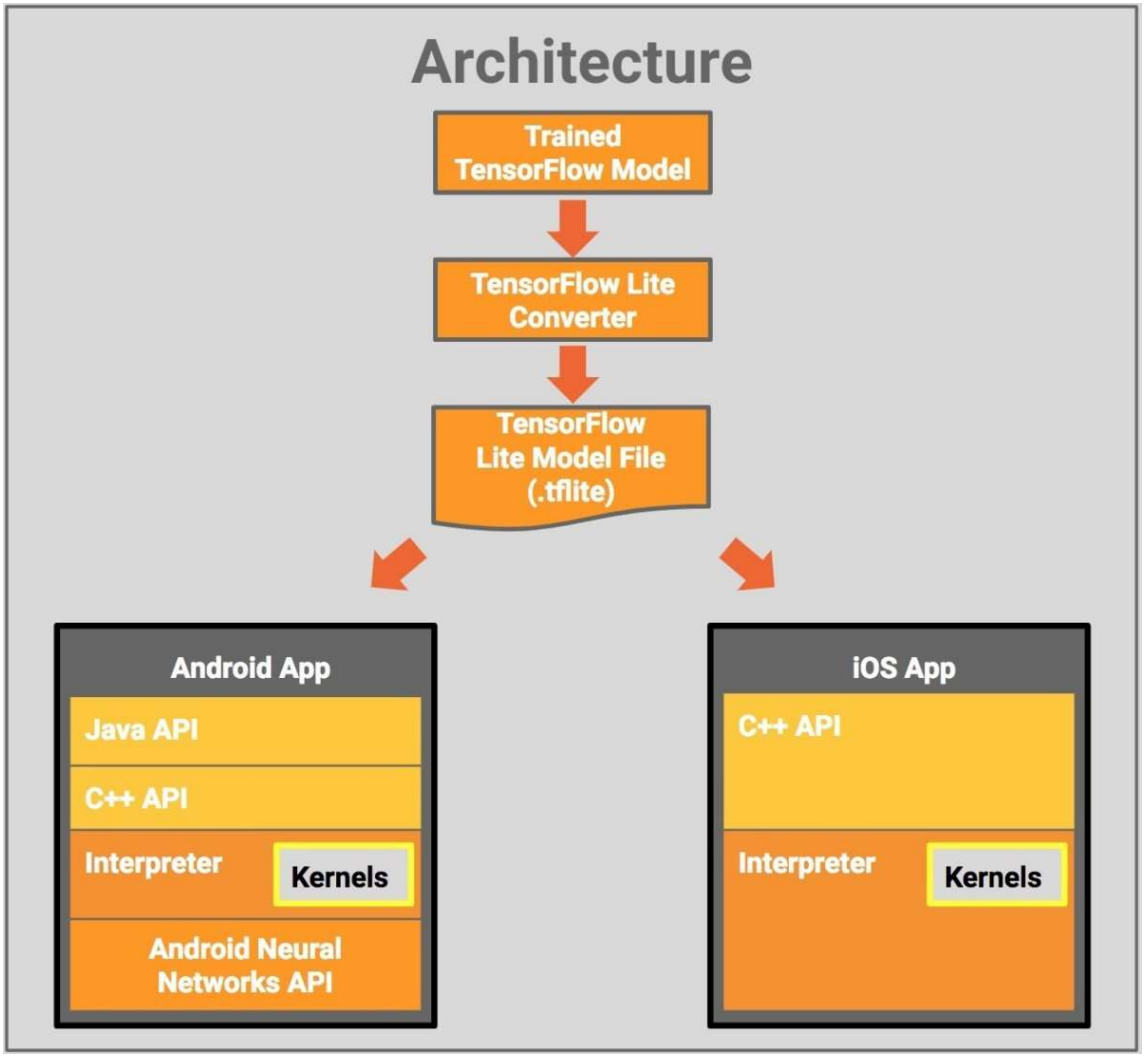

Рисунок 1.6 - Архітектура TensorFlow Lite

TensorFlow Lite дозволяє досить невеликим моделям нейронних мереж працювати на пристроях Android та iOS, та не вимагає підключення до інтернету. Основна ідея TensorFlow Lite полягає в тому, що спочатку тренується повноцінна модель TensorFlow, далі вона перетворюється у формат моделі TensorFlow Lite. Після цього вона може використовуватися у мобільному додатку на Android або iOS. Крім того, може бути використана одна з попередньо навчених моделей TensorFlow Lite для класифікації зображень.

TensorFlow Serving – це гнучка, високоефективна система для моделей машинного навчання, розроблена для промислового використання. Можна мати кілька сервісних версій із декількома версіями, і клієнти можуть запитувати або останню версію, або конкретний ідентифікатор версії для конкретної моделі, що полегшує випробування нових алгоритмів та експериментів. Ви можете представляти складені моделі як декілька незалежних сервлеблів або як одиничні композитні сервлеби. Доступ до сервісних елементів контролюється джерелами, і джерела можуть знаходити сервірувані дані з довільних систем зберігання.

Eager – це експериментальний інтерфейс до TensorFlow, який забезпечує обов'язковий стиль програмування, подібний до NumPy. Це ще один корисний спосіб спрощення коду для матричних операцій і моделей TensorFlow Eager сумісне з масивами NumPy, прискоренням графічного процесора, автоматичною диференціацією та використанням класів шарів в модулі tf.layers.

Загалом, TensorFlow залишається на передньому краї машинного навчання та систем глибокого навчання. Як вже було згадано, TensorFlow добре себе показує у сферах продуктивності, розгортання, простоти навчання, простоти програмування та сумісності із загальноприйнятими бібліотеками та утилітами Python.

Основними конкурентами є MXNet, набір інструментів Microsoft Cognitive Toolkit та TensorBoard, Facebook Caffe2 від Facebook. Двигун прогнозування H2O.ai з відкритим кодом був вдосконалений завдяки чудовій фірмовій настройці гіперпараметрів та інженерному рівню, Driverless AI, що варто, але недешево. Scikit-learn продовжує залишатись приємним для використання в рамках своїх обмежень, підтримуючи ML, але не глибокі нейронні мережі. A Spark MLlib - чудовий варіант для тих, хто не потребує

тренування глибоких нейронних мереж. TensorFlow є прекрасним вибором для побудови моделей глибокого навчання.

Таблиця 1.1 – Порівняльна таблиця інструментів розробки програм для розпізнавання відео

Назва інструменту	Платформа	Ціна	Підтримувана мова програмування	Підтримувані функції
Scikit Learn	Linux, Mac OS, Windows	Безкоштовно	Python, Cython, C, C++	класифікація, регресія, скупчення, попередня обробка
PyTorch	Linux, Mac OS, Windows	Безкоштовно	Python, C++, CUDA	Модуль Autograd Модуль Optim Модуль nn
TensorFlow	Linux, Mac OS, Windows, Android, IOS	Безкоштовно	Python, C++, CUDA	Завантаження та перетворення даних Попередня обробка та візуалізація даних.
Keras	Linux, Mac OS, Windows, Android, IOS	Безкоштовно	Python	API для нейронних мереж

Продовження таблиці 1.1

Назва інструменту	Платформа	Ціна	Підтримувана мова програмування	Підтримувані функції
irghust	Linux, Mac OS, Windows, Android, IOS	Безкоштовно	Python	Пост обробка
tmoopen	Linux, Mac OS, Windows, Android, IOS	Безкоштовно	Python	Пост обробка
irghust	Linux, Mac OS, Windows, Android, IOS	Безкоштовно	Python	Пост обробка

1.4.Постановка задачі проектування

З огляду на методи вирішення проблеми та програмні продукти для їх реалізації можна зробити висновок про те, що для даної дослідницької роботи буде достатньо реалізувати кожен метод розпізнавання зображень за допомогою існуючих програмних продуктів. Для реалізації методу Seq-NMS потрібно використати реалізації від irghust та tmoopen. Також є бажаним написати свою власну реалізацію і порівняти існуючі рішення з самописною програмою.

Для інших методів потрібно використовувати рішення на TensorFlow і Keras, крім того потрібно використати готові навчені моделі, і зробити порівняння кожного методу з кожною моделлю. Моделі повинні бути трьох рівнів точності і відповідно обчислювальної складності для забезпечення розгляду кожного методу через різні призми швидкості та точності. В результаті повинні бути отримані тестові середовища для кожної пари метод-модель, які в свою чергу повинні забезпечувати можливість збору всієї необхідної інформації для подальшої обробки даних.

2. ДОСЛІДЖЕННЯ ПРОБЛЕМИ І СИНТЕЗ МАТЕМАТИЧНОЇ МОДЕЛІ

2.1.Математичні моделі

Більшість методів виявлення об'єктів (R-CNN) призначені для виявлення об'єктів на одному незалежному кадрі. Однак, оскільки ми стурбовані виявленням об’єктів у відео, було б марно витратити важливу інформацію, щоб повністю ігнорувати часовий компонент. Однією з проблем, яку можна помітити при R-CNN на наборі для перевірки, є те, що не максимальне придушення (NMS) часто вибирає неправильний обмежувальний квадрат після класифікації об'єктів. Воно обирає надмірно великі рамки, що призводить до меншого перетину над об’єднанням (IoU) із основною рамкою істини, оскільки об’єднання площ є великим. Великі рамки часто мають дуже високі показники оцінок, можливо тому, що під час об’єднання RoI не доступна додаткова інформація. Для боротьби з цією проблемою використовується переміщення рамок в різних кадрах. Припускається, що сусідні кадри повинні мати подібні об'єкти, а їх обмежувальні рамки повинні бути однаковими за положенням і розміром, тобто часовою узгодженістю.

Щоб використати цю тимчасову узгодженість, можна застосувати евристичний метод для перерахування обмежувальних полів у відео під назвою Seq-NMS. Seq-NMS має три етапи: крок перший вибір послідовності; крок 2 переоцінка послідовності; крок 3 придушення. Ми повторюємо ці три кроки, поки не залишиться жодної послідовності. Рисунок 1 ілюструє цей процес.

Seq-NMS виконується на одному відео V, який складається з набору T рамок, $\{v_0, \dots, v_T\}$. Для кожного кадру t , ми маємо набір областей пропозицій

регіону b_t і оцінку s_t для обох за розміром n_t, яка змінюється для кожного кадру. Набір пропозицій для цілого відео позначається символом

$$B = \{b_0, \dots, b_T\}.$$

Так само набір оцінок для всього відео позначається символом

$$S = \{s_0, \dots, s_T\}.$$

Враховуючи набір областей, що обмежують область B, та оцінки їх виявлення S як вхідні дані, вибір послідовності має підмножину полів B^{seq} та пов’язані з ними бали $S^{seq'}$. Функція переоцінки приймає $S^{seq'}$ і створює новий набір оцінок S^{seq}.

Вибір послідовності. Для кожної пари сусідніх кадрів у V, обмежувальний блок у першому кадрі може бути пов'язаний з обмежувальним полем у другому кадрі, якщо їх IoU перевищує деякий поріг. Ми знаходимо потенційні зв'язки в кожній парі сусідніх кадрів у всьому кліпі. Потім ми намагаємось знайти максимальну послідовність балів у всьому кліпі. Тобто, ми намагаємось знайти послідовність блоків, які максимізують суму оцінок об’єктів, з урахуванням обмеження, що всі сусідні поля повинні бути пов’язані.

$$i' = \underset{i_{t_s},\dots,i_{t_e}}{arg\max} \sum_{t=t_s}^{t_e} s_t[i_t]$$

$$0 \le t_s \le t_e < T$$

$$IoU(b_t[i_t], b_{t+1}[i_{t+1}] > 0.5, \forall t \in [t_s t_e)$$

Це можна ефективно знайти, використовуючи простий алгоритм динамічного програмування, який підтримує максимальну послідовність оцінок на сьогодні в кожному полі. Оптимізація повертає набір індексів i'які використовуються для вилучення послідовності рамок

$$B^{seq} = \{b_{t_s}[i_{t_s}], \dots b_{t_e}[i_{t_e}]\}$$

та їх бали

$$S^{seq'} = \{s_{t_s}[i_{t_s}], \dots s_{t_e}[i_{t_e}]\}.$$

Повторна оцінка послідовності. Після вибору послідовності оцінки в ній покращуються. Ми застосовуємо функцію F до оцінок послідовності, щоб отримати

$$S^{seq} = F(S^{seq'}).$$

Використовуються дві різні функції переоцінки: середня та максимальна.

Придушення. Поля в послідовності видаляються з набору вікон, які ми посилаємо. Крім того, ми застосовуємо дане правило всередині кадрів таким чином, що якщо обмежувальне поле у кадрі $t, t \in [t_s, t_e]$, та має IoU з b_tчерез деякий поріг, він також вилучається з набору кандидатів.

Якщо дано завдання розпізнавання зображення за допомогою загорткової нейтральної мережі N, що виводить результат для вхідного зображення I як $y = N\ (I)$. Наша мета - застосувати мережу до всіх відеокадрів I_i, $i\ =\ 0, \dots, \infty$, швидко і точно.

Враховуючі сучасні архітектури CNN та додатки можна розкласти N на дві послідовні підмережі. Перша підмережа N_{feat}, яка називається мережею функцій, повністю згорнута і виводить ряд проміжних карт функцій,

$$f = N_{feat}(I).$$

Наступна підмережа N_{task}, називається мережею завдань, має специфічну структуру та виконує розпізнавання на картах об’єктів,

$$y = N_{task}(f).$$

Послідовні відеокадри дуже схожі. Подібність ще сильніша у глибоких картах об’єктів, які кодують семантичні поняття високого рівня. Можна використати схожість для зменшення обчислювальних витрат. Зокрема, функціональна мережа N_{feat} працює лише на розріджених ключових кадрах.

Карти функцій інших кадрів I_i поширюються з їх попередніх кадрів I_k. Особливості глибоких згорткових шарів кодують семантичні поняття які відповідають просторовим розташуванням в зображеннях. Така просторова відповідність дозволяє нам дешево розповсюджувати карти об’єктів способом просторової деформації.

Нехай $M_{i\to k}$ – двовимірне поле потоку. Його можна отримати за допомогою алгоритму оцінки потоку F, такого як, $M_{i\to k} = \mathrm{F}(I_k, I_i)$. Розмір цього поля білінійно змінено до тієї ж роздільної здатності як у карт об'єктів для розповсюдження. Це проектує назад місце розташування p у поточному кадрі i до розташування $p + \delta p$ у ключовому кадрі k, де $\delta p = M_{i\to k}(p)$.

Оскільки значення δp загалом дробові, ознака викривлення обчислюється за допомогою білінійної інтерполяції

$$f_i^c = \sum_q G(q, p + \delta p) f_i^c(q)$$

де c ідентифікує канал на картах об’єктів f, q перераховують всі просторові розташування на картах об’єктів, а $G(\cdot,\cdot)$ позначає білінійне корінь інтерполяції. Можна звернути увагу, що G – двох-вимірний і може бути розділений на два одновимірних

$$G(q, p + \delta p) = g(q_x, p_x + \delta p_x) \cdot g(q_y, p_y + \delta p_y)$$

де $g(a, b) = max(0, 1 - |a - b|)$.

Просторове викривлення може бути неточним через помилки в оцінках потоку, оклюзія об’єкта тощо. Щоб краще визначити ознаки, їх амплітуди модулюються за допомогою «масштабування поля» $S_i \to k$, яке має такі ж просторові та розмірні розміри каналів, що і карти об'єктів. Поле масштабу отримується шляхом застосування “функції масштабу” S на двох кадрах,

$$S_{i \to k} = S(I_k, I_i).$$

Так як, функція поширення ознак визначається як

$$f_i = W(f_k, M_{i \to k}, S_{i \to k}),$$

де W застосовує рівняння для всіх локацій та всіх каналів на картах об’єктів і множить об’єкти з масштабами $S_{i \to k}$ стихійно.

Запропонований алгоритм розпізнавання відео називається глибокий функціональний потік.

Згортання, кероване потоком, відбувається з врахуванням опорного кадру I_i та сусідній кадр I_j, поле потоку $M_{i\to j} = F(I_i, I_j)$ оцінюється потоковою мережею F. Карти об’єктів на сусідній рамці перекошені система відліку відповідно до потоку. Викривлення функція визначається як

$$f_{j\to i} = W(f_j, M_{i\to j}) = W(f_j, F(I_i, I_j)),$$

де $W(\cdot)$ – функція білінійного викривлення, застосована до всіх розташування кожного каналу на картах функцій та $f_{j\to i}$ позначає особливості карт, перекошених від кадру j до кадру i.

Після згортання об’єктів в системі відліку накопичується декілька карт об’єктів з сусідніх кадрів(включаючи власні). Ці особливості карти надають різноманітну інформацію про екземпляри об'єкта (наприклад, різноманітне освітлення, точки зору, пози, несуттєві деформації). Для агрегування використовується різна вага в різних місцях це дозволяє всім каналам функцій спільно використовувати таку ж вагу. Двовимірні вагові карти для згорнутих елементів $f_{j\to i}$ позначаються як $w_{j\to i}$. Сукупні ознаки в системі відліку $\bar{f}_i$ можна виразити як

$$\bar{f}_i = \sum_{j=i-K}^{i+K} w_{j\to i} f_{j\to i},$$

де K вказує діапазон сусідніх кадрів для агрегації (за замовчуванням $K = 10$). Це рівняння подібне до формули моделей уваги, де змінюються ваги призначені функціям в буфері пам'яті.

Потім агреговані ознаки $\bar{f}_i$ подаються у виявлення підмережа для отримання результатів,

$$y_i = N_{det}(\bar{f}_i)$$

У порівнянні з базовим методом та попередніми методами, даний метод збирає інформацію з декількох кадрів до отримання остаточних результатів виявлення.

Адаптивна вага вказує на важливість усіх буферних кадрів $[I_i - K, \ldots, I_i + K]$ до опорного кадру I_i в кожному просторовому розташуванні. Зокрема, у розташуванні p, якщо спотворені елементи $f_{j\to i}(p)$ близькі до ознак $f_{\mathrm{i}}(\mathrm{p})$, їм присвоюється більша вага, в іншому випадку менша вага. Тут використовується метрика схожості косинусів для вимірювання подібності між викривленими ознаками та ознаками.

Більше того, безпосередньо не використовуємо згорткові особливості, отримані від $N_{feat}\ (I)$. Натомість застосовується крихітна повністю згорнута мережа $E\ (\cdot)$ до властивостей f_i та $f_{j\to i}$, які проектують функції для вимірювання схожості та називається підмережою.

Вага оцінюється за допомогою наступної формули

$$w_{j\to i}(p) = \exp\left(\frac{f^e_{j\to i}(p) \cdot f^e_i(p)}{\left|f^e_{j\to i}(p)\right|\left|f^e_i(p)\right|}\right),$$

де $f^e = E(f)$ позначає функції для вимірювання подібності, а вага $w_{j\to i}$ нормується для кожного просторового розташування p над найближчими кадрами,

$$\sum_{j=i-K}^{i+K} w_{j\to i}(p) = 1.$$

Оцінка ваги може бути розглянута як процес, знаходження подібності косинусів функцій які проходять через операцію SoftMax.

Навчання глибоких характеристик потоку функцій спочатку розроблена для отримання відповідності зображення низького рівня пікселів. Це може бути швидким висновком, але може бути недостатньо точним для завдання розпізнавання, в якому високий рівень карти функцій змінюється по-різному, як правило, повільніше, ніж пікселі [21, 39]. Для моделювання таких варіацій ми пропонуємо також використовувати CNN для оцінки поля потоку та такого масштабу що всі компоненти можуть бути спільно навчені в цілому для поточної задачі.[2]

Навчання виконується стохастичним градієнтним спуском (SGD).

У кожному міні-пакеті пара випадкових відеокадрів, $\{I_k, I_i\}, 0 \leq i - k \leq 9$, вибірково вибирається. У прямому проході функціональна мережева N_{feat} застосовується на I_k для отримання карт функцій f_k. Далі потікова мережа F працює на кадрахI_i, I_k для оцінки поля потоку та поля масштабу. Коли i > k, карти функцій f_k поширюються на f_i. В іншому випадку карти об'єктів ідентичні, і розповсюдження не проводиться. Нарешті, мережа завдань N_{task} застосовується на f_i для отримання результату y_i, який несе збиток проти основного результату істини.

Градієнти помилок втрат знову поширюються по всьому, щоб оновити всі компоненти. Зверніть увагу, що наше навчання враховує особливий випадок, коли i = k, і вироджується до тренувального навчання.

Потокова мережа набагато швидша, ніж функціональна мережа. Він пройшов попередню підготовку на наборі даних Flying Chairs. Потім ми додаємо функцію масштабування S як вихід однорангових елементів в кінці мережі, шляхом відповідного збільшення кількості каналів в останньому згортковому шарі. Функція шкали ініціалізується для всіх (ваги та

упередження у вихідному рівні ініціалізуються як 0s та 1s відповідно). Потім мережа розширеного потоку доопрацьовується.

Функція поширення ознак є нетрадиційною. Це без параметрів і повністю диференціюється. При зворотному розповсюдженні ми обчислюємо похідну ознак у f_i відносно ознак у f_k, поле масштабу $S_i \to k$ та поле потоку $M_i \to k$.

Перші два легко обчислити, використовуючи правило ланцюга. Нарешті ми маємо кожен канал c і розташування p у поточному кадрі

$$\frac{\partial f_i^C(p)}{\partial M_{i\to k}(p)} = S_{i\to k}^c(p) \sum_q \frac{\partial G\left(q, p+\delta_p\right)}{\partial \delta p} f_k^c(q)$$

Зверніть увагу, що поле потоку M (·) є двовимірним, і ми використовуємо $\partial\delta p$ для позначення $\partial\delta p_x$ та $\partial\delta p_y$ для простоти.

Запропонований метод можна легко зробити на наборах даних, де анотуються лише розріджені кадри, що зазвичай трапляється через високі витрати на маркування в задачах розпізнавання відео.

У цьому випадку для навчання на кадрі можна використовувати лише анотовані кадри, тоді як DFF може легко використовувати всі кадри, доки фрейм I_i буде анотований. Іншими словами, DFF може повною мірою використовувати дані з анотацією. Це потенційно вигідно для багатьох завдань розпізнавання відео.

Аналіз складності висновків для кожного неключового кадру, коефіцієнт обчислювальних витрат запропонованого підходу та підходу для кадру

$$r = \frac{O(F) + O(S) + O(W) + O(N_{task})}{O(N_{feat}) + O(N_{task})}$$

де O (·) вимірює складність функції.

Щоб зрозуміти це співвідношення, спочатку зауважимо, що складність N_{task}, як правило, невелика. Хоча його точка розбиття в N є довільною, як підтверджено в експерименті, достатньо, щоб було збережено лише один шар ваги, який можна знайти, у N_{task}. Хоча N_f і F мають значну складність, ми маємо

$$O(N_{task}) \ll O(N_{feat}) \text{ та } O(N_{task}) \ll O(F)$$

У нас також є $O(W) \ll O(F)$ та $O(S) \ll O(F)$, оскільки W і S дуже прості. Таким чином, відношення рівняння є наближеним як

$$r \approx \frac{O(F)}{O(N_{feat})}$$

Це в основному визначається співвідношенням складності потокової мережі F та особливостей мережі N_f, які можна точно виміряти, наприклад, за їх FLOPs.

У порівнянні з підходом до кадру, загальний коефіцієнт прискорення в алгоритмі також залежить від розрідженості ключових кадрів. Нехай у кожному l послідовних кадрів буде один ключовий кадр, коефіцієнт прискорення становить

$$s = \frac{l}{1 + (l - 1) * r}$$

Планування ключового кадру яке зазначено в алгоритмі та рівнянні, вирішальним фактором для швидкості висновку є час виділення нового ключового кадру. Використовуємо просте планування ключового кадру, тобто тривалість ключового кадру l є фіксованою константою. Це легко реалізувати та налаштувати.

Однак для різноманітних змін вмісту зображення може знадобитися різний l, щоб забезпечити плавний компроміс між точністю та швидкістю. В ідеалі слід виділити новий ключовий кадр, коли вміст зображення різко змінюється.

Таблиця 2.1 – Оцінка складності алгоритмів

	FlowNet	FlowNet Half	FlowNet Inception
ResNet-50	9,20	33,56	68,97
ResNet-101	12,71	46,30	95,24

Для будь-якого навчального якоря і ми мінімізуємо наступні багатозадачні втрати:

$$L = L_{cls}(p_i, p_i^*) + \lambda_1 p_i^* L_{box}(t_i, t_i^*) + \lambda_2 p_i^* L_{pts}(l_i, l_i^*) + \lambda_3 p_i^* L_{pixel}.$$

Класифікація обличь $L_{cls}(p_i, p_i^*)$, де p_i прогнозоване місце якоря і, а p_i^* дорівнює одиниці для позитивного і нулю для негативного якоря. Класифікаційні втрати L_{cls} це втрати для бінарних класів(обличчя або не обличчя). $L_{box}(t_i, t_i^*)$, де $t_i = \{t_x, t_y, t_w, t_h\}$ і $t_i^* = \{t_x^*, t_y^*, t_w^*, t_h^*\}$ представляють координати передбачуваного вікна які пов'язані з позитивним якорем. Потрібно нормалізувати положення рамки(тобто розташування в центрі, ширина та висота), для цього використовується $L_{box}(t_i, t_i^*) = R(t_i - t_i^*)$,

де R – функція втрат.

2.2.Метод для включення згорткових LSTM

Метод для включення згорткових LSTM у рамки виявлення одного зображення як засіб поширення інформації на рівні кадру в часі. Однак інтеграція LSTM несе значні обчислювальні витрати і заважає мережі працювати в режимі реального часу. Для вирішення цієї проблеми вводиться вузьке місце – LSTM із розділеними поглибленнями звивин та принципами проектування вузьких місць для зменшення обчислювальних витрат. Остаточна модель перевершує аналогічні однокадрові детектори за точністю, швидкістю та розміром моделі.

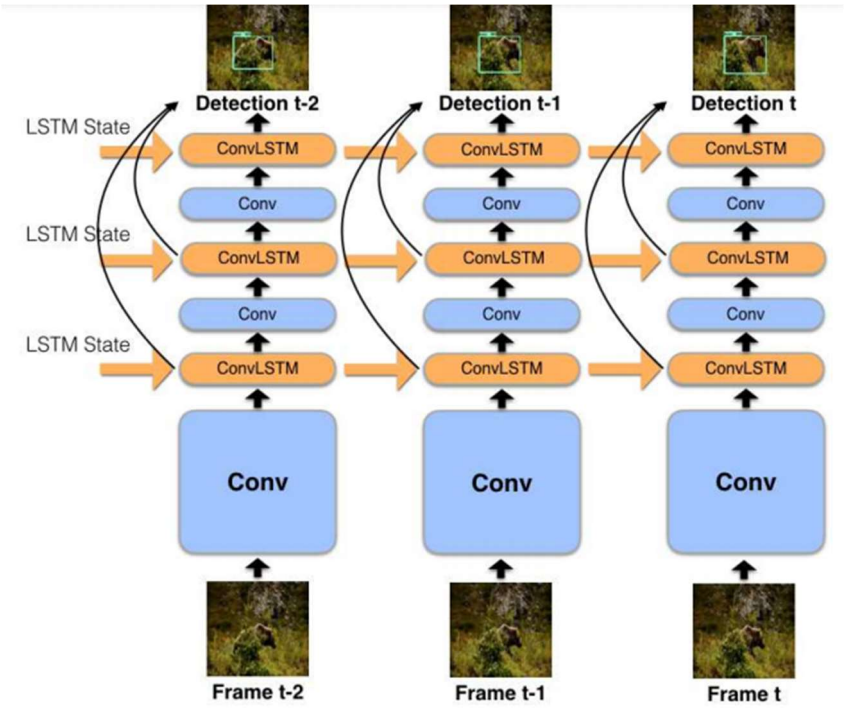

Рисунок 2.1 – Приклад ілюстрації спільної моделі LSTM-SSD

Інтеграція згорткових LSTM з SSD

Розглянемо відео як послідовність кадрів зображень V = {I_0, I_1,. . . B}. Наша мета - відновити виявлення на рівні кадру {D_0, D_1,. . . D_n}, де кожен D_k – це перелік розташувань обмежувальних коробок та передбачень класів, що відповідають зображенню I_k. Розглядаємо налаштування в Інтернеті, де виявлення D_k передбачається з використанням лише кадрів до I_k.

Модель прогнозування можна розглядати як функцію

$$F(I_t, S_{t-1}) = (D_t, S_t)$$

де $S_t = \{S_k^0, S_k^0, \ldots S_k^{m-1}\}$ визначається як вектор функціональних карт, що описують відео до кадру k. Можна використовувати нейромережу з m шарами LSTM для апроксимації цієї функції, де кожна карта особливостей S_{t-1} використовується як вхід стану для одного LSTM, а кожна карта особливостей S_t отримується з виходу стану LSTM. Щоб отримати виявлення у всьому відео, просто запускається кожне зображення через мережу послідовно.

Для побудови моделі спочатку приймається фреймворк SSD, заснований на архітектурі Mobilenet, і замінюються всі згорткові шари в шарах функцій SSD на глибокі сепараційні звивини. Також обрізається базова мережа Mobilenet, видаляючи остаточний шар. Замість того, щоб мати окремі мережі виявлення та LSTM, потім вводяться згорткові шари LSTM безпосередньо в однокадровий детектор. Свертові LSTM дозволяють мережі кодувати як просторову, так і часову інформацію, створюючи уніфіковану модель для обробки тимчасових потоків зображень.

Уточнення функцій за допомогою LSTM

Застосовуючи відеопослідовності, можна інтерпретувати LSTM-стани як функції, що представляють часовий контекст. Потім LSTM може використовувати часовий контекст для вдосконалення свого введення на кожному часовому кроці, одночасно витягуючи додаткові тимчасові сигнали з введення та оновлюючи свій стан. Цей режим уточнення є достатньо загальним для застосування на будь-якій проміжній карті об'єктів, розміщуючи шар LSTM відразу після нього. Карта функцій використовується як вхід для LSTM, а вихід LSTM замінює карту функцій у всіх майбутніх обчисленнях.

Спершу визначається однокадровий детектор як функцію $G(I_t) = D_t$. Ця функція буде використана для побудови складеної мережі з m шарами LSTM. Можна розглядати розміщення цих LSTM як розподіл шарів G на m + 1 підмережі $\{g_0, g_1, \ldots g_m\}$ задовільний

$$G(I_t) = (g_m \cdot \cdots \cdot g_1 \cdot g_0)(I_t)$$

Також визначається кожен рівень LSTM $L_0, L_1, \ldots L_{m-1}$ як функція

$$L_k(M, s_{t-1}^k) = (M_+, s_t^k)$$

де M і M_+ – представити карти одного розміру. Тепер шляхом послідовних обчислень

$$(M_+^0, s_t^0) = L_0(g_0(I_t), s_{t-1}^0$$
$$(M_+^1, s_t^1) = L_1(g_1(M_+^0), s_{t-1}^1$$

сформувавши функцію $F(I_t, s_{t-1}) = (D_t, s_t)$, що представляє мережу LSTM-SSD. На практиці входи та виходи рівнів LSTM можуть мати різні розміри, але однакові обчислення можуть виконуватися до тих пір, поки перший згорнутий шар кожної підмережі F змінює свої вхідні розміри.

У цій архітектурі експериментально вибирається розділи G. Зверніть увагу, що розміщення LSTM раніше призводить до більших обсягів вводу, і обчислювальні витрати швидко стають непомірними. Щоб зробити можливим додане обчислення, розглядається розміщення LSTM лише після карт функцій з найменшими просторовими розмірами, що обмежується шаром Conv13 та картами характеристик SSD. В рамках цих обмежень розглянемо кілька способів розміщення шарів LSTM:

— Помістити один LSTM після шару Conv13

— Скласти кілька LSTM після шару Conv13

— Помістити один LSTM після кожної карти об’єктів

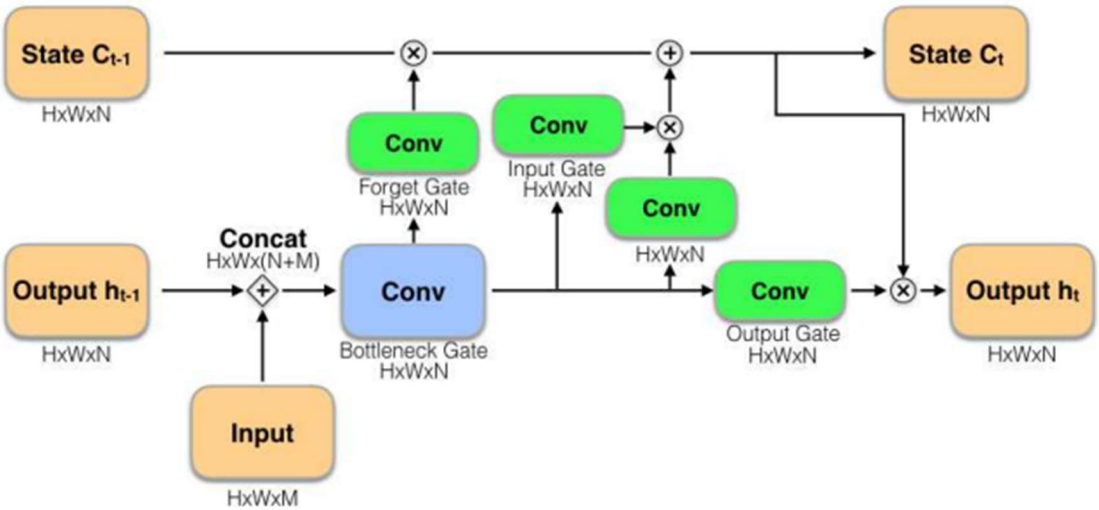

Рисунок 2.2 – Ілюстрація вузького місця LSTM

Множник розширеної ширини

LSTM за своєю суттю є дорогими через необхідність обчислення декількох шлюзів за один прямий прохід, що представляє проблему в мережах, орієнтованих на ефективність. Щоб вирішити цю проблему, вноситься безліч змін, які роблять LSTM сумісними з метою виявлення мобільних об'єктів у реальному часі.

Спочатку розглядається розмірність LSTM. Можна отримати більш точний контроль над архітектурою мережі, розширивши множник ширини каналу α.

Початковий множник ширини – це гіперпараметр, який використовується для масштабування розміру каналу кожного шару. Замість того, щоб застосовувати цей множник рівномірно до всіх шарів, вводиться три нові параметри α_{base}, α_{ssd}, α_{lstm}, які контролюють розміри каналів різних частин мережі.

Будь-який заданий шар в базовій мережі Mobilenet з N вихідними каналами модифікований так, щоб мати вихідні канали $N\alpha_{base}$, тоді як α_{ssd} застосовується до всіх карт функцій SSD, а α_{lstm} застосовується до шарів LSTM. Для цієї мережі встановлюється $\alpha_{base} = \alpha$, $\alpha_{ssd} = 0{,}5\alpha$ і $\alpha_{lstm} = 0{,}25\alpha$. Вихід кожного LSTM становить одну четверту розміру вхідного сигналу, що суттєво зменшує необхідні обчислення.

Ефективні шари вузького місця – LSTM

Нехай M і N - кількість вхідних та вихідних каналів у LSTM відповідно. Оскільки визначення згорткових LSTM різниться в різних роботах, визначається стандартний згортковий LSTM як:

$$f_t = \sigma\left((M+N)W_f^N * [x_t, h_{t-10}]\right)$$

$$i_t = \sigma\left((M+N)W_i^N * [x_t, h_{t-10}]\right)$$

$$o_t = \sigma\left((M+N)W_o^N * [x_t, h_{t-1}\ ]\right)$$

$$c_t = f_t * c_{t-1} + i_t * \varphi\left((M+N)W_f^N * [x_t, h_{t-10}]\right)$$

$$h_t = o_t * \varphi(c_t)$$

Цей LSTM приймає 3D-карти функцій x_t і h_{t-1} як вхідні дані та об'єднує їх по каналу. Він виводить карту функцій h_t і стан комірки c_t. Крім того, $jW^k \star X$ позначає згорнуту згортку по глибині з вагами W, вхідними X, j вхідними каналами і k вихідними каналами, φ позначає функцію активації, а ◦ позначає продукт Адамара.

Таблиця 2.2 – шари в одній з наших архітектур LSTM-SSD

Шар	Розмір фільтра	Крок
Conv1	3 × 3 × 3 × 32	2
Conv2	3 × 3 × 32 dw 1 × 1 × 32 × 64	1 1
Conv3	3 × 3 × 64 dw 1 × 1 × 64 × 128	2 1
Conv4	3 × 3 × 128 dw 1 × 1 × 128 × 128	1 1
Conv5	3 × 3 × 128 dw 1 × 1 × 128 × 256	2 1
Conv6	3 × 3 × 256 dw 1 × 1 × 256 × 256	1 1

Продовження таблиці 2.2

Шар	Розмір фільтра	Крок
Conv7	3 × 3 × 256 dw 1 × 1 × 256 × 512	2 1
Conv8-12	3 × 3 × 512 dw 1 × 1 × 512 × 512	1 1
Conv13	3 × 3 × 512 dw 1 × 1 × 512 × 1024	2 1
Bottleneck-LSTM	3 × 3 × 1024 dw 1 × 1 × (1024 + 256) × 256 3 × 3 × 256 dw 1 × 1 × 256 × 1024	1 1 1 1
Feature Map 1	1 × 1 × 256 × 128 1 × 1 × 256 × 128 1 × 1 × 128 × 256	1 2 1
Feature Map 2	1 × 1 × 256 × 64 3 × 3 × 64 dw 1 × 1 × 64 × 128	1 2 1
Feature Map 3	1 × 1 × 128 × 64 3 × 3 × 64 dw 1 × 1 × 64 × 128	1 2 1
Feature Map 4	1 × 1 × 128 × 32 3 × 3 × 32 dw 1 × 1 × 32 × 64	1 2 1

Використання глибоко відокремлених звивин негайно зменшує необхідні обчислення у 8 - 9 разів порівняно з попередніми визначеннями. Також обирається трохи незвичний φ (x) = ReLU (x). Незважаючи на те, що активації ReLU зазвичай не використовуються в LSTM, вважається важливим не змінювати межі карт функцій, оскільки наші LSTM розміщені між згортковими шарами.

Також можна використати той факт, що LSTM має значно менше вихідних каналів, ніж вхідні. Вноситься незначна модифікація рівнянь LSTM, спочатку обчислюючи карту функцій вузького місця з N каналами:

$$b_t = \varphi((M+N)W_b^N * [x_t, h_{t-1}])$$

Потім b_t замінює входи у всіх інших воротах. Можна називати цю нову формулювання "Вузьким місцем-LSTM". Переваги цієї модифікації подвійні. Використання карти функцій вузького місця зменшує обчислення у воротах, перевершуючи стандартні LSTM у всіх практичних сценаріях. По-друге, Bottleneck-LSTM глибший за стандартний LSTM, що більш глибокі моделі перевершують широкі та мілкі моделі.

Нехай просторові розміри кожної карти об’єктів мають значення $D_F \times D_F$, а розміри кожного поглибленого згорткового ядра - $D_K \times D_K$. Тоді обчислювальна вартість стандартного LSTM становить:

$$4(D_K^2 * (M+N) * D_F^2 + (M+N) * N * D_F^2)$$

Вартість стандартного GRU майже однакова, за винятком провідного коефіцієнта 3 замість 4. Тим часом вартість вузького місця - LSTM становить:

$$D_K^2 * (M+N) * D_F^2 + (M+N) * N * D_F^2 + 4(D_K^2 * N * D_F^2 + N^2 * D_F^2)$$

Тепер встановіть $D_K = 3$ і нехай $k = \frac{M}{N}$. Тобто, поки вихід нашого LSTM має менше ніж втричі більше каналів, ніж вхідний, це ефективніше, ніж стандартний LSTM.

Оскільки було б надзвичайно незвично, щоб цей стан не виконувався, стверджується, що вузьке місце - LSTM є більш ефективним у всіх практичних ситуаціях. Вузьке місце-LSTM також є більш ефективним, ніж GRU, коли $k > 1$. У нашій мережі $k = 4$, і вузьке місце-LSTM істотно ефективніше будь-яких альтернатив.

3. ПРОЕКТУВАННЯ СИСТЕМИ ПОРІВНЯННЯ МЕТОДІВ МОНІТОРИНГУ ВІДЕОПОТОКУ В СИСТЕМАХ МОНІТОРИНГУ І КОНТРОЛЮ

3.1.Вибір засобів реалізації

Для реалізації вище описаних алгоритмів були вибрані наступні сучасні да гнучкі в використанні інструменти. Основною мовою програмування виступає Python, тому, що він має багату екосистему для роботи у сфері машинного навчання. Ця мова програмування підтримується практично всіма відомими бібліотеками, що забезпечить сумісність з широким спектром інструментів, що в свою чергу дозволить розробляти рішення різної складності в дуже короткий час. До мінусів цієї мови програмування можна віднести те, що вона є порівняно повільною, що негативно впливає на продуктивність програм. Але і з цієї ситуації є вихід, тому що Python дозволяє виконувати код на других мовах програмування.

Основним інструментом для розробників в сфері машинного навчання є безкоштовна бібліотека під назвою TensorFlow.

Tensorflow – досить молодий фреймворк для глибокого машинного навчання, що розробляється в Google Brain, має інтеграцію з keras, та підтримку мобільних платформ. Останнім часом фреймворк розвивається ще й в сторону класичних методів, і в деяких частинах інтерфейсу вже чимось нагадує scikit-learn. До поточної версії інтерфейс змінювався активно і часто, але розробники пообіцяли заморозити зміни в API. Ми будемо розглядати тільки Python API, хоча це не єдиний варіант - також існують інтерфейси для C ++ і мобільних платформ.

Keras – бібліотека для глибокого вивчення Python. Його мінімалістичний, модульний підхід дозволяє з легкістю побудувати і запустити глибокі нейронні мережі.

3.2.Проектування інтерфейсу і функціональних модулів

"Модель" у Keras "є контейнером", "складається з шарів", які з'єднані між собою за допомогою "Вузлів".Одна «спеціалізована модель» із послідовністю складених шарів називається «послідовною моделлю». „Модель пов’язана” з „Ініціалізаторами” для ініціалізації ваги / зміщення та інших тензорів. „Модель пов’язана” із „втратами”, „метриками”, „оптимізаторами” та „зворотними викликами” для управління процесами навчання моделі та висновків. “Модель MXNet - це спеціалізована модель” для управління робочими процесами навчання та висновків для серверної бази MXNet. Ця спеціалізація необхідна, оскільки MXNet не підтримує спільні тензори для оновлень на місці, символічний оптимізатор та функціональний API низького рівня для виконання графіків. Детальніше про це в розділі «Відмінності дизайну Keras-MXNet» далі в документі. «MXNet Optimizer - це спеціалізований оптимізатор» для вибору оптимізатора Keras як оптимізатора MXNet. Ця спеціалізація необхідна, оскільки MXNet не підтримує символічний оптимізатор та низькорівневий функціональний API для виконання графіків.

“Модель MXNet пов’язана” з “BucketingModule” MXNet для управління навчальними процесами та робочими висновками для серверної бази MXNet, використовуючи API вперед, назад, оновлення API. «Символ Кераса» - це тензорна структура даних, що використовується в Keras-MXNet, яка «складається» з тензорних даних (NDArray) та символу (Символ). Ця

структура даних була введена для сумісності з концепцією Keras Tensor, яка включає як дані, так і символ. “MXNet Backend” – це колекція реалізації низькорівневого оператора Keras із використанням символьних MXNet та API NDArray.

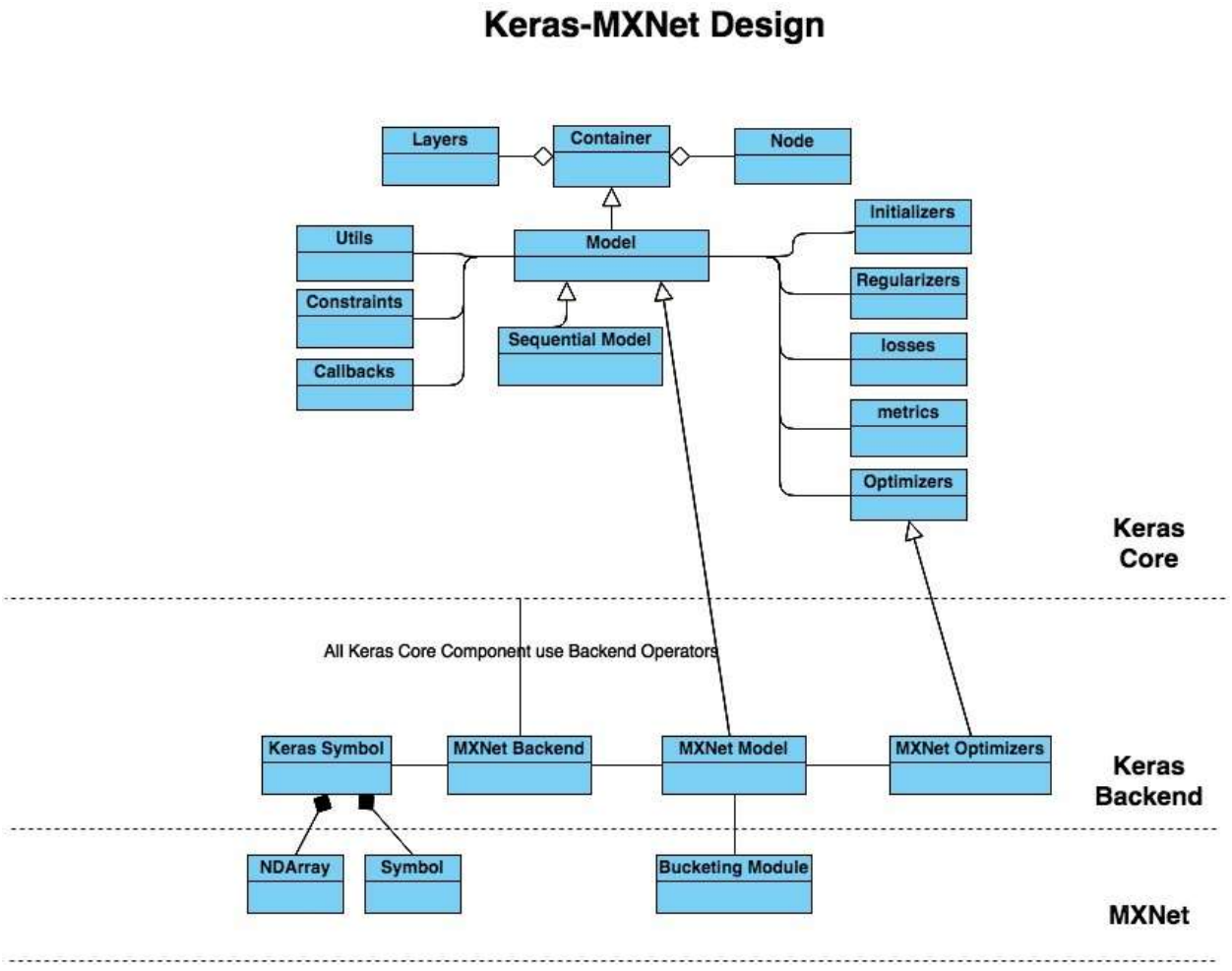

Рисунок 3.1 – Діаграма загальної моделі Keras

Символічний градієнтний оператор: дано граф, який обчислює градієнти(граф для зворотного проходу). На даний момент MXNet не підтримує це як оператор.

У Keras цей оператор використовується основним двигуном Keras для підготовки графіка до обчислювальних градієнтів (зворотний прохід). Потім

цей графік об'єднується із символічним графіком (прямий прохід). Разом двигун Keras матиме 1 символічний графік, який представляє як прохід вперед, так і назад.[3]

Символічний оптимізатор: MXNet (KVStore) не підтримує символічний оптимізатор. KVStore використовує імперативну логіку оптимізатора для градієнтних оновлень.

У Keras весь оптимізатор є символічним, тобто оптимізатори готують символічний графік для оновлення символу ваг символом градієнтів та логікою оптимізатора. Потім цей символічний графік оптимізатора об'єднується разом із уже об'єднаним символічним графіком, що дає один графік = символічний графік мережі + символічний графік градієнта + оновлення символічного графіка ваги оптимізатора

Спільні тензори (змінні) / Оперативне оновлення: MXNet не підтримує варіант використання спільних символів, що вказують на ті самі дані (ndarray). тобто 2 символи, що вказують на один і той же ndarray, який можна змінити з 2 різних символічних графіків.

У Keras ваги – це символьний тензор, який використовується в мережевому графіку (прямий прохід), а згодом використовується в символічному графіку оптимізатора для оновлення ваги з градієнтами.

Функціональний API низького рівня для виконання графіків: Модуль MXNet – це конструкція високого рівня, здатна обробляти навчання / оцінку / умовивід нейронної мережі. Він складається з оптимізатора, KVStore та інших. Однак MXNet не підтримує інтерфейс виконання графіків низького рівня.

У Keras основний движок готує один символічний графік (computation_graph + gradient_computation + optimizer_updates). Цей графік надається функціональному API низького рівня, що виконується для кожної input_batch.

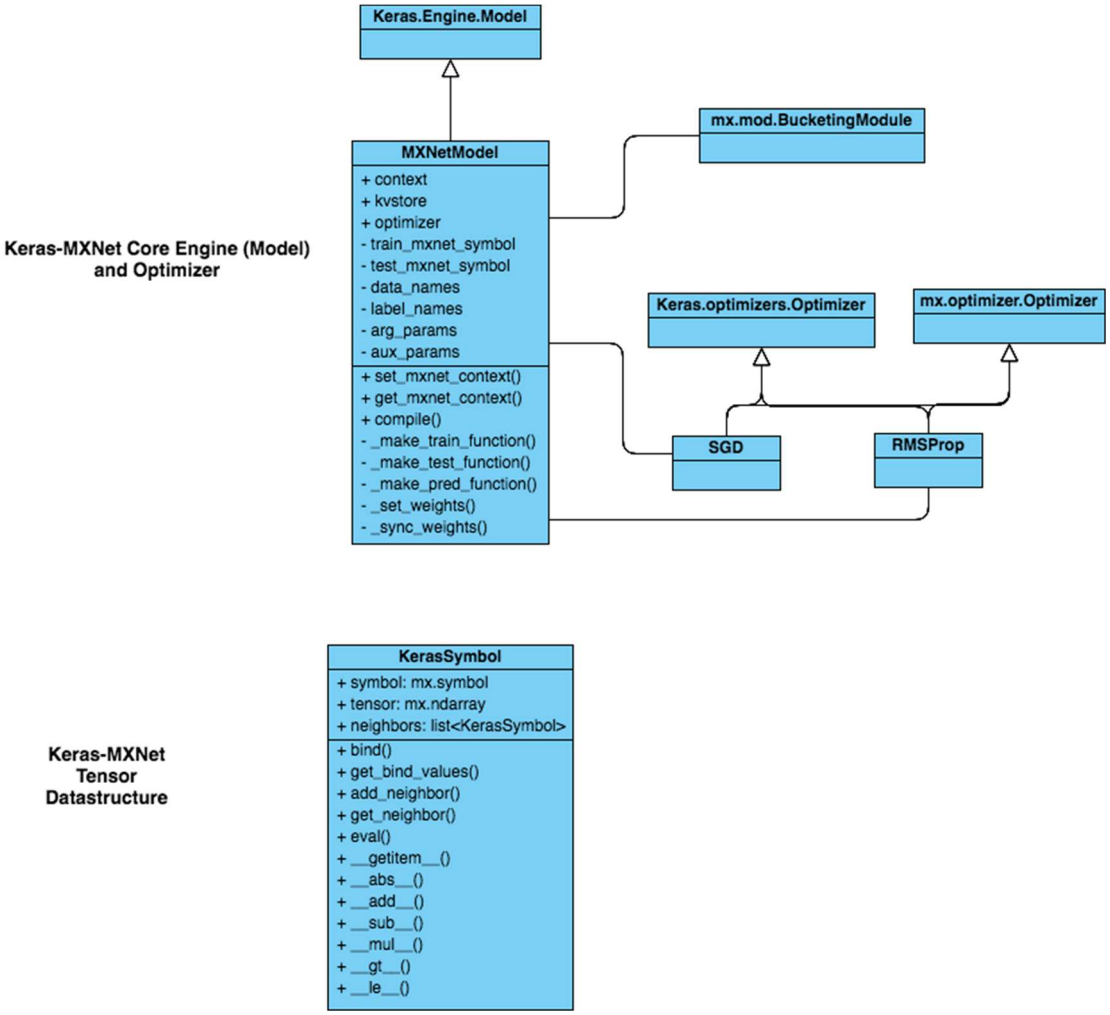

Рисунок 3.2 – Модель центральної частини

Далі представлена діаграма навчання, яка демонструє типовий порядок виконання різних команд та обчислень, взаємодію різних частин програми відносно часу, під час навчання, так званої послідовної моделі.

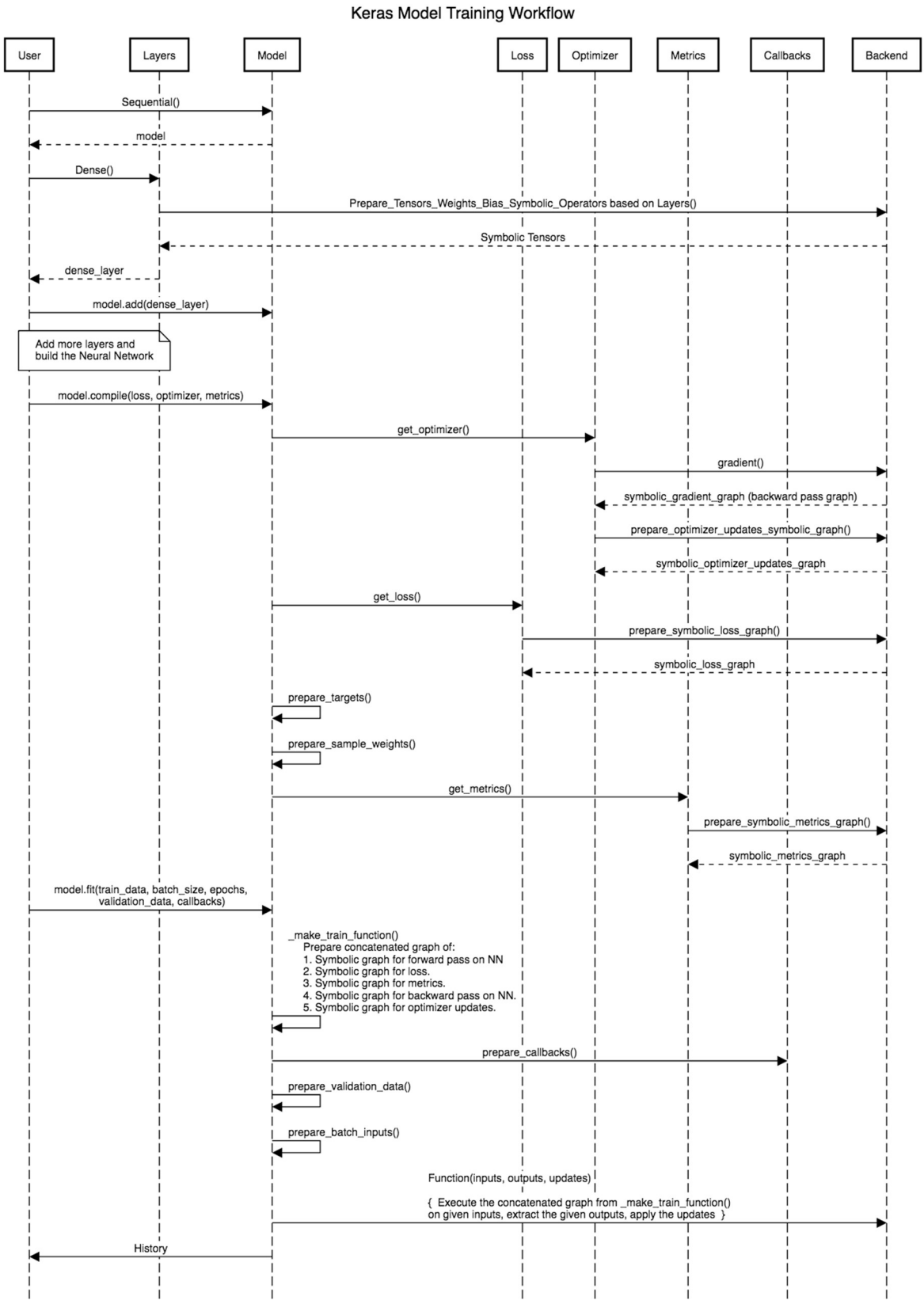

Рисунок 3.3 – Діаграма навчання

Наступною представлена діаграма активності для навчання моделі MXNet.

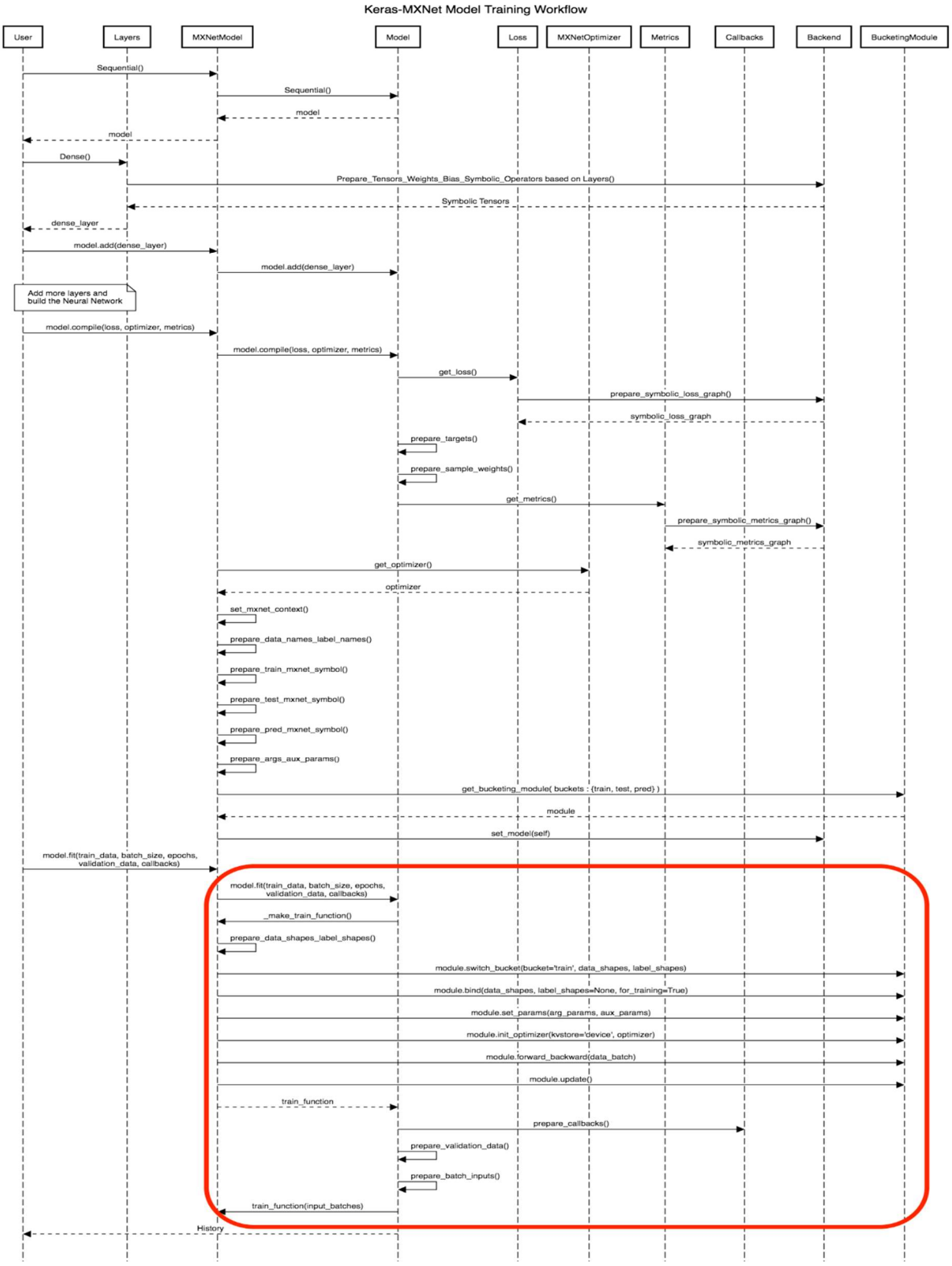

Рисунок 3.4 – Діаграма активності під час навчання моделі MXNet.

Нижче зображена UML діаграма для кожного шару який використовується в даній роботі для проектування різних моделей.

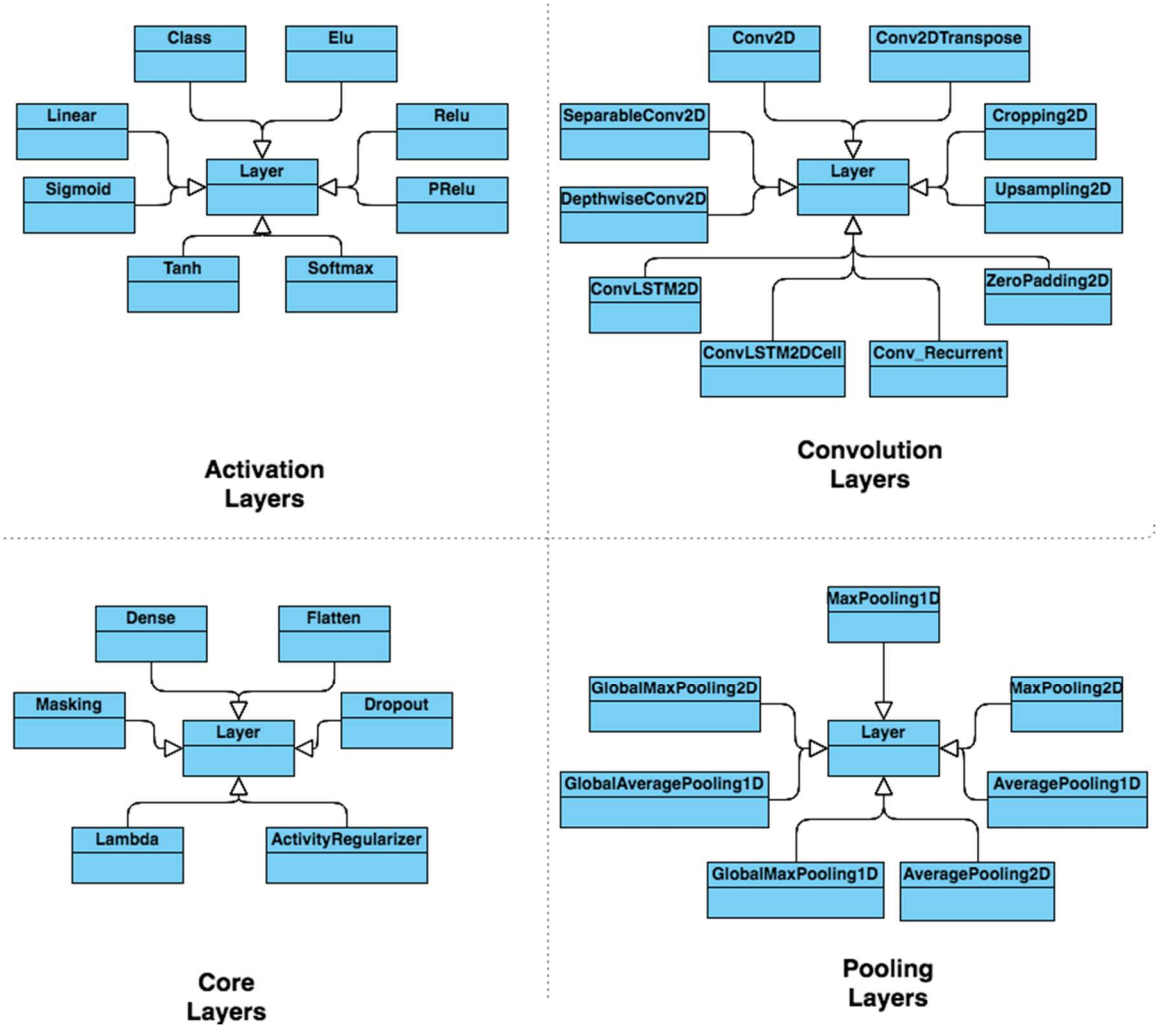

Рисунок 3.5 – Діаграма ріних шарів

4. РЕАЛІЗАЦІЯ СИСТЕМИ

4.1.Опис програми

Бібліотека компонентів, призначена для програмної реалізації нейронних мереж. Як приклад, створені компоненти реалізують дві нейромережеві парадигми: рекуррентную нейронну мережу, в нашому випадку - це мережа Хопфілда і багатошарову нейронну мережу навчаються за алгоритмом зворотного поширення помилки (back propagation).

Основним призначенням бібліотеки є інтеграція нейронних мереж в інформаційні системи, для розширення аналітичних можливостей систем. Реалізація нейронних мереж у вигляді компонентів, наявність відкритого коду дозволяє легко вбудовувати в інші програми. Об'єктно-орієнтоване виконання додає особливу гнучкість, досить переписати пару методів і ви можете отримати компонент, оптимізований під ваші завдання.

Існує три базових класу TNeuron, TLayer, TNeuralNet. Всі інші є похідними від них. На рис.1 приведена ієрархія класів, суцільними лініями показано успадкування (стрілкою вказаний нащадок), пунктирними в яких класах вони використовуються.

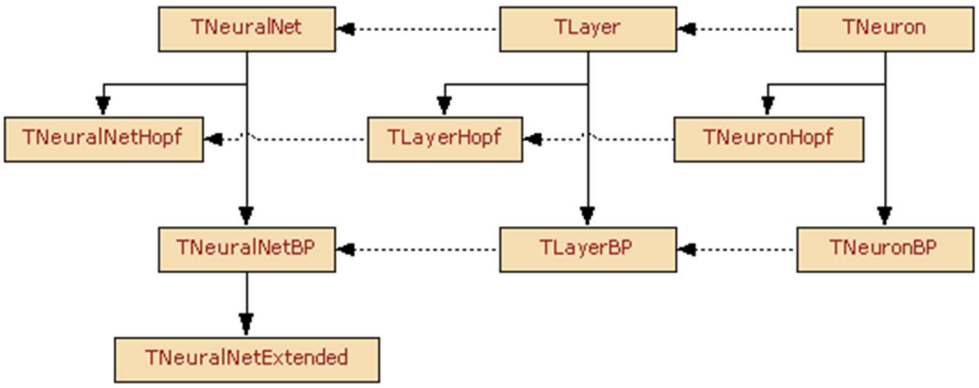

Рисунок 4.1 – Ієрархія класів

TNeuron є базовим класом для нейронів, несе всю основну функціональність, має індексовані властивість Weights, що представляє собою вагові коефіцієнти (синапси), властивість Output, яке є виходом нейрона (результатом обчислень) і акумулятор, роль якого, виконує метод ComputeOut.

TNeuronHopf, нащадок TNeuron, реалізує нейрон використовуваний в нейронної мережі Хопфілда, єдиною відмінністю від базового класу, є використання активационной функції в перекритому методі ComputeOut.

Наступним породженим класом, є TNeuronBP службовець для програмної реалізації багатошарових нейронних мереж. Абревіатура BP в імені класу не повинна вводити вас в оману, що нейрон цього типу використовується виключно в мережах учнів за алгоритмом зворотного поширення, цим, ми зайвий раз хотіли підкреслити, що в нашому випадку нейронна мережа навчається за цим алгоритмом.

Переписаний метод ComputeOut, який використовує тепер нелінійну активаційну функцію, яка реалізована у вигляді індексованого властивості процедурного типу OnActivationF. Крім того, додані дві важливі властивості, Delta – містить локальну помилку і індексовані властивість PrevUpdate – містить величину корекції вагових коефіцієнтів на попередньому кроці навчання мережі.

Основним призначенням базового класу TLayer і його нащадків TLayerHopf і TLayerBP є об'єднання нейронів в шар, для спрощення роботи з нейронами.

Компонент TNeuralNet базовий компонент для всіх видів нейронних мереж. TNeuralNet забезпечує необхідну функціональність похідних компонентів. Цей компонент підтримує методи для роботи з шарами мережі (AddLayer, DeleteLayer) і методи для маніпуляцій з вихідними даними (AddPattern, DeletePattern, ResetPatterns). Метод Init служить для побудови

нейронної мережі. Більшість методів оголошених в розділі public в базовому компоненті і його нащадків - віртуальні, що дозволяє легко перекривати їх.

Компонент TNeuralNetHopf реалізує нейронну мережу Хопфілда.

Додатково включені наступні методи: InitWeights – запам'ятовує пред'явлені зразки в матриці образів і метод Calc – обчислює вихід мережі Хопфілда.

Компонент TNeuralNetBP реалізує багатошарову нейронну мережу навчаються за алгоритмом зворотного поширення помилки.

Додатково включені наступні методи: Compute – обчислює вихід нейронної мережі, використовується після навчання мережі; TeachOffLine – навчає нейронну мережу. Компонент дозволяє в режимі design-time, у вікні Object Inspector, конструювати нейронну мережу додаючи або видаляючи шари і нейрони в мережі.

4.2.Керівництво користувача

1) Для того щоб скористатися програмою потрібно завантажити файл recogn.py, скопіювати в робочу папку, додати файл відео з назвою input.avi, та виконати відповідну команду ля запуску програми в терміналі. Якщо користувач бажає винести налаштування, тоді йому потрібно зробити наступні кроки

2) Створити файл з розширенням *.py в робочій папці, та відкрити його за допомогою тестового редактора. Далі потрібно підключити бібліотеку keras.

3) Для початку, вам потрібно зібрати шари (layers) для побудови моделей (models). Модель це (зазвичай) граф шарів. Найбільш поширеним видом моделі є стек шарів: модель tf.keras.Sequential.

4) Далі потрібно налаштувати шари. Доступно багато різновидів шарів tf.keras.layers. Більшість з них використовують загальний конструктор аргументів:

a. activation: Установка функції активації для шару. У цьому параметрі вказується ім'я вбудованої функції чи викликається об'єкт. У параметра немає значення за замовчуванням.

b. kernel_initializer і bias_initializer: Схеми ініціалізації створюють ваги шару (ядро і зрушення). У цьому параметрі може бути ім'я або викликається об'єкт. За замовчуванням використовується ініціалізатор "Glorot uniform".

c. kernel_regularizer і bias_regularizer: Схеми регуляризації додаються до ваг шару (ядро і зрушення), такі як L1 або L2 регуляризації. За замовчуванням регуляризація не встановлюється.

5) Після того як модель сконструйована, налаштуйте процес її навчання викликом методу compile. Даний метод приймає три важливі аргументи:

a. optimizer: Цей об'єкт визначає процедуру навчання. Передайте в нього екземпляри оптимізатора з модуля tf.keras.optimizers, такі як tf.keras.optimizers.Adam або tf.keras.optimizers.SGD. Якщо ви просто хочете використовувати параметри за замовчуванням, ви також можете вказати оптимізатори ключовими словами, такими як 'adam' або 'sgd'.

b. loss: Це функція яка мінімізується в процесі навчання. Серед поширених варіантів середньоквадратична помилка (mse), categorical_crossentropy, binary_crossentropy. Функції втрат вказуються на ім'я або передачею викликається об'єкта з модуля tf.keras.losses.

c. metrics: Використовуються для моніторингу навчання. Це строкові імена або викликаються об'єкти з модуля tf.keras.metrics.

Крім того, щоб бути впевненим, що модель навчається і оцінюється eagerly, перевірте що ви передали компілятору параметр run_eagerly = True

6) Наступний крок навчання на даних NumP. Для невеликих наборів даних використовуйте масиви NumPy які поміщаються в пам'ять для навчання і оцінки моделі. Модель «навчається» на тренувальних даних, використовуючи метод `fit` який приймає приймає три важливі аргументи:

a. epochs: Навчання розбите на * епохи *. Епоха це одна ітерація по всім вхідним даними (це робиться невеликими партіями).

b. batch_size: При передачі даних NumPy, модель розбиває дані на менші блоки (batches) і ітерірует по цих блоках під час навчання. Це число вказує розмір кожного блоку даних. Пам'ятайте, що останній блок може бути меншого розміру якщо загальне число записів не ділиться на розмір партії.

c. validation_data: При прототіпіруванії моделі ви хочете легко відстежувати її продуктивність на валідаційних даних. Передача з цим аргументом кортежу вхідних даних і міток дозволяє моделі відображати значення функції втрат і метрики в режимі виведення для даних, що передаються в кінці кожної епохи.

7) Використовуйте Datasets API для масштабування великих баз даних або навчання на декількох пристроях. Передайте екземпляр `tf.data.Dataset` в метод fit. Оскільки Dataset видає дані пакетами, цей шматок коду не вимагає аргументу batch_size. Датасети можуть бути також використані для валідації.

8) Методи tf.keras.Model.evaluate і tf.keras.Model.predict можуть використовувати дані NumPy і tf.data.Dataset.

9) Побудова складних моделей. Модель tf.keras.Sequential це простий стек шарів за допомогою якого можна представити довільну модель. Використовуйте Keras functional API для побудови складних топологій моделей, таких як:

a. Моделі з декількома входами,

b. Моделі з декількома виходами,

c. Моделі із загальними шарами (один і той же шар викликається кілька разів),

d. Моделі з непослідовними потоками даних (напр. Залишкові зв'язку).

e. Побудова моделі з functional API працює наступним чином:

Примірник шару викликається і повертає тензор.

Вхідні і вихідні тензори використовуються для визначення примірника tf.keras.Model – ця модель навчається точно так само як і `Sequential` модель.

10) Створіть повністю настроюється модель за допомогою сабклассінга tf.keras.Model і визначення вашого власного прямого поширення. Створіть шари в методі __init__ і встановіть їх як атрибути екземпляра класу. Визначте пряме поширення в методі call.

Сабклассінг моделі особливо корисний коли включений eager execution, оскільки він дозволяє написати пряме поширення імперативно. Примітка: якщо вам потрібно щоб ваша модель завжди виконувалася імперативно, ви можете встановити dynamic = True коли викликаєте конструктор super. Ключовий момент: Використовуйте правильний API для роботи. Хоч сабклассінг моделі забезпечує гнучкість, за неї доводиться платити більшою складністю і великими можливостями для призначених для користувача помилок. Якщо це можливо вибирайте functional API.

11) Власні шари користувача. Створіть користувацький шар сабклассінгом tf.keras.layers.Layer і реалізацією таких методів:

a. __init__: Опціонально визначте підшари які будуть використовуватися в цьому шарі.

b. * Build: Створіть ваги шару. Додайте ваги за допомогою методу add_weight

c. call: Визначте пряме поширення.

d. Опціонально, шар може бути серіалізовані реалізацією методу get_config і методу класу from_config.

12) Колбек це об'єкт переданий моделі щоб кастомизировать і розширити її поведінку під час навчання. Ви можете написати свій користувальницький колбек або використовувати вбудований tf.keras.callbacks який включає в себе:

a. tf.keras.callbacks.ModelCheckpoint: Збереження контрольних точок моделі за регулярні інтервали.

b. tf.keras.callbacks.LearningRateScheduler: Динамічний зміна кроку навчання.

c. tf.keras.callbacks.EarlyStopping: Зупинка навчання в тому випадку коли результат при валідації перестає поліпшуватися.

d. tf.keras.callbacks.TensorBoard: Моніторинг поведінки моделі за допомогою TensorBoard

13) Збереження тільки конфігурації моделі. Конфігурація моделі може бути збережена – це серіалізуются архітектуру моделі без всяких ваг. Збережена конфігурація може відновити і форматувати ту ж модель, навіть без коду визначає вихідну модель. Keras підтримує формати сериализации JSON і YAML

14) Eager execution – це імперативне програмування середовище яке виконує операції негайно. Це не потрібно для Keras, але підтримується tf.keras і корисно для перевірки вашої програми і налагодження. Всі будують моделі API `tf.keras` сумісні eager execution. І хоча можуть бути використані `Sequential` і functional API, eager execution особливо корисно при сабклассіроваиі моделі і побудові користувальницьких шарів - ці API вимагають від вас написання прямого поширення у вигляді коду (замість API які створюють моделі шляхом складання існуючих шарів).

15) tf.keras моделі можна запускати на безлічі GPU з використанням tf.distribute.Strategy. Цей API забезпечує розподілене навчання на декількох GPU практично без змін в існуючому коді.

На даний момент, tf.distribute.MirroredStrategy єдина підтримувана стратегія розподілу. MirroredStrategy виконує реплікацію в графах з синхронним навчанням використовуючи all-reduce на одній машині. Для використання `distribute.Strategy`, вкладіть інсталяцію оптимізатора, конструкцію і компіляцію моделі в` Strategy` `.scope ()`, потім навчіть модель.

Наступний приклад розподіляє tf.keras.Model між безліччю GPU на одній машині.

4.3. Технологія тестування програмного забезпечення

Відповідно до даного методу створюється необхідна кількість незалежних тестів, у яких вхідні дані генеруються випадковим чином. Недоліком даного методу є загальна кількість тестів, які необхідно згенерувати відповідно до вимог надійності, до того ж забезпечивши незалежність цих тестів. Так, наприклад, для забезпечення надійності програмного забезпечення з імовірністю відмови не більше 10-5 і з помилкою не більше 5%, потрібно згенерувати 299 572 тестів. З метою скорочення кількості необхідних тестів Майєрсом було запропоновано розглядати розбиття безлічі вихідних даних на еквівалентні класи.[4]

Відповідно до даного методу створюється необхідна кількість незалежних тестів, у яких вхідні дані генеруються випадковим чином. Недоліком даного методу є загальна кількість тестів, які необхідно згенерувати відповідно до вимог надійності, до того ж забезпечивши незалежність цих тестів. Так, наприклад, для забезпечення надійності

програмного забезпечення з імовірністю відмови не більше 10-5 і з помилкою не більше 5%, потрібно згенерувати 299 572 тестів. З метою скорочення кількості необхідних тестів Майєрсом було запропоновано розглядати розбиття безлічі вихідних даних на еквівалентні класи

Відповідно до даної методики необхідно розбити множину значень вхідних даних на кінцеве число підмножин (які будуть називатися класами еквівалентності), щоб кожний тест, що є представником певного класу, був еквівалентним будь-якому іншому тесту цього класу. Два тести є еквівалентними, якщо вони виявляють ті самі помилки.

Проектування тестів за методом класів еквівалентності проводиться у два етапи: – виділення за специфікацією класів еквівалентності; – побудова множини тестів. На першому етапі відбувається вибір зі специфікації кожної вхідної умови та розбиття її на дві або більше групи, що відповідають так званим — правильним класам еквівалентності(ПКЕ) та — неправильним класам еквівалентності(НКЕ), тобто множинам допустимих для програми й недопустимих значень вхідних даних. Цей процес залежить від вигляду вхідної умови. Наприклад: якщо вхідна умова описує множину($|x| <= 0.5$), то визначається один ПКЕ($-0.5 <= x <= 0.5$) і два НКЕ($x < -0.5; x > 0.5$). На другому етапі методу класів еквівалентності виділені класи використовуються для побудови тестів. Для НКЕ тести проектуються таким чином, що кожен тест покриває один і тільки один НКЕ, доки всі НКЕ не будуть покриті. Метод класів еквівалентності дозволяє значно скоротити кількість тестів у порівнянні з методом випадкового тестування, але також має свої недоліки. Основний з них - це складність виділення класів еквівалентності, особливо НКЕ, а також можливий пропуск певних типів високоефективних тестів (тобто тестів, що характеризуються великою ймовірністю виявлення помилок). Так, наприклад, мінімальні й максимальні припустимі значення вхідних параметрів дозволяють виявити більшість помилок, пов'язаних з відповідностями й

переповненнями типів даних. Для вирішення даної проблеми був запропонований метод аналізу граничних умов.

Під граничними умовами розуміють ситуації, що виникають безпосередньо на границі певної вхідної або вихідної умови, вище або нижче її. Метод аналізу граничних умов відрізняється від методу класів еквівалентності наступним: - вибір будь-якого представника класу еквівалентності здійснюється таким чином, щоб перевірити тестом кожну границю цього класу; - при побудові тестів розглядаються не тільки вхідні умови, але й вихідні (тобто певні специфіковані обмеження на значення вхідних даних). Загальні правила методу аналізу граничних умов: побудувати тести для границь множини допустимих значень вхідних даних і тести з недопустимими значеннями, що відповідають незначному виходу за межі цієї множини. Наприклад, для множини $[-1.0; 1.0]$ будуються тести $-1.0; 1.0; -1.001; 1.001;$ Зауважимо, що на практиці з метою локалізації несправностей створюють також тести, що відповідають допустимим значенням, тобто є внутрішніми для множини та ті, що незначно відхиляються від граничних значень: $-1.0;\ 1.0;\ -1.001;\ 1.001; 0.999; -0.999$ Якщо множина допустимих значень вхідних даних дискретна, то будуються тести для мінімального й максимального значення вхідних умов і тести для значень, більших або менших цих величин. Наприклад, якщо вхідний файл може містити від 1 до 255 записів, то вибираються тести для порожнього файлу та файлу, що містить 1, 254, 255 і 256 записів. 1) використовувати перше правило для кожної вихідної умови; 11 2) якщо вхідні й вихідні дані програми являють собою впорядковану множину (послідовний файл, лінійний список та ін.), то при тестуванні зосередити увагу на першому й останньому елементі множини; 3) повторити процедуру для всіх знайдених граничних умов. Аналіз граничних умов - один з найбільш корисних методів проектування тестів. Але він часто

виявляється неефективним через те, що граничні умови іноді ледь вловимі, а їхнє виявлення досить важко.

Структурне тестування, або тестування «білого ящика», - це методика аналізу вихідного коду програми. Існує три різновиди структурного тестування: тестування на основі потоку керування програми, на основі потоку даних та мутаційне тестування. При використанні першого типу тестується логіка програми, що представлена у вигляді графа керування: вершинами є оператори, а гілками - переходи між ними. При тестування на основі потоку даних увага приділяється взаємозв'язкам між змінними. Виділяються вершини, у яких змінна ініціалізується та в яких використовується, і вивчаються переходи й взаємозв'язки між такими вершинами.

Мутаційне тестування полягає у внесенні несправностей у вихідний код програми та порівняння роботи вихідної програми та програми мутанта. Оскільки здійснити вичерпне структурне тестування вкрай важко, необхідно вибрати такі критерії його повноти, які допускали б їхню просту перевірку й полегшували б цілеспрямований підбір тестів. Зупинимося на цьому докладніше.

Для тестування на основі потоку керування існує ряд критеріїв: покриття операторів, покриття рішень, шляхів, циклів, умов і т.д. Найпростішим є критерій покриття операторів.

Критерій покриття операторів (C0) – кожен оператор програми повинен буди виконаний (покритий) хоча б один раз. Цей критерій є найбільш слабким з використовуваних у структурному тестуванні, тому що проходження всіх операторів не гарантує перевірку правильності послідовності попарних переходів між ними.

Критерій покриття рішень(C1) – кожна гілка алгоритму (кожний перехід між вершинами) має бути пройдена (виконана) хоча б один раз. Виконання даного критерію, у загальному випадку, забезпечує й покриття операторів,

проте критерій С1 не є ідеальним. Так, наприклад, він не забезпечує перевірку правильності обробки операторів логічних переходів та циклів.

Критерій покриття шляхів (С∞) – кожен шлях в алгоритмі, де шлях – це послідовність вершин $(n_{start}, n_1, \ldots, n_m, n_{final})$ має бути протестований хоча б один раз. Два шляхи вважаються ідентичними, якщо послідовності вершин ідентичні. Це найбільш повний критерій, проте його реалізація ускладнена через величезну кількість необхідних тестів. Так, наприклад, проблемою є тестування циклічних структур, в силу необхідності їх багаторазового повторення. Для спрощення даної процедури було запропоновано два наступних критерії:

Граничне тестування циклу – відповідно до даного критерію має бути виконаний вхід у кожний цикл (проте виконання ітерацій не вимагається).

Внутрішнє тестування циклу – відповідно до даного критерію має бути виконаний вхід у кожний цикл і як мінімум одна ітерація. Вочевидь, що виконання другого критерію забезпечує виконання критерію граничного тестування циклів, але разом з тим, вимагає більшої кількості тестів. Вище було зазначено, що часто виникають проблеми з тестуванням логічних конструкцій розгалуження. Для цього були запропоновані критерії покриття умов. Використання даних критеріїв може дати підбір тестів, що забезпечують перехід у програмі, що пропускається при використанні критерію С1.

Критерій покриття умов/рішень – крім вимог, представлених в попередньому пункті додається умова - кожна гілка алгоритму (кожний перехід) повинна бути пройдена (виконана) хоча б один раз. На практиці найчастіше виникає ситуація, при якій правильність функціонування атомарних умов не гарантує істинності загального виразу умови. З метою забезпечення цієї умови був уведений наступний критерій.

Модифікований критерій покриття умов/рішень – кожна атомарна умова, що має вплив на істинність загального виразу-умови, має бути

протестована, при цьому тести повинні бути незалежні від інших часткових умов. Повна перевірка правильності функціонування операторів умови може бути здійснена лише шляхом перебору всіх варіантів комбінацій атомарних умов, що входять у загальний предикатний вираз, що знайшло відображення у наступному критерії.

На сьогоднішній день, практика створення сучасного ПЗ полягає в тому, що акцент ставиться на конструюванні складних компонентів з окремих, вже розроблених модулів. Це називається компонентнобазованим підходом. Можна виділити чотири основних групи компонентів: - Комерційні компоненти інших постачальників; - Внутрішні розроблені компоненти для інших проектів; - Внутрішні розроблені компоненти, які були оновлені для повторного використання; - Нові сконструйовані компоненти. Компонентно-базоване ПЗ характеризується: - різною природою програмних компонентів; - повторним використанням компонентів; - використанням підмножини наданої функціональності; - недоступністю вихідного коду. Даний підхід дозволяє зменшити час і витрати на програмування, однак різна природа програмних компонентів, відсутність вихідного коду, використання підмножини наданої функціональності й складності інтеграції являють собою основні проблеми для тестування компонентно-базованого програмного забезпечення, особливо для інтеграційного тестування. Відомі катастрофи Therac25, Ariane 5, у яких помилки ПЗ були пов'язані саме з повторним використанням попередньо «працюючих» компонентів, тобто було неякісно здійснене інтеграційне тестування.

Як було зазначено вище, для інтеграційного тестування, потрібно не тільки згенерувати тестові приклади, необхідно також розробити нові критерії й метрики, які дозволять кількісно оцінювати якість цих тестових прикладів. Основні визначення Розглянемо наступні терміни й UML-базовані позначення: компонент, інтерфейс, поведінка компонента та взаємодія

компонентів. Компонент (Component) Компонент – це фрагмент функціональності, який забезпечує доступ до сервісу за допомогою інтерфейсів та постачається незалежно. Інтерфейс (Interface) Інтерфейси являють собою точки доступу до компонентів, через які клієнтський компонент може запитувати сервіси, надавані іншим компонентом і оголошені в його інтерфейсі. Кожний інтерфейс може містити безліч операцій, а кожна операція реалізує заданий сервіс. Інтерфейси складаються із двох частин: синтаксичної та семантичної. Синтаксична частина містить оголошення сервісів, наданих компонентом або ним використовуваних. Вона може бути розділена на дві частини: - Запропонований інтерфейс (Providedinterface) (сервіси, які компонент надає); 45 – Необхідний інтерфейс (Requiredinterface) (сервіси, необхідні для функціональності компонента). Позначимо: Component - C, Provided Interface - PI(c), Required Interface – RI(c). Семантична частина містить опис поведінки компонента, надаючи інформацію про правила, згідно яким оголошений сервіс може бути використаний або викликаний. Активізація інтерфейсу (звертання до інтерфейсу) може бути здійснена іншим інтерфейсом, за допомогою виключення або явного користувальницького виклику (наприклад, натисканням кнопки). Деякі виключення й користувальницькі дії, які вимагають виклику компонентів, не є частинами будь-якого іншого інтерфейсу. Щоб уніфікувати подання, об'єднаємо такі події у віртуальний інтерфейс, що буде враховувати всі такі можливі події. Поведінка компонента Поведінка компонента може бути представлена за допомогою UML діаграм станів. Діаграма станів описує динаміку поведінки компонента у відповідь на зовнішні стимули (вплив). Діаграма станів складається зі станів та переходів. Стан моделює ситуацію, під час якої спостерігається інваріантність (стабільність, незмінюваність) поточних умов. Позначимо: States (Component) – множина всіх можливих станів компонента. Перехід - це спрямоване відношення між вихідною вершиною й цільовою.

Діаграми станів представляють можливий порядок змін 46 станів, де кожний перехід являє собою реалізацію операції або варіанта використання (use case). Переходи визначені наступною п'ятіркою: стан-джерело, станприймач, тригер, дія і умова Transition:= (Source, Target, Trigger, Effect, Guard) Стан-Джерело – визначає первісну вершину переходу. Стан-Приймач – визначає цільову вершину, що досягається по завершенні переходу. Тригер – обумовлює сервіс, що запускає перехід. Дія – обумовлює необов'язкову діяльність, що виконується при запуску. Умова - являє собою обмеження на запуск переходу. Умова перевіряється в той момент, коли перехід надходить на вхід кінцевого автомата. Якщо умова істинно на цей момент, то перехід здійснюється, у противному випадку - немає. Source, Target (States(Component) Trigger (ProvidedInterface(Component) Effect (RequiredInterface(Component) Взаємодія компонентів Система складається з різних компонентів, які взаємодіють між собою. Всі елементи описані згідно із системою позначень, представленою вище. Поведінка системи визначається взаємодією компонентів. Взаємодія компонентів може бути описана, використовуючи діаграми взаємодії UML. Вони представляють взаємозв'язки між компонентами як серію послідовних повідомлень. Діаграми взаємодії 47 описують як статичну структуру, так і динамічне поводження системи. Повідомлення - це іменовані елементи, які визначають задані різновиди зв'язків у взаємодії. В якості зв'язку (комунікації) може розглядатися передача сигналу, виклик операції, створення або знищення об'єкта. Повідомлення задають не тільки різновид зв'язків, але й специфікують джерело й приймач. Повідомлення мають наступний синтаксис: Message:= (Name, AssignmentList) AssignmentList:= ( ) $\cup$ (d1,..., dn), n $\in$ N , di = Assignment(pi), де ріпараметр, n – кількість параметрів. Повідомлення в діаграмі взаємодії впорядковані в послідовності й пронумеровані.

5. ЕКСПРЕМЕНТАЛЬНО-ПРАКТИЧНА ЧАСТИНА

На відміну від наборів даних із зображеннями, великомасштабні відеонабори набагато важче збирати та коментувати. Підхід оцінюється на двох останніх наборах даних: Cityscapes для семантичної сегментації та ImageNet VID для виявлення об’єктів.

5.1.Налаштування експерименту

Cityscapes — це набір даних для міських сцен та транспортних засобів. Він містить фрагменти вуличних сцен, зібраних із 50 різних міст, із частотою кадрів 17 кадрів в секунду.

Поїзд, перевірка та тестові набори містять 2975, 500 та 1525 фрагментів відповідно. Кожен фрагмент має 30 кадрів, де 20-й кадр коментується мітками наземної істини на рівні пікселів для семантичної сегментації. Існує 30 семантичних категорій. Відповідно до протоколу, навчання проводиться на поїзному комплекті, а оцінка виконується на валідаційному комплекті. Точність сегментації вимірюється середньою оцінкою на рівні пікселів перетину над об'єднанням (mIoU).

Як під час навчання, так і для умовиводу, розмір зображення зменшується, щоб мати коротші сторони 1024 та 512 пікселів для функціональної мережі та потокової мережі відповідно. Під час навчання SGD 20 ітерацій виконуються на 8 графічних процесорах (кожен графічний процесор вміщує одну міні-партію, таким чином ефективний розмір партії × 8), де швидкість навчання становить 10^{-3} та 10^{-4} для перших 15K та останніх ітерацій 5K відповідно.[5]

ImageNet VID. Він призначений для виявлення об’єктів у відео. Набори навчання, перевірки та тестування містять відповідно 3862, 555 та 937

повністю анотованих фрагментів відео. Частота кадрів для більшості фрагментів становить 25 або 30 кадрів в секунду. Існує 30 категорій об'єктів, які є підмножиною категорій у наборі зображень ImageNet DET2. Оцінка виконується на наборі перевірки, використовуючи стандартну метрику середньої точності (mAP).

Як під час навчання, так і для умовиводу, розмір зображення зменшується, щоб мати коротші сторони - 600 пікселів та 300 пікселів для функціональної мережі та потокової мережі відповідно. Під час навчання SGD ітерації 60K виконуються на 8 графічних процесорах, де швидкість навчання становить 10^{-3} та 10^{-4} для перших 40K та останніх 20K ітерацій відповідно.

5.2.Методологія оцінки та результати

Глибокий потік функцій є гнучким і дозволяє різні варіанти дизайну. Для уточнення фіксується значення за замовчуванням протягом експериментів, якщо не вказано інше. Для функціональної мережі N_{feat} модель ResNet-101 є типовою. Для потокової мережі F за замовчуванням використовується FlowNet. Тривалість ключового кадру l дорівнює 5 для сегментації Cityscapes та 10 для виявлення VID ImageNet за замовчуванням на основі різної частоти кадрів відео в наборах даних.

Для кожного фрагмента обчислюється l пар зображень, (k, i), k = i - l + 1, ..., i, для кожного кадру i з анотацією основних істин. Оцінка часу проводиться на робочій станції з графічним процесором NVIDIA GTX950 та процесором Ryzen 5 3600.

Таблиця 5.1 – Опис DFF, SFF та Frame

Метод	Навчання мережі розпізнавання зображень N	Навчання проточної мережі F
Frame (oracle baseline)	тренується на одинарних кадрах	не використовується потокова мережа
SFF-slow	те саме, що і Frame	SIFT-Flow (з найкращими параметрами), навчання не проводиться
SFF-fast	те саме, що і Frame	SIFT-Flow (параметри за замовчуванням), навчання не проводиться
DFF	тренується на парі кадрів	init. on Flying Chairs , тонко налаштований
DFF fix N	те саме, що Frame, потім виправлено	те саме, що і DFF
DFF fix F	те саме, що і DFF	init. on Flying Chairs, потім виправлено
DFF separate	те саме, що і Frame	init. on Flying Chairs

Таблиця 5.2 – Порівняння точності та часу роботи

Методи	Cityscapes (l = 5)		ImageNet VID (l = 10)	
	mIoU(%)	час виконання (fps)	mAP(%)	час виконання (fps)
Frame	71,1	1,52	73,9	4,05
SFF-slow	67,8	0,08	70,7	0,26
SFF-fast	67,3	0,95	69,7	3,04
DFF	69,2	5,60	73,1	20,25
DFF fix N	68,8	5,60	72,3	20,25
DFF fix F	67,0	5,60	68,8	20,25
DFF separate	66,9	5,60	67,4	20,25

Перевірка архітектури DFF. Ми порівняли DFF з декількома базовими лініями та варіантами.

- Frame: тренувати N на одинарних кадрах
- SFF: використовувати заздалегідь розрахований потік великих переміщень. SFF-швидкий і SFF-повільний, приймають різні параметри.
- DFF: запропонований підхід, N та F навчаються наскрізно. Кілька варіантів включають виправлення DFF N (виправлення N на тренуванні), DFF виправлення F (виправлення F на тренуванні) та DFF окремі (N та F навчаються окремо).

Зауважимо, що базовий фрейм досить сильний, щоб служити еталоном для порівняння. Реалізація нагадує найсучасніший DeepLab для семантичної сегментації та R-FCN для виявлення об’єктів. У DeepLab показник mIoU

становить 69,2% із великою моделлю поля зору DeepLab із використанням ResNet-101 на наборі даних перевірки Cityscapes. Базові лінії Frame трохи вищі на 71,1%, на основі тієї ж моделі ResNet.

Для виявлення об'єкта, базовий рівень фрейму має mAP 73,9% із використанням R-FCN та ResNet-101. Для порівняння, порівняльний показник mAP у 73,8% повідомляється, комбінуючи детектори об’єктів CRAFT та DeepID-net, навчені за даними ImageNet, використовуючи як VGG-16, так і GoogleNet- v2 моделі, з різними прийомами (багато масштабне навчання / тестування, додавання контекстної інформації, ансамбль моделей).

SFF-fast має розумний час роботи, але точність значно знижується. SFF-slow використовує найкращі параметри для оцінки потоку. Це набагато повільніше. Його точність трохи покращена, але все ще погана. Це вказує на те, що готовий потік може бути недостатнім.

Запропонований підхід DFF має найкращі загальні показники. Його точність трохи нижча, ніж у Frame, і вона в 3,7 та 5,0 разів швидша для сегментації та виявлення відповідно. Як і слід було очікувати, три варіанти без використання спільних тренувань мають гіршу точність. Особливо, падіння точності фіксацією F є значним. Це вказує на необхідність спільного наскрізного навчання (особливо потокового).

Протестувавши інший варіант DFF із видаленою функцією шкали S. Падає точність як для сегментації, так і для виявлення (менше одного відсотка). Це показує, що масштабована модуляція функцій трохи корисна.

Точність прискорення компромісу. Досліджуючи компроміс, змінюючи потокову мережу F, функціональну мережу N_{feat} та тривалість ключового кадру l. Оскільки набори даних Cityscapes та ImageNet VID мають різну частоту кадрів, тестується l = 1, 2, ..., 10 для сегментації та l = 1, 2, ..., 20 для виявлення.

Загалом, DFF досягає значного прискорення при гідному падінні точності. Він плавно торгується точністю до швидкості та гнучко відповідає різним потребам програми. Наприклад, при виявленні він покращує 4,05 кадрів в секунду кадру ResNet-101 Frame до 41,26 кадрів в секунду ResNet-101 + FlowNet Inception. Швидкість у 10 разів швидша за рахунок помірного зниження точності з 73,9% до 69,5%. При сегментації це покращує 2,24 кадрів в секунду ResNet-50 Frame до 17,48 кадрів в секунду ResNet-50 FlowNet Inception, за рахунок зниження точності з 69,7% до 62,4%.

Таблиця 5.3 - Результати використання різних точок розбиття для N_{task}.

№ шару в N_{task}	Cityscapes (l = 5)		ImageNet VID (l = 10)	
	mIoU(%)	час виконання (fps)	mAP(%)	час виконання (fps)
21	69,1	2,87	73,2	7,23
12	69,1	3,14	73,3	8,04
5	69,2	3,89	73,2	9,99
1 (за замовчуванням)	69,2	5,60	73,1	20,25
0	69,5	5,61	72,7	20,40

Який потік F слід використовувати? Вигідним є найменший початковий рівень FlowNet. Здебільшого він швидший за своїх двох аналогів на одному рівні точності.

Яку функцію N_{feat} нам слід використовувати? У зоні високої точності точна модель ResNet-101 явно краща, ніж ResNet-50. У високошвидкісній зоні висновки щодо двох завдань різні. Для виявлення ResNet-101 все ще вигідний.

Для сегментації криві продуктивності перетинаються приблизно з 6,35 точки в секунду. Для більшої швидкості ResNet50 стає кращим, ніж ResNet-101.

Різні висновки можуть бути частково пов’язані з різною частотою кадрів відео, ступенем динаміки двох наборів даних. Набір даних Cityscapes має не тільки низьку частоту кадрів 17 кадрів в секунду, але і більш швидку динаміку. Важко використовувати часову надмірність для тривалого поширення. Для досягнення такої ж високої швидкості ResNet-101 потребує більшої довжини кадру l, ніж ResNet-50. Це, в свою чергу, значно збільшує складність навчання.

Розділений пункт N_{task}. Де можна розділити N_{task} на N?

За замовчуванням N_{task} зберігає один шар із вагою навчання (1 × 1 конв. На 1024-денних функціональних картах). До цього є 3 × 3 conv-шар, який зменшує розмір до 1024. Перед цим є серія «Вузького місця» в ResNet, кожна з 3 шарів.

Точка розколу повертається назад, щоб зробити різні N_{task} з 5, 12 і 21 шарами відповідно. Той, що має 5 шарів, додає шар зменшення розміру та один вузький вузол (conv5c). Той, що має 12 шарів, додає ще дві одиниці (conv5a та conv5b) на початку conv5. Той, що має 21 шар, додає ще три одиниці в conv4. Переміщується єдиний шар за замовчуванням N_{task} в N_{feat}, залишаючи N_{task} з шаром 0 (з вагами, що вивчаються). Це еквівалентно безпосередньому розповсюдженню без параметричних оціночних карт як у семантичній сегментації, так і в виявленні об’єктів.

Загалом, варіація точності досить мала, щоб нею нехтувати. Швидкість стає нижчою, коли N_{task} має більше шарів. Використання 0-шару здебільшого еквівалентно використанню 1-го шару як точністю, так і швидкістю. Вибирається 1 шар за замовчуванням, оскільки це залишає деякі настроюванні параметри після поширення функції, що може бути більш загальним.

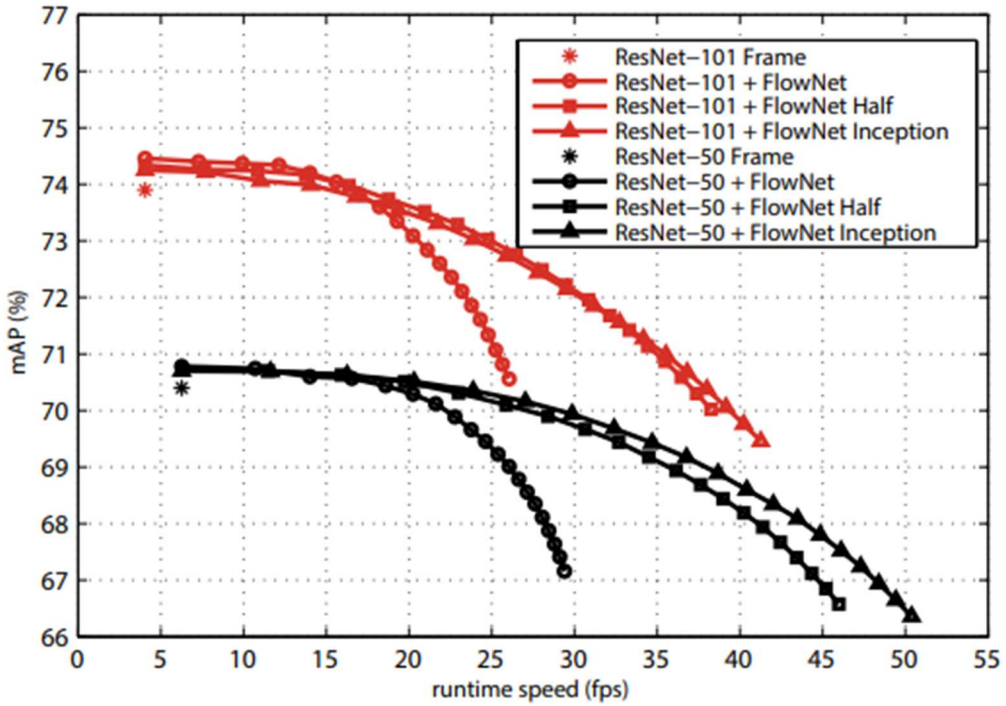

Рисунок 5.1 – Перший результат швидкості виконання

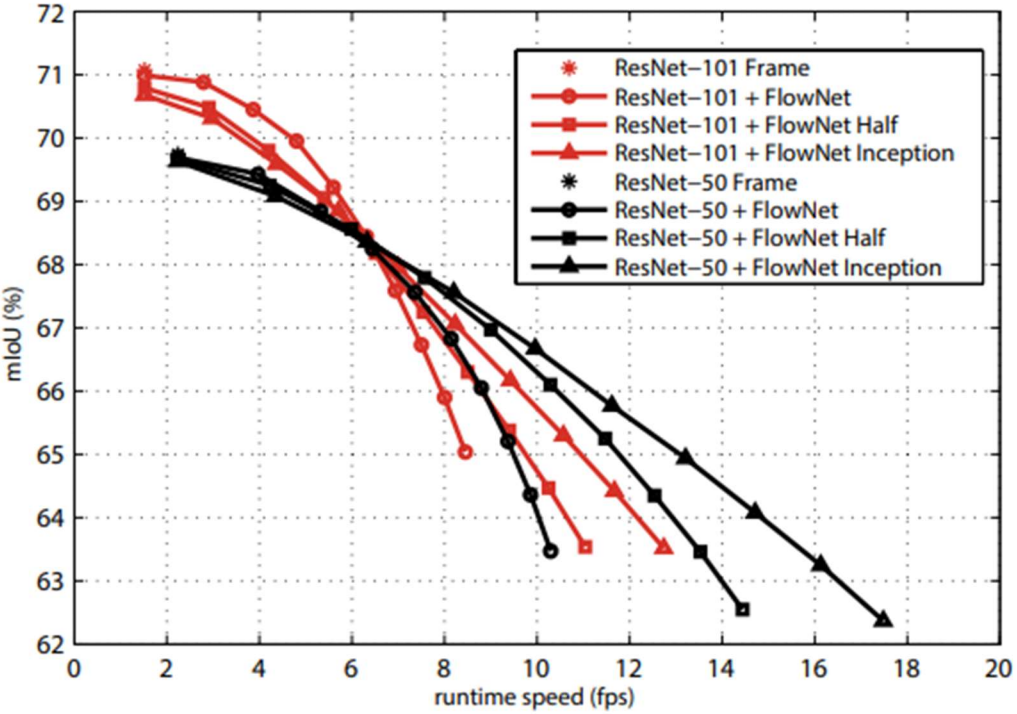

Рисунок 5.2 – Другий результат швидкості виконання

5.3.Результати

Цей метод може отримати додаткові переваги завдяки вдосконаленням в оцінці потоку та плануванні ключових кадрів. У цій роботі застосовується FlowNet.

Проектуванню швидшої та точнішої потокової мережі, безумовно, буде приділено більше уваги в майбутньому. Для планування ключових кадрів хороший планувальник цілком може значно покращити як швидкість, так і точність. І ця проблема, безумовно, варта подальших досліджень.

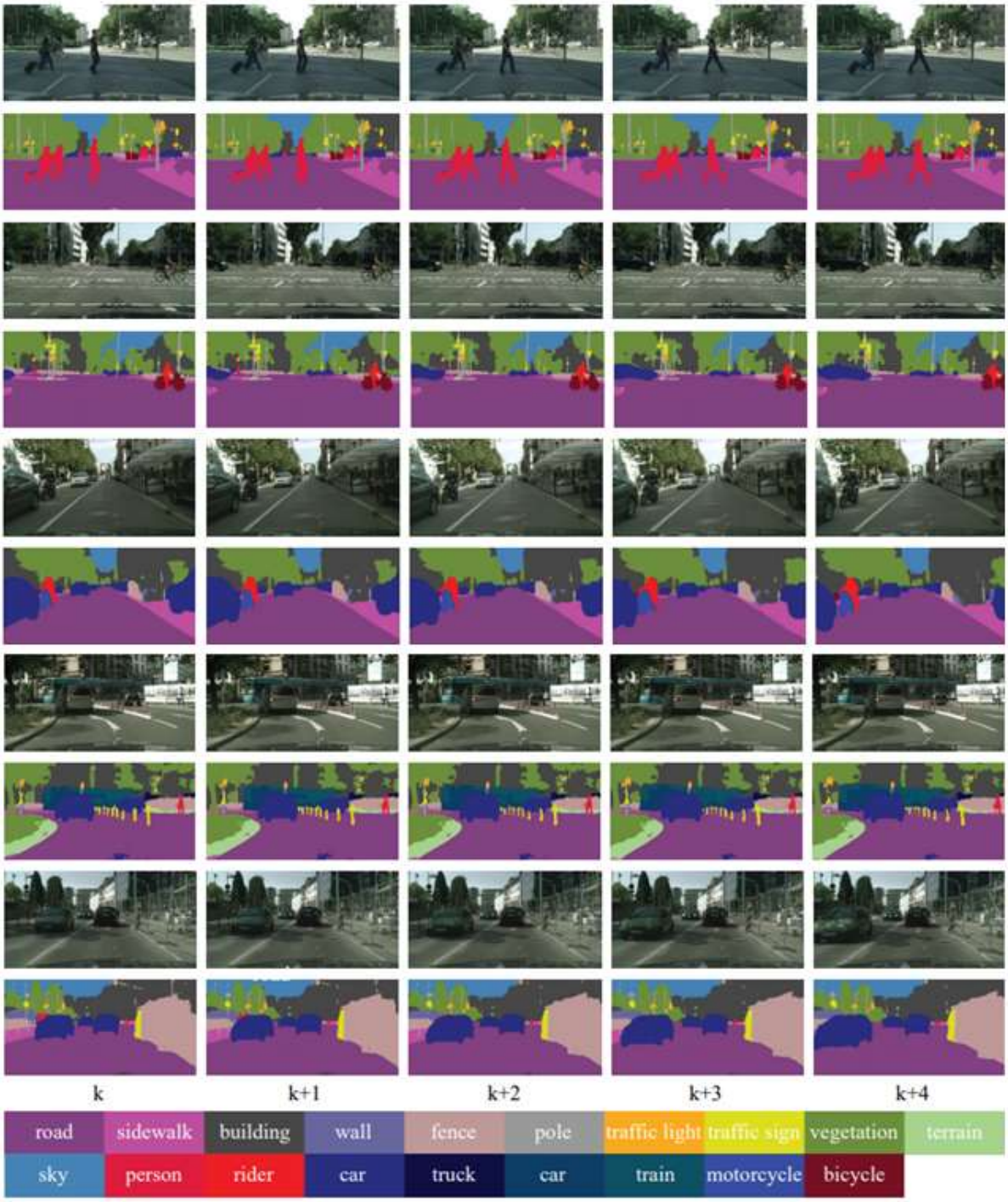

Рисунок 5.3 — Результати сегментації набору даних перевірки Cityscapes

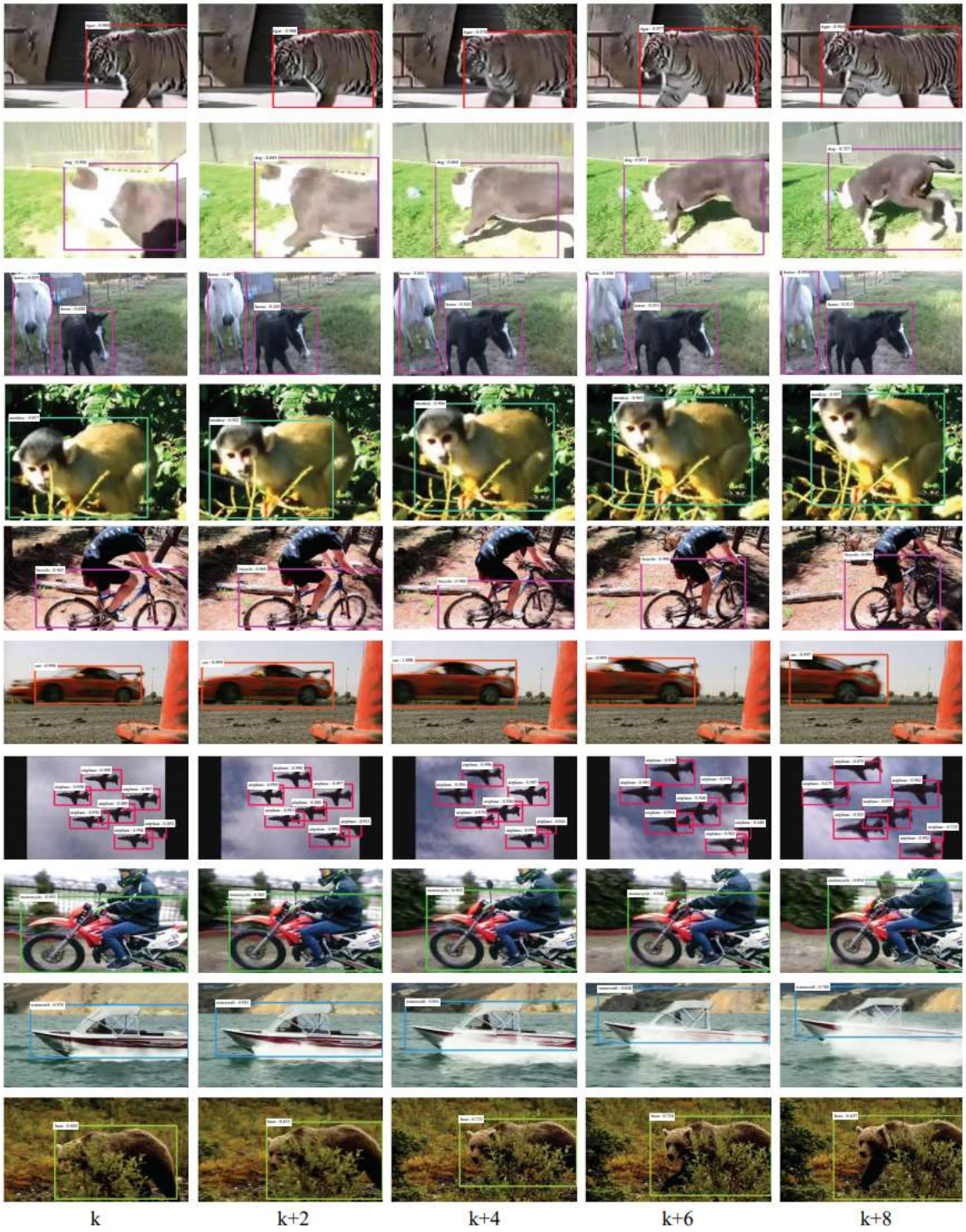

Рисунок 5.4 — Результати виявлення об’єктів у наборі даних перевірки VID ImageNet

Таблиця 5.4 — Архітектура мережі FlowNet

Шар	Тип	Крок	Вихід
conv1	7x7 conv	2	64
conv2	5x5 conv	2	128
conv3	5x5 conv	2	256
conv3 1	3x3 conv		256
conv4	3x3 conv	2	512
conv4 1	3x3 conv		512
conv5	3x3 conv	2	512
conv5 1	3x3 conv		512
conv6	3x3 conv	2	1024
conv6 1	3x3 conv		1024

Таблиця 5.5 — Архітектура мережі FlowNet Half

Шар	Тип	Крок	Вихід
conv1	7x7 conv	2	32
conv2	5x5 conv	2	64
conv3	5x5 conv	2	128
conv3 1	3x3 conv		128
conv4	3x3 conv	2	256
conv4 1	3x3 conv		256
conv5	3x3 conv	2	256
conv5 1	3x3 conv		256
conv6	3x3 conv	2	512
conv6 1	3x3 conv		512

6. ЕКОНОМІЧНА ЧАСТИНА

У даній роботі були досліджені існуючі методи обробки відеопотоків з рухомими об’єктами та розроблені відповідні технічні рішення для подальшого впровадження в системах моніторингу та контролю.

Техніко-економічне обґрунтування дослідження проводиться для отримання прибутку від вкладеного капіталу. Від якості підготовленої інформації про проект безпосередньо залежить розмір отриманого доходу. Якісні розрахунки та обґрунтування дозволять мінімізувати можливі ризики впровадження проекту.

Так як інновації за своїм складом та змістом близькі до реальних інвестицій, то, передусім, такі проекти вимагають розробки ТЕО.

В цьому випадку економічна частина роботи містить виконання таких етапів:

- оцінювання комерційного потенціалу розробки;
- прогнозування витрат на виконання наукової роботи та впровадження її результатів;
- прогнозування комерційних ефектів від реалізації результатів дослідження;
- розрахунок ефективності вкладених інвестицій та період їх окупності.

6.1.Розрахунок собівартості і вартості програмного продукту

Собівартість продукції складається з ряду найменувань витрат. Сюди входять: витрати на основні матеріали, витрати на програмне забезпечення, основна і додаткова зарплата, видатки на утримання та експлуатацію обладнання, а також цілий ряд загальнодержавних податків і відрахувань.

Наведемо переліки робіт для розробників програмного продукту. Досвідченим шляхом було встановлено тривалість робіт для кожного із співробітників, а також роботи над проектом в цілому.

У таблиці 6.1 показані співробітники, які працюють над реалізацією даного проекту, а також їх щоденні та щомісячні посадові оклади.

Перелік робіт, що виконуються співробітниками і їх тривалість наведена в таблиці 6.2 і становить 20 робочих днів у місяці.

Для дослідження були обрані три позиції співробітників, а саме дослідника, тестувальника та програміста.

Таблиця 6.1 – Склад виконавців робіт

Посада	Посадові оклади, грн	
	Місячні	Денні
Програміст	38000	1900
Дослідник	42300	2115
Тестувальник	37000	1850

Таблиця 6.2 – Розрахунок трудомісткості робіт

Вид робіт	Трива-лість, дні	Трудомісткість, чол / дні	Виконавець		
			Дослідник	Програ-міст	Тестува-льник
Підготовча робота					
Постановка задачі	1	1	+		
Аналіз предметної області	3	9	+	+	+

Продовження таблиці 6.2

Вид робіт	Трива-лість, дні	Трудомісткість, чол / дні	Виконавець		
			Керівник	Програміст	Тестувальник
Технічне завдання (ТЗ)					
Визначення вимог до програми і технічних засобів	2	6	+	+	+
Погодження та затвердження ТЗ	1	3	+	+	+
Дослідження предметної області					
Збір необхідних даних для дослідження	45	90	+	+	
Розробка тестового середовища	110	215	+	+	+
Тестування	40	60		+	+
Розробка інструкції користувача	2	6	+	+	+
Впровадження					
Випробування на реальних даних	4	12	+	+	+

Продовження таблиці 6.2

Вид робіт	Тривалість, дні	Трудомісткість, чол / дні	Виконавець		
			Керівник	Програміст	Тестувальник
Здача продукту в експлуатацію	6	18	+	+	+
Всього:	214	420	360	419	329

Розраховуємо собівартість робіт. Почнемо з фонду основної заробітної плати (ЗП). Обчислимо основну ЗП учасників дослідження з урахуванням трудовитрат, кількості виконавців і середньоденної ЗП:

$$Зосн = 2115 * 360 + 1900 * 419 + 1850 * 360 = 2\,233\,500 \text{ грн}$$

Додаткова заробітна плата становить 10 ... 20% від основної заробітної плати (домовимося, що Здод становить 12%).

$$Здод = Зосн * 12\% = 266\,820 \text{ грн}$$

Фонд заробітної плати складає:

$$Ззаг = Зосн + Здод = 2\,233\,500 + 266\,820 = 2\,500\,320 \text{ грн}$$

Однак необхідно розрахувати єдиний соціальний внесок. На даний момент він становить 22%. Таким чином:

$$ЄСВ = Ззаг * 22\% = 2\,500\,320 * 22\% = 550\,070{,}4 \text{ грн}$$

Розрахуємо вартість матеріалів і комплектуючих, необхідних для дослідження. Результати розрахунків надані в таблиці 6.3.

Інші прямі витрати складають експлуатаційні витрати й амортизаційні відрахування.

Експлуатаційні витрати розраховуються за формулою:

$$Зев = Тмч * Смч,$$

де $Т_{мч}$ – час кодування та налагодження програмного продукту на ЕОМ.

$$Тмч = Q * m60 * Kкв = 4202 * 1260 * 1{,}3 = 646.46с,$$

де $С_{мч}$ – вартість машинного часу ($С_{мч} = 17$ грн./год); m – середній час на кодування та відладку однієї умовної команди ($m = 12$ хв). Експлуатаційні витрати $З_{ев} = 7\,964{,}32$ грн.

Таблиця 6.3 – Витрати на матеріали для розробки програми

Матеріал	Кількість, шт.	Ціна за одиницю, грн.	Сума, грн.	Призначення
Картридж для принтеру	5	450	2 250	Друк документації
Папір формату А4	1000	0,3	300	Друк документації, звітів
Флеш-пам’ять	3	4000	12 000	Запис інформації

Продовження таблиці 6.3

Матеріал	Кількість, шт.	Ціна за одиницю, грн.	Сума, грн.	Призначення
Послуги мережі інтернет	3	150	450	Пошук інформації
Ліцензійна Windows 10	3	2800	8 400	Операційна система
Microsoft Office 365	3	1400	4 200	ПО для створення документації
Підсумок			27 600	

Розрахунок амортизаційних відрахувань. Для виконання робіт, пов'язаних з проектуванням програмного продукту та супутньої документації потрібно обладнання, яке представлене в таблиці 6.4.

Таблиця 6.4 – Допоміжне обладнання

Устаткування	Кількість, шт.	Ціна за одиницю, грн.	Сума, грн.
Ноутбук	3	40 000	120 000
Стіл	3	2 500	7 500
Стілець	3	3 950	11 850
Роутер	1	1 000	1 000
Комп’ютерна мишка	3	700	2 100
Принтер	1	2 700	2 700
Підсумок			262 110

Норму амортизації на технічні засоби $Н_а = 0{,}21$.

Річні амортизаційні відрахування на повне відновлення технічних засобів розраховуватися за формулою:

$$Ав = ЦоНа = 262\ 110 * 0.21 = 55\ 043.1 \text{ грн.}$$

Амортизаційні відрахування за період створення програмного продукту:

$$Ап = Ав * \frac{Кдн}{Крд}$$

де $К_{вд} = 214$ дн. – кількість відпрацьованих днів (з таблиці 6.2);

$К_{рд} = 249$ дн. – кількість робочих днів у році.

$$Ап = \frac{55\ 043.1*214}{249} = 43\ 306.12 \text{грн.}$$

Таблиця 6.5 – Собівартість і ціна продукту

№	Статті	Сума, грн.	Примітка
1	Основна заробітна плата (ОЗП)	2 233 500	Зосн
2	Додаткова заробітна плата (ДЗП)	266 820	12% від ОЗП
3	Єдиний соціальний внесок	550 070,4	22% (ОЗП +ДЗП)

Продовження таблиці 6.5

№	Статті	Сума, грн.	Примітка
4	Матеріали й куплені вироби	27 600	із таблиці 6.3
5	Амортизація	43 306,12	Ап
6	Собівартість (С)	3 121 296,52	п.1+п.2+п.3+п.4+п.5
7	Прибуток (П)	624 259,3	20% від С
8	Ціна без ПДВ	3 745 555.82	П + С
9	ПДВ	749 111.16	20% від ціни без ПДВ
10	Ціна з ПДВ	4 494 667	п.8 + п.9

6.2.Розрахунок точки беззбитковості

При впровадженні у виробництво продукту важливо знати чи стане цей виробничий процес рентабельним і чи буде він приносити бажаний прибуток. Для цього необхідно визначити точку беззбитковості (ТБ) і зобразити її графічно.

Для підтвердження стійкості продукту на ринку необхідно, щоб значення ТБ було менше значення номінальних обсягів виробництва. Чим далі від них значення ТБ (у відсотковому співвідношенні), тим стійкіше проект. Проект зазвичай визнається стійким.

Точку беззбитковості можна розрахувати за формулою:

$$N\text{б} = \text{К}/\text{Ц} - \text{С},$$

де К – умовно-постійні витрати, приймаємо рівними ціні програми;

Ц, С – ціна і собівартості одиниці виробу.

Ціна програми визначається за формулою:

$$Цпрог = Спрог + Ппрог,$$

де Спрог – загальний кошторис витрат (собівартості) програми, грн;

Ппрог – плановий прибуток, що забезпечує рентабельну роботу безпосередніх виконавців проекту, грн.

Розрахуємо вартість об'єкта:

$$Цпрог = 3\,121\,296{,}52 + 624\,259{,}3 = 3\,745\,555.82 \text{ грн.}$$

Розрахована ціна є переддоговірної ціною розробника - це мінімально допустима ціна, що враховує кошторис витрат на розробку теми і прибуток, розраховану за установчого коефіцієнту рентабельності.

При остаточному призначенні ціни теми необхідно врахувати надбавки, пов'язані зі збутом вироби. Податок на додану вартість приймається 20% від вартості об'єкта. Точка беззбитковості дорівнює:

$$N_б = 4\,494\,667 / (3\,745\,555.82 - 3\,121\,296{,}52) = 7.2 \approx 8 \text{ шт.}$$

Таким чином, показник ТБ = 8. Це означає, що реалізація третього приладу забезпечить беззбитковість проекту.

Графічне представлення ТБ представлено на рис. 6.1.

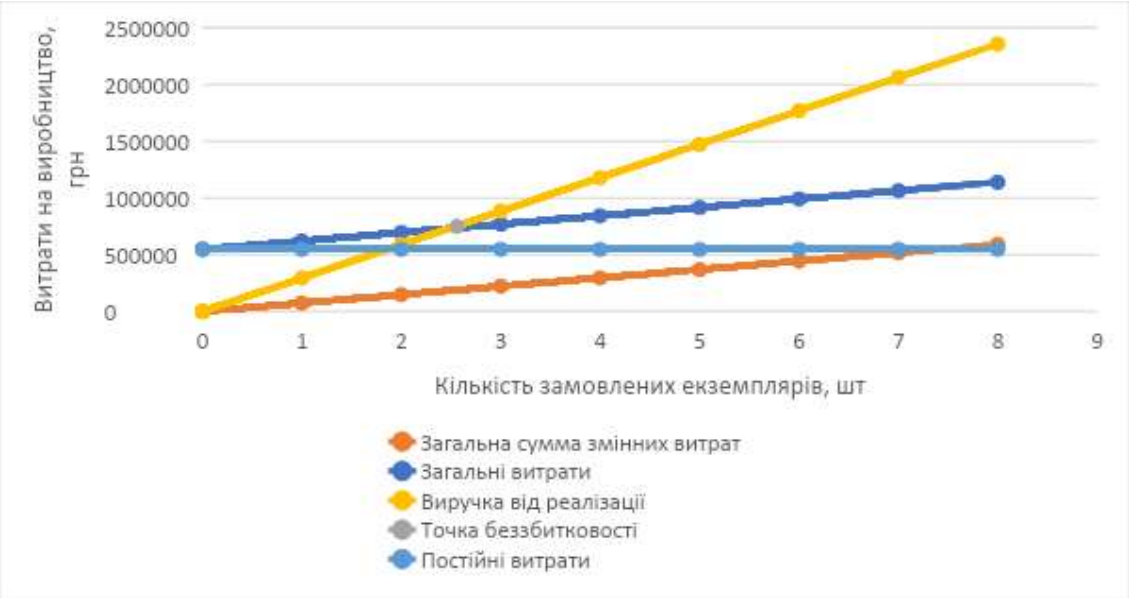

Рисунок 6.1 – Графік беззбитковості

Точка беззбитковості є важливим показником при визначенні фінансового положення компанії. Фінансова стійкість компанії залежить від обсягів виробництва над точкою беззбитковості.

Графік беззбитковості показує, що при обсязі продажів 3 шт. дохід від реалізації продукції перевищує загальні витрати, таким чином, в зоні між ними ми отримуємо прибуток.

6.3.Висновки

У результаті економічних розрахунків було проведено економічне обґрунтування дослідження методів обробки відеопотоків з рухомими об’єктами. Розрахунок вартості проведений з урахуванням всіх трудовитрат, ПДВ, відрахувань в пенсійний фонд, у фонд зайнятості і відрахувань на соціальне страхування, накладних витрат. Розрахована собівартість дослідження та подальшого впровадження отриманих концепцій та програмних продуктів до ринку складає 3 121 296,52 грн.

ВИСНОВКИ

Знаходження оптимального методу для розпізнавання об’єктів у відеопотоці досі залишається складною проблемою в області комп'ютерного зору. В даній роботі було приділено велику увагу збільшення ефективності роботи програм. Було встановлено, що системи розпізнавання об’єктів у відеопотоці далеко не ідеальні для адекватної роботи у всіх ситуаціях із реального світу.

У роботі представлений огляд основних методів та їх реалізація у галузі розпізнавання відео. Також були представлені результати тестувань, які дозволяють якість роботи кожного метода в однакових умовах. Це все дозволяє в майбутньому вибрати оптимальний метод для вирішення певного спектру задач. Якщо розглядати представлені в роботі методи в якості вирішення проблем систем моніторингу за допомогою відеоданих. То можна зробити висновок про те, що вибір оптимального методу залежить від класу задач які підлягають вирішення, та матеріальних і часових ресурсів організації яка планує впроваджувати дані системи. [7]

Також, не менш важливим параметром є правильне налаштування системи, так як при точних налаштуваннях можна добитися кращих результатів навіть за допомогою відносно простих методів, в порівнянні з більш складними але неправильно налаштованими. А це в свою чергу вимагає кваліфікованих працівників для налаштування цієї системи, тому дозволити даний тип рішення для моніторингу можуть собі дозволити тільки великі організації, з хорошим фінансовим забезпеченням. Все це в купі не дає однозначної відповіді на те який метод краще за все застосовувати в даних системах.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In ICLR, 2016.
2. Електронний ресурс «Від дослідження до виробництва» [Електронний ресурс]. URL: https://pytorch.org/
3. K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, ¨ F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.
4. Електронний ресурс «NumPy» [Електронний ресурс]. URL: https://numpy.org/
5. Електронний ресурс «Keras. Простий. Гнучкий. Потужний» [Електронний ресурс]. URL: https://keras.io/
6. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. IJCV, 115(3):211–252, 2015.
7. Електронний ресурс «Обучение и оценка модели с Keras» [Електронний ресурс]. URL: https://habr.com/ru/post/485890/