Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Центр післядипломної освіти

Кафедра _____Програмної інженерії_____

(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти – другий (магістерський)

Дослідження та оптимізація методів класифікації зображень з використанням синтетичних даних

(тема)

Виконав: студент 2 курсу, групи ІПЗмзд – 18 – 1_____

_____Вилегжанін С. С._____

(прізвище, ініціали)

спеціальності 121 – Інженерія програмного забезпечення____

(код і повна назва спеціальності)

____Освітньо–наукової програми_____

(тип програми)

____Інженерія програмного забезпечення_____

(повна назва освітньої програми)

Керівник _____к. т. н., доц. каф. ПІ Вечур О. В._____

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З. В. Дудар

20___ р.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Центр післядипломної освіти

Кафедра _Програмної інженерії_____

Рівень вищої освіти – другий (магістерський)

Спеціальність _121–Інженерія програмного забезпечення_____

(код і повна назва)

Тип програми _освітньо–наукова програма_____

Освітня програма _Інженерія програмного забезпечення_____

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___» _____ 20___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові _____Вилегжаніну Станіславу Сергійовичу_____

(прізвище, ім’я, по батькові)

1. Тема роботи _Дослідження та оптимізація методів класифікації зображень з використанням синтетичних даних_____

затверджена наказом університету від «27» березня 2020 р. № 41Стз

2. Термін подання студентом роботи до екзаменаційної комісії 26 червня 2020 р._

3. Вихідні дані до роботи *систематизовані залежності параметрів налаштувань сцени до результатів навчання, методи вдосконалення синтетичних зображень для класифікації. Використовувати ОС Windows, Unity, Python, Jupyter Notebook, CNN, Tensorflow*_____

4. Перелік питань, що потрібно опрацювати в роботі *мета дослідження, аналіз практичного застосування результатів, огляд залежностей реалістичності зображення до результатів навчання, методи вдосконалення синтетичних зображень, алгоритми класифікації зображень*_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз предметної галузі	05.04.2020	Виконано
2.	Підготовка пояснювальної записки	01.05.2020	Виконано
3.	Підготовка презентації та доповіді	09.05.2020	Виконано
4.	Нормоконтроль, рецензування	16.05.2020	Виконано
5.	Попередній захист	22.05.2020	Виконано
6.	Занесення диплома в електронний архів	22.05.2020	Виконано
7.	Допуск до захисту у зав. кафедри	22.05.2020	Виконано

Дата видачі завдання «___» _____ 20___ р.

Студент _____

(підпис)

Керівник роботи _____ к. т. н., доц. каф. ПІ Вечур О. В.__

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 68 с., 22 рис., 9 табл., 3 дод., 12 джерел.

СИНТЕТИЧНІ ДАНІ, ЗГОРТКА, КЛАСИФІКАЦІЯ, РЕНДЕРІНГ, ПРОГНОЗ, ІГРОВИЙ ДВИЖОК, UNITY, АУГМЕНТАЦІЯ, СЕГМЕНТАЦІЯ, НЕЙРОН.

Метою роботи є дослідження методів класифікації зображень з використанням синтетичних даних у контексті використання ігрових движків.

Методи дослідження базуються на вже існуючих алгоритмічних базах для класифікації зображень, такі як згорткова нейрона мережа. Методи розробки методів апробації дослідження засновані на використанні таких технологій та мов програмування як Python, Tensorflow, Jupyter Notebook та Unity.

В результаті роботи було проведено аналіз різноманітних математичних та алгоритмічних рішень у контексті класифікації зображень побудовані залежності між параметрами налаштування генерації сцени та результатами навчання моделі, були розглянуті алгоритми надання реалізму до синтетичних зображень.

SYNTHETIC DATA, CONVOLUTION, CLASSIFICATION, RENDERING, FORECAST, GAME ENGINE, UNITY, AUGMENTATION, SEGMENTATION, NEURON.

The aim of the work is to study the methods of image classification using synthetic data in the context of the use of game engines and other supplementary techniques.

The research methods are based on already existing algorithmic bases for image classification, such as a convolutional neural network. Research approbation is based on the use of technologies and programming languages such as Python, Tensorflow, Jupyter Notebook and Unity.

As a result, the analysis of various mathematical and algorithmic solutions in the context of image classification, the relationships between the parameters of the scene generation settings and the learning outcomes of the model, the algorithms for providing realism to synthetic images were considered.

ЗМІСТ

ВСТУП

Сучасні програмні системи потребують все більше даних для реалізації нових комплексних алгоритмів оптимізації дій з рутинного життя кожної людини. Це і самостійно керовані машини, і системи безпеки та продукти у сфері робототехніки. У всіх напрямків життєдіяльності, які користуються рішеннями у контексті штучного інтелекту – є одна спільна проблема – це отримання специфічних даних для обраного домену. Кожен розробник має можливість побудувати новий продукт з використанням штучного інтелекту, але він може бути не конкуренто спроможним, чи працювати не досить ефективно, через той факт що навчена система машинного навчання не отримала достатньо даних. Здобуття конкретних даних може коштувати досить дорого та досить часто буває непрактичним (наприклад, якщо нам потрібно відстежувати об’єкти які можуть бути знищенні у процесі – вибухи, пожежі, хімічні реакції тощо). У нагоді в цьому напрямку стають системи створення синтетичних даних. Зараз існує досить істотна кількість систем, котрі мають можливість генерувати візуальні дані різноманітними методами.

Серед них предметом дослідження було обрано використання саме ігрових движків, адже вони мають надзвичайну кількість інструментів для маніпуляції зображенням, освітленням, фізичними властивостями та мають можливість точно та ефективно створювати фізично–реалістичні симуляції будь–яких ситуацій. Однак, для створення таких систем використовується монструозний набір можливих керованих параметрів, котрі треба систематизувати та класифікувати, яким чином вони впливають на результати навчання систем по класифікації зображень чи відео, та чи можливо пожертвувати декількома з них для оптимізації усього процесу загалом. Також, при використанні цього підходу значну роль відіграють саме геометричні об’єкти використані у процесі та те наскільки вони деталізовані [1].

Беручи до уваги актуальність цієї проблеми та ігрових движків як інструментів до її вирішення, метою даного дослідження є побудова залежності

налаштувань при генерації синтетичних зображень з задачею оптимізувати як і процес генерації даних, так і процес навчання. Додатково будуть розглядатися можливості та інструменти програмного вдосконалення штучно створених зображень для придання їм більшої реалістичності, та вплив цих механізмів та алгоритмів на результуючі системи.

У цьому контексті слід зазначити, що елементом наукової новизни при проведенні цього дослідження є систематизація та актуалізація залежностей налаштувань сцен у ігрових движках для створення синтетичних даних до результатів прогнозів по класифікації зображень за обраним доменом.

Для безпосередньої перевірки результатів, є релевантним використанням популярних та перевірених часом систем та алгоритмічних баз, наприклад таких, як TensorFlow, які використовуються у великій кількості сучасних проектів. Якщо результати дослідження покажуть достатню ефективність системи, то це доведе можливість створення більш точних моделей у випадку більш спеціалізованих програмних пакетів для машинного навчання.

Практичним значенням цього дослідження є автоматизація та оптимізація підходу до регуляції та керування процесом створення синтетичних візуальних даних для систем, які використовують класифікацію зображень. За результатами дослідження стане можливим планувати ступінь деталізації об’єктів потрібний для певного прикладного домену, що дозволить заздалегідь спланувати витрати на створення синтетично створених сцен. Результати проведення дослідження матимуть змогу надатиповну картину того, як певні фізичні та візуальні механізми ігрових движків можуть впливати на результати навчаннями та чи є вони досить ефективними, що дозволить створювати їх більш ефективно.

1 ОПИС ПРОВЕДЕНИХ ТЕОРЕТИЧНИХ ТА ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

1.1 Розгляд згорткової нейронної мережі як основного інструменту навчання

Ігрові движки, такі як Unreal Engine і Unity, – являють собою потужні інструменти для створення реалістичних симуляцій світу. На відміну від реального, вони можуть дозволити нейронним мережам навчатися в дешевих, безпечних, керованих, повторюваних середовищах з нескінченними ситуаціями, вражаючої графікою та наближеною до реального світу фізикою. Автомобілі без водіїв – чудовий приклад, якщо автомобіль стає не придатним для використання під час тренувань, це коштує часу, грошей та потенційно людських життів. Навіть якщо ушкодження вдасться уникнути втручанням людини, неможливо практикувати небезпечні ситуації безпечним способом для людей всередині або поза транспортним засобом. Якщо автомобіль розбивається в симуляції, він може або почати спочатку, або навчитися безпечно тягнути машину на бік дороги. Система симуляції ситуацій також може практикувати стільки разів, скільки це буде потрібно.

Першим кроком, перед початком аналізу специфіки навчання моделей з використанням синтетичних даних, слід розглянути саме інструменти навчання моделей, які доступні для переважної більшості розробників. Одним з таких є вже раніше обраний Tensorflow, який включає в себе велику кількість математичних реалізацій нейронних мереж. У досліджуваному випадку ми розглядаємо інструментарій для класифікації зображень чи відео. Одним з основних та часто використовуваних елементів навчання моделі є згорткові нейронні мережі. Згорткова нейронна мережа (Convolutional neural network), надалі – CNN, – це нейронна мережа, в якій щонайменше один шар є звивистим (розділяється на декілька шляхів). Типовий випадок складається з певної комбінації наступних шарів:

– згорткові шари;

– шари пулінгу (об’єднання);

– повноз’єднані (закриті) шари.

Слід зазначити що саме CNN мають великий успіх у видах проблем, саме таких, як розпізнавання зображень. Схематичну структуру / архітектуру навчання моделі можна побачити на рисунку 1.1.

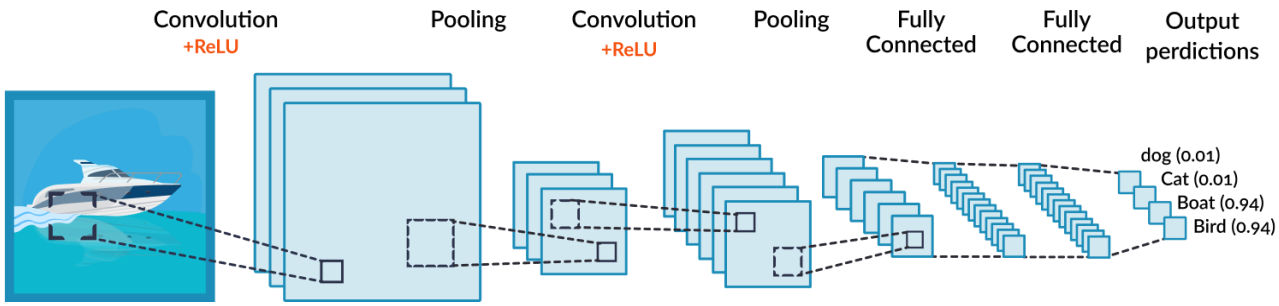

Рисунок 1.1 – Базова архітектура навчання моделі CNN

Далі розглянемо кожен з шарів та математичні операції які виконуються при аналізі зображення у вигляді матриці. У машинному навчанні згортка (convolution) змішує фільтр та похідну матрицю для тренування ваг. Термін «згортання» в машинному навчанні часто є скороченим способом посилання на операцію згортки, або на згортковий шар. Алгоритм машинного навчання без згортків повинен був би вивчити окрему вагу для кожної комірки у великому тензорі [1].

Наприклад, навчання алгоритму машинного навчання на зображеннях 2048 x 2048 буде змушене знаходити окремі ваги 4M. Завдяки згорткам алгоритм машинного навчання повинен лише знайти ваги для кожної комірки у згортковому фільтрі, що значно зменшує пам’ять, необхідну для тренування моделі. Згортковий фільтр (Convolutional filter) – це матриця, що має той же ранг (ступень), що і похідна матриця, але значно меншої форми. Такі фільтри можуть бути побудовані вручну в залежності від поставленої задачі, однак, інноваційність та головна перевага CNN полягає саме в автоматичному визначенні та побудові фільтрів. Це

ще одна додаткова причина, чому такі мережі застосовуються в задачах класифікації Наприклад, маючи вхідну матрицю розміром 28x28, фільтром може бути будь–яка 2D матриця, менша ніж 28x28. Приклад накладання фільтру можна побачити на рисунку 1.2.

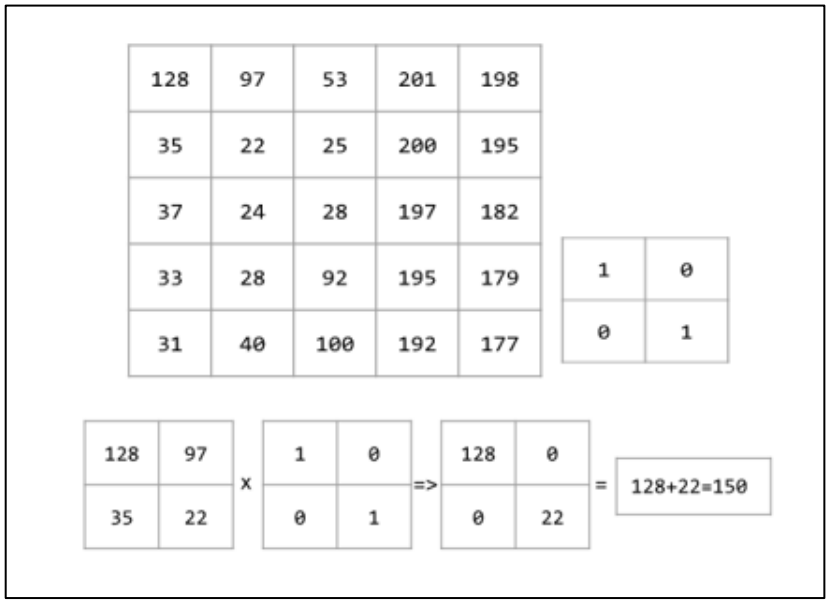

Рисунок 1.2 – Накладання фільтру

У машинному навчанні згорткові фільтри, як правило, задаються випадковими значеннями, а потім мережа тренується на ідеальних значеннях цього фільтру. Розглянувши теоретичні відомості про згортку та згортковий фільтр слід розглянути безпосередньо практичне поняття, етап навчання – шар згортки. Простими словами, згортковий шар – це накладання згорткового фільтру на початкову матрицю та отримання нової. Кожне «накладання» матриці фільтру на початкову називається операцією згортки. Також така процедура дозволяє оперувати значно меншими порядками матриць, ніж початкові. Адже може використовуватись зображення з великим розширенням (наприклад , може транслюватися до матриці 800x600). Побачити приклад виконання цієї операції можна на рисунку 1.3. Зліва початкова матриця, жовтим кольором відмічено

значення на які накладається фільтр, голубим – результуюче значення у похідній матриці.

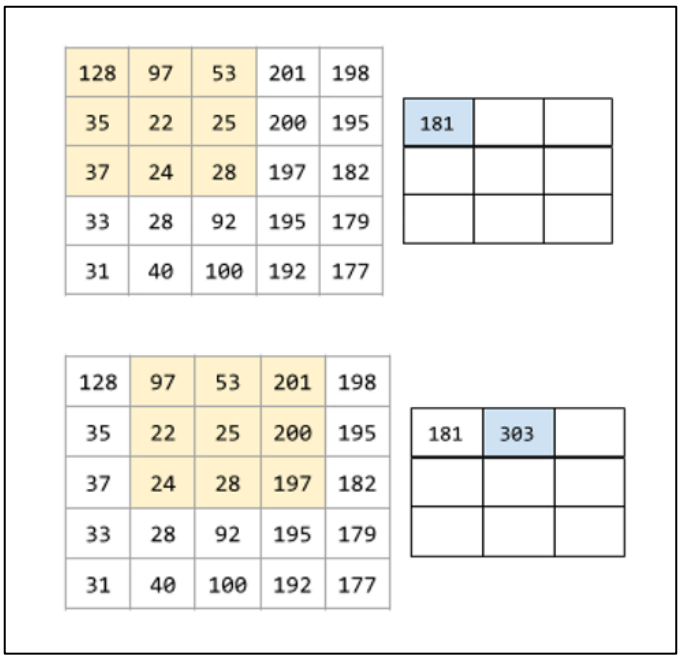

Рисунок 1.3 – Виконання операції згортки

Наступним будівельним елементом у структуризації моделі CNN є процес пулінгу матриці. Наприклад, наведена така матриця розмірами 3x3 (за якою буде зменшуватись порядок первинної матриці), яка зображена на рис. 1.4.

Рисунок 1.4 – Матриця 3x3

Пулінг (pooling) – це процес приведення матриці (або матриць), створеної попереднім згортковим шаром, до матриці меншого ступеня (порядку). Він зазвичай передбачає взяття максимального або середнього значення на всій

об'єднаній площі. Операція пулінгу, подібно до згорткової операції, ділить похідну матрицю на фрагменти, а потім просуває цю згорнуту операцію кроками [2].

Наприклад, припустимо, що операція об’єднання ділить матрицю на 2x2 зрізи з кроком 1x1. Як показано на наступній схемі, зображеній на рисунку 1.5, відбуваються чотири операції об’єднання, кожна операція об’єднання вибирає максимальне значення чотирьох у цьому фрагменті.

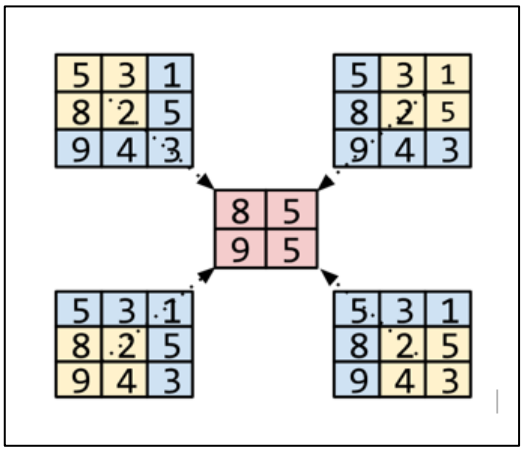

Рисунок 1.5 – Схема операції пулінгу

Пулінг допомагає забезпечити транзитивну інваріантність вхідної матриці. Транзитивна інваріантність – це є можливість алгоритму успішно класифікувати зображення навіть тоді, коли положення об’єктів усередині зображення змінюється. Наприклад, алгоритм може ще ідентифікувати собаку, чи то вона знаходиться в центрі кадру, чи в лівому кінці кадру.

Пулінг у теорії комп’ютерного зору більш формально відоме, як просторове об’єднання. Програми часових рядів зазвичай позначають пулінг, як «тимчасове об’єднання». Менш формально це об’єднання часто називають subsampling або downsampling. Останнім блоком при створенні конволюційної моделі є повноз’єднаний шар. Повноз’єднаний шар – це закритий шар, в якому кожен вузол з’єднаний з абсолютно кожним вузлом наступного закритого шару.

Закритий шар – це синтетичний шар в нейронній мережі між вхідним шаром (тобто особливостями, features) і вихідним шаром (передбачення, predictions). Закриті шари зазвичай містять функцію активації (наприклад, ReLU) для тренувань. Нейронна мережа може містити більше одного прихованого шару. Приклад візуалізації з'єднання шарів з повноз'єднаним шаром можна побачити на рисунку 1.6.

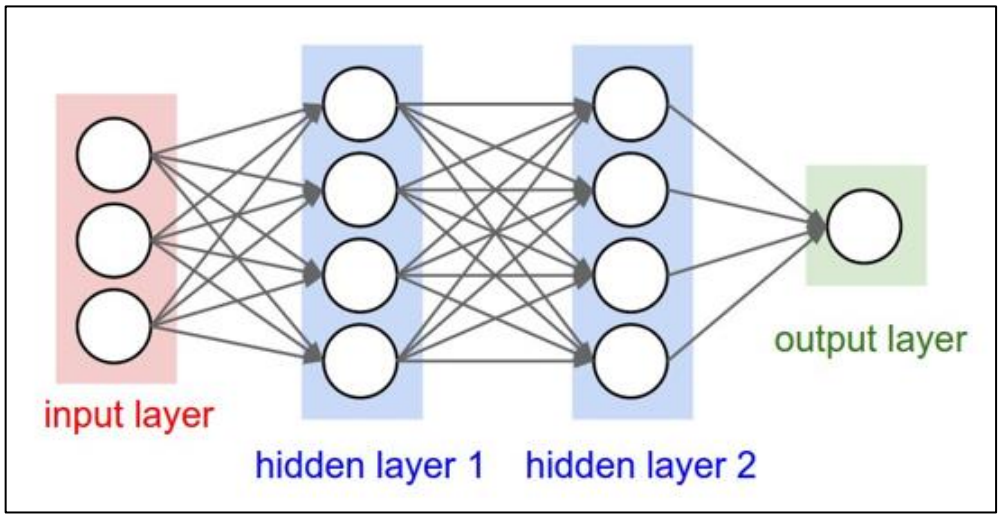

Рисунок 1.6 – Візуалізації з'єднання шарів з повноз'єднаним шаром

У практичному сенсі Tensorflow дозволяє дуже легко будувати послідовні переходи з одного шару конволюційної нейронної мережі до іншого. Як вхідні дані, CNN приймає «тензори» форми (висота зображення, ширина зображення, кольорові канали), ігноруючи розмір батчу. Наприклад, налаштуємо наш CNN для обробки входів форми (32, 32, 3). Це можна зробити, передавши аргумент input_shape до нашого першого шару:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

1.2 Розгляд функції активації

Щоб завершити приклад моделі, ми подаємо останній шар на виході зі згорткової основи (форми 4x4x64) в один або кілька повноз’єднаних шарів для виконання класифікації. Повно з’єднані шари приймають вектори, як вхідні дані (що є однопросторовим), тоді як вихідний потік – 3D. За допомогою директиви «activation=» ми вказуємо функцію активації [3].

Дамо визначення функції активації та оптимальні види її використання у контексті класифікації синтетичних зображень.

Функція активації – це функція, яка приймає зважену суму всіх значень з попереднього шару, а потім генерує та передає вихідне значення (як правило, нелінійне) наступному шару.

Існує декілька видів функцій активації – шагова, лінійна, сігмоїд, тан, ReLu (rectified linear unit або функція випрямляч). Коротко розглянемо практичну роль кожної з них та вирішемо котра з них найбільш адаптована під мету цього дослідження.

Шагова функція – це така функція для котрої її значення Y , якщо воно вище порогу, воно автоматично стає активованим. Якщо значення нижче за поріг, то це значення стає неактивованим. Подібний тип функції є досить примітивним, адже він є бінарним, тобто має змогу відповідати критерію лише «так» чи «ні». Припустимо, створюється бінарний класифікатор.

Система, яка повинна бінарно сказати «так» або «ні» (активувати або не активувати). Тепер розглянемо випадок використання, коли потрібно підключити кілька таких нейронів, щоб отримати більше різноманітних класів. –Class1, class2, class3 тощо. Що станеться, якщо більше 1 нейрона буде «активовано». Така ситуація породжує невизначеність. Подібна невизначеність у задачі класифікації зображень, де класи можуть бути дуже диферсифіковані, тим паче з використанням синтетичного набору даних.

Візуально дуже легко зрозуміти цю концепцію активації. Графічний вигляд шагової функції можна побачити на рисунку 1.7.

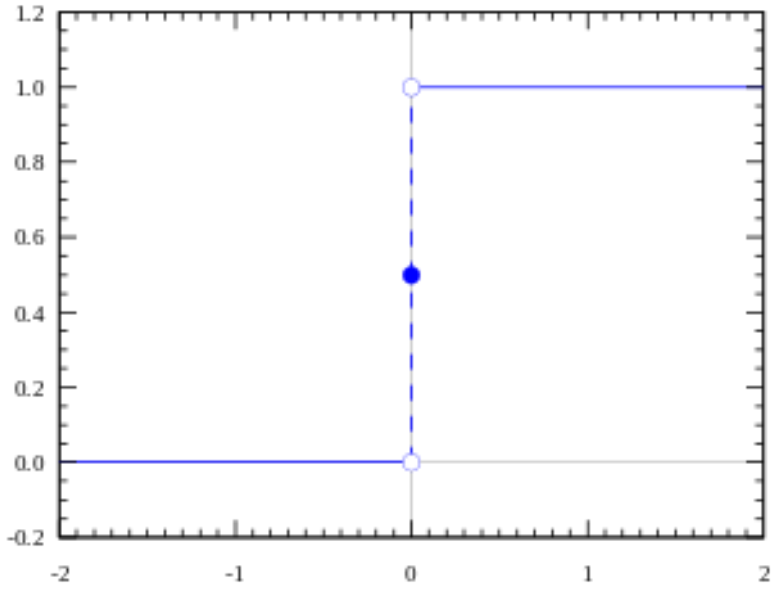

Рисунок 1.7 – Шагова функція

Лінійна функція – функція прямої лінії, де вихідне значення пропорційно вхідному (що є зваженою сумою від нейрона).Таким чином, дана функція дає діапазон активацій, а саме тому це не є бінарною у порівнянні з шаговою. У цьому випадку є можливість з’єднати кілька нейронів разом.

Цей тип функції не підходить для випадку класифікації зображень через те, що нівелює можливість використовувати декілька шарів. Незалежно від того, скільки у нас шарів, якщо всі вони мають лінійний характер, остаточна функція активації останнього шару – це не що інше, як лише лінійна функція введення найпершого шару [4].

Сігмоїдна функція – це функція, яка у своїй суті є нелінійною, тому її поєднання також будуть нелінійні. З використанням даної функції можливо використання декількох поступових шарів моделі.

Іншою перевагою цієї функції активації є те, що на відміну від лінійної функції, результат функції завжди буде знаходитися в діапазоні (0,1) порівняно з

(–inf, inf) лінійної функції. Це означає, що значення завжди будуть знаходитися у контрольованому діапазоні. Вигляд графіку можна побачити на рисунку 1.8.

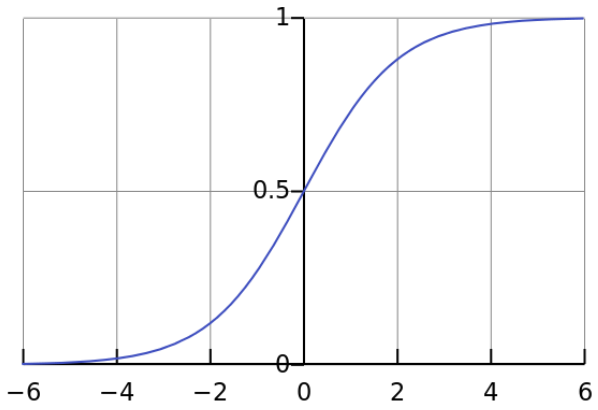

Рисунок 1.8 – Сігмоїд

Також слід відмітити одну з варіацій сигмоїду – це тан функція.

Тан це ніщо інше як масштабована сигмоїдна функція. Однак, одна з її найважливіших властивостей це те, що значення знаходиться у діапазоні [–1;1]. Це означає, що за будь–яких обставин значення активації не можуть досягти дуже великих значень, тому вона є більш оптимізованою (нейронні мережі легше оперують з меншими значеннями).

Окремим моментом слід зазначити, що градієнт сильніший для тану, ніж для сигмоїду. Вибір між сигмоїдом та таном буде залежати від вимоги до «сили» градієнту. Як і сигмоїд, тангент функція також має проблему поступового зникнення градієнта.

Ця проблема полягає у тому що, чим більше шарів використовується у нейронній мережі тим більше градієнт функції втрат наближається до нуля, що робить мережу важкою для навчання. Причина цьому, те що функція стискає великий вхідний простір до простору між 0 і 1 (чи іншого). Тому велика зміна вхідних даних функції спричинить дуже незначну зміну результату навчання.

Графічний вигляд такої функції дає розуміння цієї концепції. Його можна побачити на рисунку 1.9.

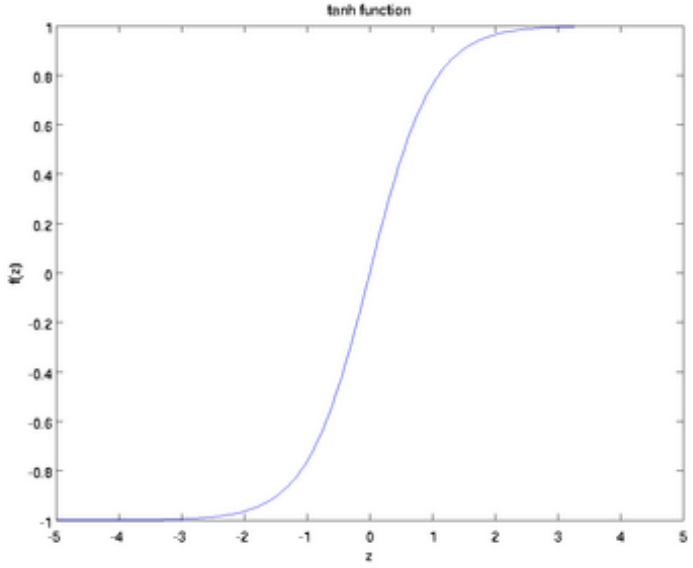

Рисунок 1.9 – Тан функція

ReLu має вигляд A(x) = max(0,x), графік якої наведений на рисунку 1.10. Ця функція дає результат x, якщо x додатний, та 0 в іншому випадку.

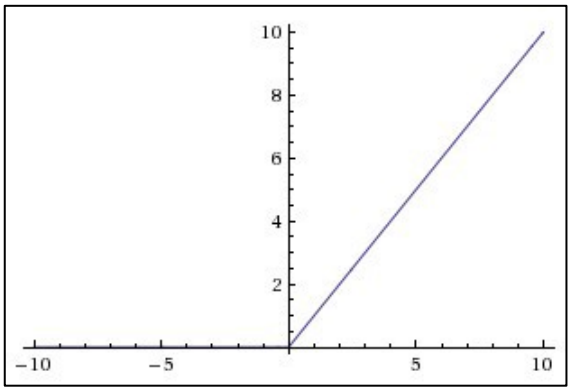

Рисунок 1.10 – Функція ReLu

На перший погляд може здатися, що даний випадок має такі ж самі проблеми як і лінійна функція.

Перш за все, ReLu все ще має нелінійний характер. Тому і комбінації ReLu також будуть нелінійні, функція по своїй суті є гарним наближувачем значень. Будь–яку функцію можна наблизити до комбінацій ReLu. Діапазон ReLu становить [0, inf).

Ще один момент, який слід зазначити, – це розподіленість значень активації. Візьмемо, наприклад, велику нейронну мережу з великою кількістю нейронів. Використання сигмоїдної функції або тану призведе до того, що майже всі нейрони активуються аналоговим способом. Це означає, що майже всі значення функції активації будуть оброблені для опису виходу мережі. Іншими словами, активація є щільною. Це неефективно з точки зору продуктивності.В ідеалі потрібно, щоб кілька нейронів у мережі не залишалися неактивними і тим самим робили значення активації розсіяними та ефективними.

ReLu надає нам цю перевагу. Взявши мережу з випадковими вагами (або іншими словами – нормалізована), майже 50 % мережі буде мати 0 активацій через основну характеристику ReLu (вихід 0 для від’ємних значень x). Це означає, що менша кількість нейронів спрацьовує (рідка активація), а мережа легша, але нічого не бездоганне, навіть ReLu.

Через горизонтальну лінію в ReLu (для від’ємного X) градієнт може піти в бік 0. Для активації в цій області, ReLu градієнт буде дорівнювати 0, через що ваги не будуть коригуватися під час спуску. Це означає, що ті нейрони, які переходять у цей стан, перестають реагувати на зміни помилки / введення (за тою просто причиною, що градієнт дорівнює 0, тому нічого не змінюється). Це називається вмираючою проблемою ReLu [5].

Ця проблема може змусити кілька нейронів просто «померти» (тобто стати неактивними) і не реагувати, роблячи значну частину мережі пасивною. У ReLu є варіанти, щоб пом’якшити цю проблему, просто перетворивши горизонтальну лінію в негоризонтальну складову. наприклад, $y = 0.01x$ для $x < 0$ зробить його трохи похилою лінією, а не горизонтальною. Такий вид має назву нещільного ReLu. Існують також й інші інтерпретації, для кожного типу задачі – треба підібрати

оптимальний. Основна ідея – дозволити градієнту бути не нульовим і мати здатність відновитись під час тренування моделі.

ReLu є обчислювально менш дорогим, ніж сигмоїд, оскільки він передбачає більш прості математичні операції. Це хороший момент, який слід враховувати, адже ми використовуємо саме CNN, що є глибокою нейронною мережею.

Отже визначивши, що функція активація ReLu є найефективнішою у ситуації многошарового побудування моделі (а саме така використовується для класифікації зображень), то маємо можливість використовувати її у навчанні нашої моделі, наприклад ось так:

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Через специфіку того, що ми використовуємо згенеровані дані ми можемо отримати разом з ними мета–дані які описують детально налаштування сцени, а також позиції об’єктів та опис додаткових використаних компонентів – світло, ефекти тощо. Наявність цих даних відкриває велику кількість можливостей у напрямку автоматичного підлаштування та оптимізації процесу навчання. Наприклад, виділивши певні конфігурації сцени (такі як розширення текстур) та динаміку їх змін можна побудувати залежність між ними та результатами навчання. Є два підходи до реалізації цього механізму – адаптація підчас навчання або збереження додаткових даних по закінченню навчання. Використання першого підходу вимагає встановлення певних контрольних точок (наприклад 200, 400 зображень), збереження зліпку навченої системи та порівняння з попереднім результатом у контексті зміни налаштувань. Такий процес буде займати експоненціально більше часу через різноманіття можливих вхідних параметрів. Отримання звіту після навчання та порівняння з іншими результатами матиме більш ефективний характер.

Один із способів розглядати реалізм у рамках синтетичних даних – це мультиплікативна кількість. Мається на увазі, щоб створити реалістичні синтетичні зображення, ми повинні забезпечити, щоб кожен із кроків, що були залучені, міг

дати реалістичні результати. Наприклад, вірогідність досягнення реалістичного зображення досить мала навіть при використанні фізично наближених характеристик, якщо геометрія низкополігональна, і навпаки. З цією метою є можливість виділити п'ять ортогональних властивостей реалізму, які є ключовими для досягнення цієї мети:

- композиція сцени;
- геометрична структура;
- освітлення джерелами світла;
- властивості використаних матеріалів;
- додаткові оптичні ефекти.

Іншим аспектом який привносить наявність мета даних у тандемі з предметом навчання – зображення, це полегшення процесу аугментації, удосконалення та сегментації зображень. Через те що ми можемо отримати абсолютно точні позиції кожного з об’єктів сцени, а також супутні дані такі як прозорість, матеріал, ім’я дані процеси можуть стати дуже точними та не будуть потребувати складних алгоритмів комп’ютерного зору бо усі додаткові дані заздалегідь визначені.

Навчання моделей машинного навчання на стандартних синтетичних зображеннях є проблематичним, оскільки зображення можуть бути недостатньо реалістичними, що призводить до того, що модель може вивчити деталі, наявні лише в синтетичних зображеннях, і не може добре узагальнити реальні зображення.

Одним із підходів до усунення цього розриву між синтетичними та реальними зображеннями було б безпосереднє вдосконалення системи генерації зображень, що може виявитись дорогою, неузагальненою та складною операцією, і навіть найкращий алгоритм візуалізації все ще не зможе відтворювати всі деталі, наявні в реальних зображеннях. Ця відсутність реалізму може змусити моделі надмірно підходити до «нереальних» деталей синтетичних зображень.

Мета «додання реалізму» полягає в тому, щоб зображення виглядали максимально реалістично, адже це впливає досить істотно на результати навчання.

Для виконання такої задачі необхідно підготувати окрему нейрону мережу, яка буду виступати «покращувачем зображення». Для того щоб навчити подібну мережу для обробки зображень, необхідні дані з реального світу. Одним із варіантів було б використовувати саме пари реальних та синтетичних зображень із піксельним співвідношенням або як було зазначено раніше –реальних зображень із анотаціями (мета–даними) [6].

Щоб створити піксельну відповідність, потрібно або зробити синтетичне зображення, яке відповідає заданому реальному зображенню, або зафіксувати реальне зображення, яке відповідає зображеному синтетичному зображенню.

Якщо це так, то буде можливість створити великий набір синтетичних даних з прямою відповідністю до реальних, що зробить подібний метод швидким і дешевим [7]. Для того щоб навчити нашу мережу для покращення за допомогою способу «навчання без вчителя», ми використовуємо допоміжну дискримінаційну мережу, яка класифікує реальні та вдосконалені (або підроблені) зображення на два класи.

Мережа – покращувач намагається «обдурити» цю дискримінаційну мережу вважати, що синтетичні зображення є справжніми. Обидві мережі тренуються по черзі, і навчання припиняється, коли дискримінатор не може відрізнити реальні зображення від підроблених [8].

Ідея використання змагальних мереж схожа на підхід GAN (Generative Adversarial Networks), який відображає випадковий вектор до зображення таким чином, що сформоване зображення не відрізняється від реального. Будування генеративних моделей – це неконтрольоване навчальному завданню машинного навчання, яке включає автоматичне виявлення та вивчення закономірностей або закономірностей вхідних даних таким чином, що модель може бути застосована для створення або виведення нових результатів, які правдоподібно [9], могли бути виведені з початкового набору даних. При використанні цього інструменту також існує дві системи – генератор та дискримінатор, які змагаються. Однак, основною задачею GAN мереж є саме відтворення нових даних, наприклад фільтрованих відео.

Приклад удосконаленого штучного зображення за допомогою додавання реалізму мережою–покращувачем можна побачити на рисунку 1.11. У порівнянні з реальними зображеннями суттєвої різниці не спостерігається.

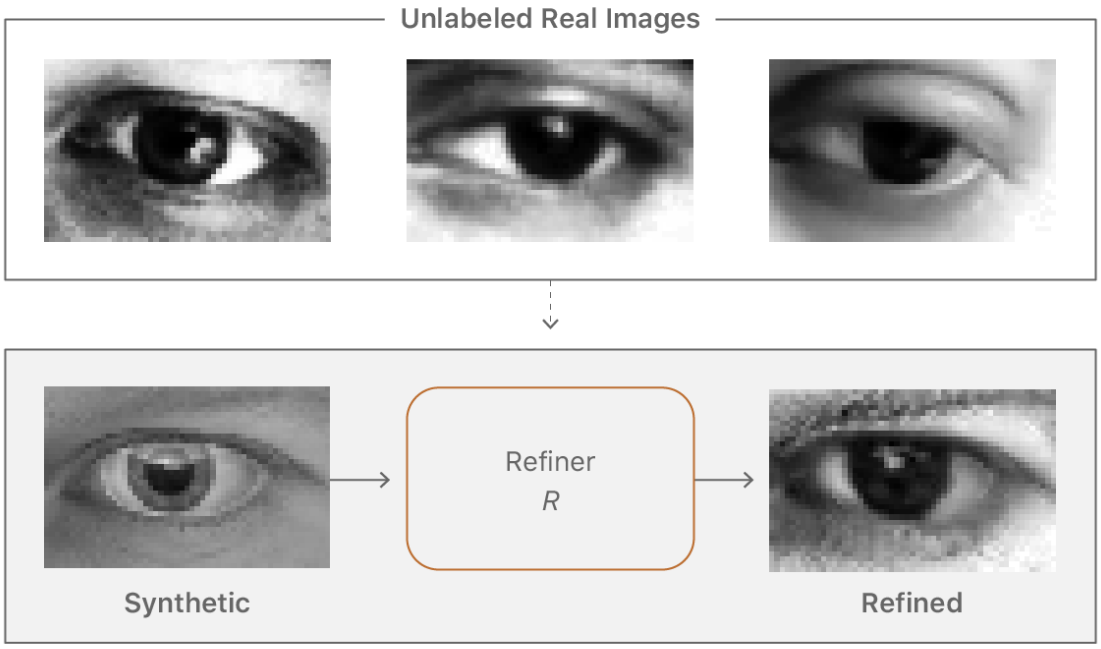

Рисунок 1.11 – Приклад надання додаткового реалізму до синтетичного зображення ока людини

У процесі розв’язування подібних проблем зазвичай найважчим та найдорожчим етапом є анотування зображень. Однак, розглядається випадок генерації синтетичних даних саме за допомогою ігрового движку, тому ця проблема вирішена, адже є можливість отримати повний стан системи у будь–який момент часу.

Ще одна ключова вимога до нейронної мережі «покращувача» – це те, що вона повинна навчитися моделювати реальні характеристики зображення, не породжуючи жодних зайвих артефактів. Коли ми тренуємо єдину дискримінаційну мережу, то мережа покращувача, як правило, надто підкреслює певні особливості

зображення, щоб обдурити поточну мережу дискримінатора, що призводить до створення артефактів.

Ключовим спостереженням є те, що будь–яка локальна зміна, взята з вдосконаленого зображення, повинна забезпечувати статистику до співвідносної локації на реальному зображенні. Тому, замість того, щоб визначати глобальну дискримінаційну мережу, ми можемо визначити мережу, яка класифікує всі локальні частини зображень окремо. Приклад такого розбиття на локальні зони можна побачити на рисунку 1.12.

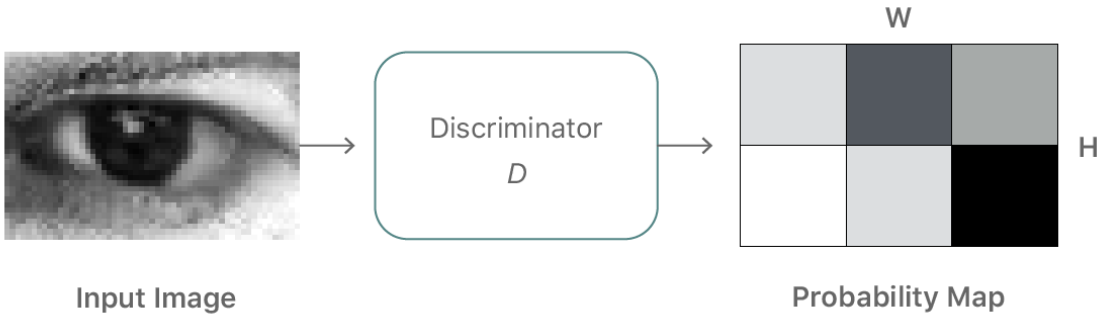

Рисунок 1.12 – Розбиття на зображення на локальні зони

Генератор може обдурити дискримінатора як зображеннями з нової вибірки, так і цільовим розподілом реальних даних. Генерація з нововведеного розповсюдження обдурює мережу–дискримінант лише до тих пір, поки мережа не дізнається про нововведення. Більш оптимальним методом, яким система створення може обдурити мережа–дискримінатор, є породження зображення з цільового розподілу, тобто з реальних даних [10].

З огляду на ці два способи перетворення мереж, найпростіше, зазвичай, генерувати нове зображення, що було розглянуто під час тренування поточного генератора та дискримінатора один проти одного.

Спрощена послідовність зображена на рисунку 1.13. Розподіл генератора та дискримінатора показано відповідними кольорами [11].

Інтегруючи історію попередніх ітерацій, в якій зберігаються зліпки генеруючих мереж, мережа – дискримінатор з більшою вірогідністю відкине ту локальну частину зображення, про яку він вже дізнався. Більш ефективний дискримінатор допомагає системі генерації зображень швидше рухатися до цільового рівня реалістичності.

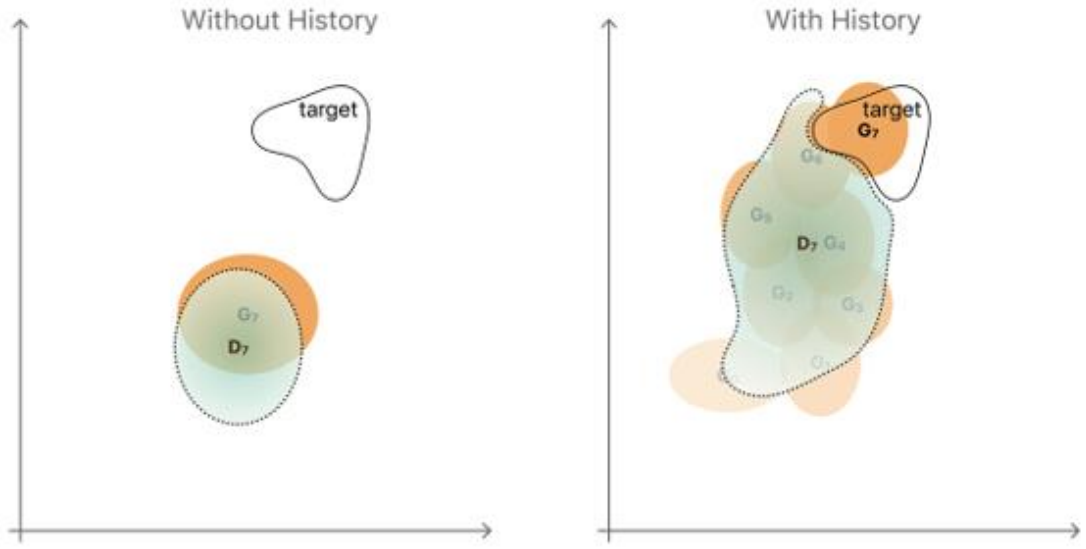

Рисунок 1.13 – Розподіл мережей генератора та дискримінатора з використанням історії та без

На практиці звичайний буфер випадкової заміни охоплює достатню різноманітність, щоб запобігти повторенню у контексті посилення дискримінатора. Тому сутність питання полягає в тому, що будь–яке вдосконалене зображення, що генерується мережею–покращувачем у будь–який час протягом усієї тренувальної процедури, є дійсно «підробленим» зображенням для дискримінатора. Було спостережено, що, побудувавши пакет для D з половиною зразків з історії та іншою половиною з виходу поточного генератора, ми можемо значно вдосконалити точність прогнозів. Однак подібний процес з використанням історії використовує надзвичайну кількість ресурсів, адже потрібно зберігати зліпки навченої системи

для абсолютно усіх етапів. В рамках класифікації зображень це безумовно може бути використано достатньо ефективно, але якщо намагатися використовувати відео (особливо з великим ступенем динамічності) для навчання системи, то це може виявитись нерентабельним.

Щодо того як проходить прогрес навчання подібної системи – по–перше, проходить предтренування для мережі–покращувача з втратами лише на внутрішню саморегуляцію, втрати на змагання з мережою–дискримінатором додаються лише тоді коли створені синтетичні зображення стають нечіткі та розмиті. На система він генерує розмите зображення, яке стає все більш реалістичним у міру проходження тренувань [12].

Отже, зрозумівши послідовність виконання операції додання реалізму та вдосконалення синтетичних зображень слід також розглянути додаткові інструменти, наприклад:

- розділення зображень на шари;
- обертання;
- вирівнювання гістограми – це технологія обробки зображень для збільшення глобального контрасту зображення за допомогою гістограми інтенсивності. Врівноважене зображення має лінійну кумулятивну функцію. Приклад вирівнювання гістограми можна побачити на рисунку 1.14;
- адаптивне вирівнювання гістограми (AHE – adaptive histogram equalization) покращує локальну контрастність зображення за рахунок обчислення декількох гістограм, що відповідають різним ділянкам зображення (відрізняється від звичайного вирівнювання, яке використовує лише одну гістограму для регулювання глобального контрасту) та використовує їх для локального регулювання. Однак AHE має тенденцію до надмірного посилення шуму у відносно однорідних областях зображення;
- зміна яркості;
- зміна контрастності та насиченості;

– наближення камери;

– деформація частин зображень;

– затінення.

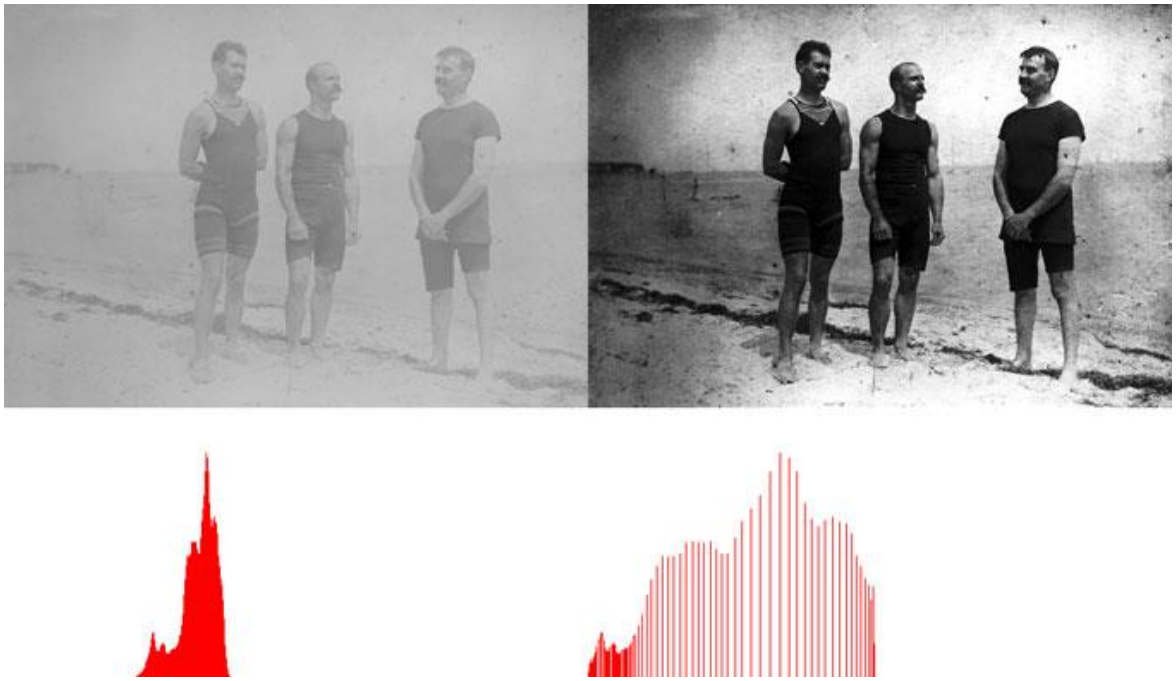

Рисунок 1.14 – Приклад використання вирівнювання гістограми

Не дивлячись на той факт що ми маємо повний контроль над похідними зображеннями, усе одно перелічені підходи можуть значно покращити результати навчання. Наприклад, якщо потрібно вибірково змінити лише частину зображення.

Одним з аспектів, котрий можна розглядати у якості додаткового механізму вдосконалення зображення – це їх комбінація. Наприклад, ми можемо згенерувати відокремлено від оточення цільові об’єкти для класифікації чи додати нові аспекти до зображення без рендерінгу нового. Така техніка може допомогти згенерувати додаткові дані для навчання чи повторного використання (наприклад, створення навколишнього оточення для узагальнених задач – місто, дороги, приміщення тощо).

Отже, на цьому етапі можна визначити 3 нейронні мережі, які потрібні для побудування більш оптимізованого процесу навчання:

– безпосередньо мережа розпізнавання;
– дискримінаційна мережа («змагається» з мережею–покращувачем);
– мережа–покращувач (намагається наблизити зображення до реалістичних).

Слід також зазначити, що однією з технік яка може суттєво вплинути на результати навчання системи є сегментація зображень. Процес сегментації зображення полягає у розділенні цільного зображення на частини – сегменти. Задача цієї операції – це спростити зображення, аби його було легше аналізувати (за допомогою масок, виділення контурів об’єктів тощо). В випадку генерації даних за допомоги ігрових движків ми маємо прямий доступ до процесу рендерінгу та інформації про стан та положення об’єктів. Саме тому, виконувати цю операцію слід на етапі генерації.

Це знов дозволяє не використовувати складні алгоритми комп’ютерного зору та дуже точно розділяти зображення на класи без артефактів. Маємо можливість легко змінювати матеріал об’єктів та генерувати, наприклад, унікальний колір для кожного з об’єктів. З точки зору роботи розглянутої системи генерації синтетичних даних інформація про результат сегментації має походити з мета–даних закріплених за зображенням, а алгоритми які слід застосовувати з частини системи де навчається мережа. У цьому контексті немає сенсу розглядати різноманітні підходи до сегментації зображень.

З цієї точки зору усе що потрібно для початку використання сегментованих зображень – це очікуване результуюче зображення. У тандемі з сегментацією можна також використовувати категоризацію (той самий процес, що і сегментація, але спільні за признаками об’єкти виділяються у один клас), зображення глибини та оптичного потоку (ці аспекти більш насущні для відео, адже вони відображають динаміку змін стану за допомогою відстеження контрастності та яркості, коли об’єкти пересуваються).

Приклад одного з способів розділити зображення на об’єкти є надання кожному об’єкту унікального кольору. Така текстура носить назву карти

ідентифікаторів. Початковий вигляд сцени до сегментації можна побачити на рисунку 1.15.

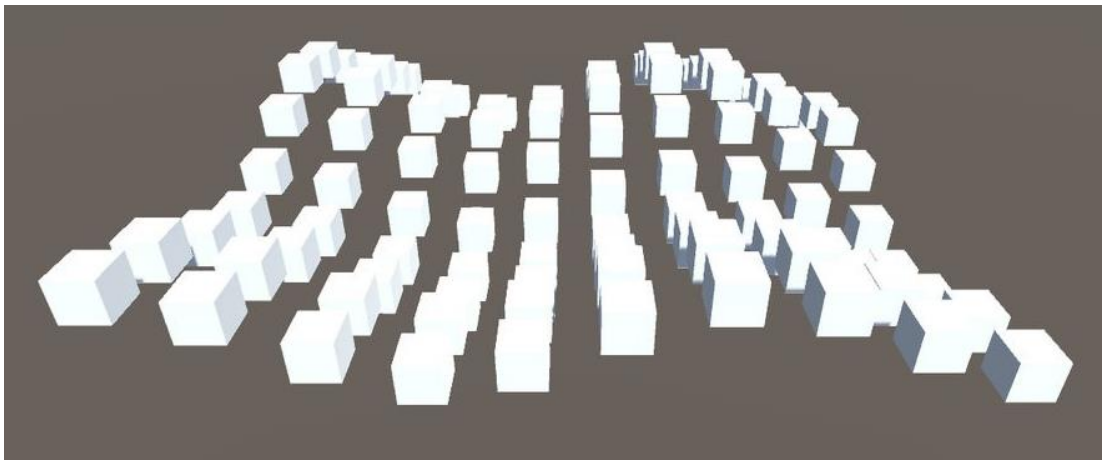

Рисунок 1.15 – Сцена до сегментації об’єктів

Використовуючи механізми системи генерації синтетичних зображень – маємо можливість встановити унікальний матеріал для кожного з об’єктів сцени. Результат цієї операції можна побачити на рисунку 1.16.

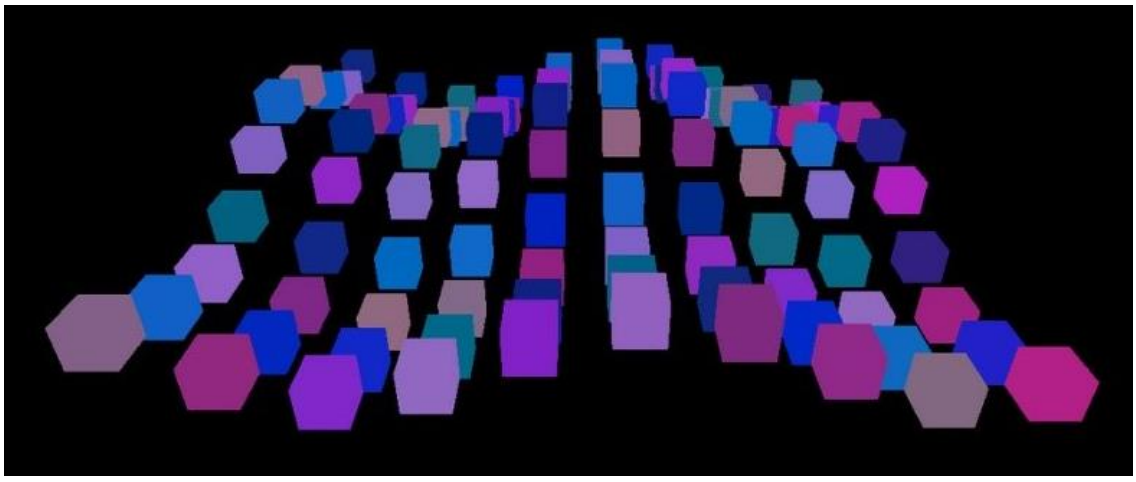

Рисунок 1.16 – Сцена після сегментації

У випадку використання категоризації для цього зображення кожен з об’єктів мав би один й той самий колір, адже вони відносяться до одного класу.

2 ОПИС РОЗРОБЛЕНОЇ ПРОГРАМНОЇ СИСТЕМИ

Для перевірки висунутих припущень була створена програмна система, яка навчається за обраним набором даних та дозволяє класифікувати зображення автомобілей. Послідовність взаємодії з генератором синтетичних даних можна побачити на рисунку 2.1.

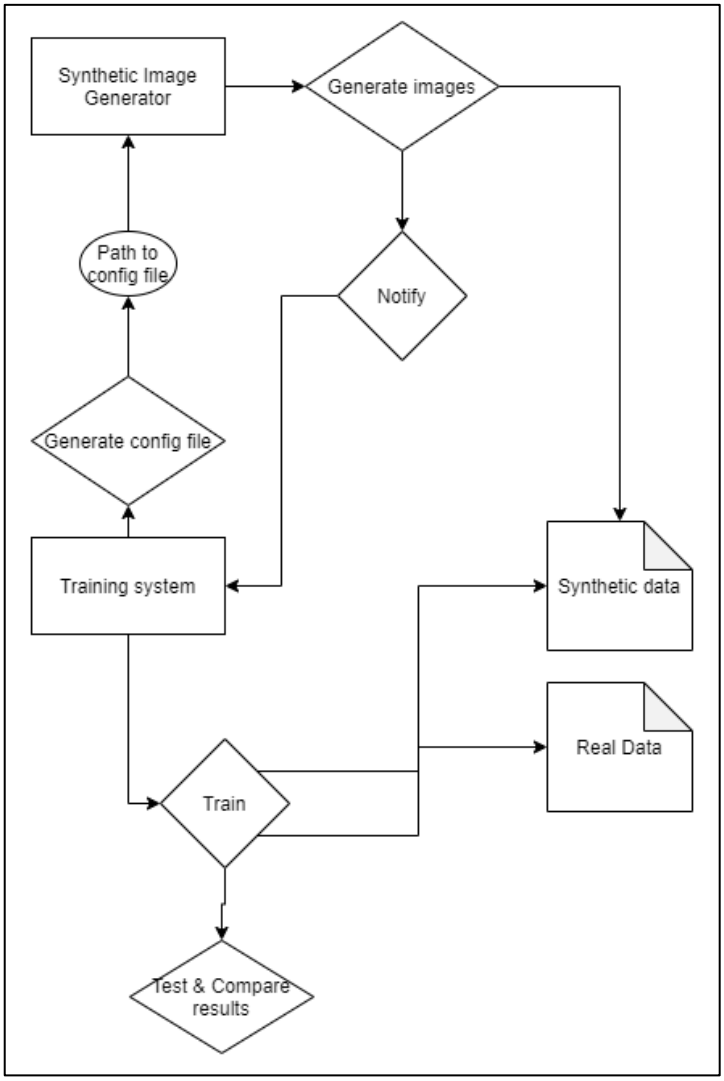

Рисунок 2.1 – Послідовність процесу навчання

Реалізація використовує Python, та алгоритмічних баз Tensorflow. На схемі зображено такі компоненти та елементи взаємодії між ними у рамках системи:

– Synthetic image generator – власне утиліта генерації зображень, яка виконує дії Generate Images – створення зображень та Notify сповіщення про помилку чи завершення завдання;
– Training system – компонент, модуль який власне викликає генератор, створює конфіг файл та передає його до генератору. На виході отримуємо треновану систему за синтетичними даними, яку ми перевіряємо за допомогою даних з реального світу.

Система, що тренується спочатку створює конфігураційний файл за яким будуть побудовані сцени для створення зображень. Далі вона передає шлях до цього файлу до генератора та він починає записувати зображення чи відео до файлової системи. Коли генератор закінчує своє виконання він сповіщає про це процес тренування та той починає власне навчання та тестову перевірку прогнозів класифікації за допомогою реальних даних та синтетичних. Приклад коду який виконує процедуру класифікації наведено нижче:

```
import tensorflow as tf
image_path = sys.argv [1]
# Read the image_data
image_data = tf.gfile.FastGFile (image_path, 'rb'). read ()
# Loads label file, strips off carriage return
label_lines = [line.rstrip () for line
                    in tf.gfile.GFile ( "logs / trained_labels.txt")]
# Unpersists graph from file
with tf.gfile.FastGFile ( "logs / trained_graph.pb", 'rb') as f:
    graph_def = tf.GraphDef ()
    graph_def.ParseFromString (f.read ())
    _ = Tf.import_graph_def (graph_def, name = '')
with tf.Session () as sess:
    # Feed the image_data as input to the graph and get first prediction
    softmax_tensor = sess.graph.get_tensor_by_name ( 'final_result: 0')
    predictions = sess.run (softmax_tensor, \
             { 'DecodeJpeg / contents: 0': image_data})
    # Sort to show labels of first prediction in order of confidence
    top_k = predictions [0] .argsort () [- len (predictions [0]):] [::
- 1]
    for node_id in top_k:
        human_string = label_lines [node_id]
        score = predictions [0] [node_id]
        print ( '% s (score =% .5f)'% (human_string, score))
```

Програма зчитує дані з папки, де знаходяться новостворені та тестові дані та класифікує їх за перед цим навченою багатошарову моделлю з використанням

CNN. Конфігурації, які передаються до генератора, використовують .yaml розширення файлів та визначають основні аспекти – наприклад, кількість згенерованих зображень.

Приклад вмісту такого файлу можна побачити нижче:

```
- step:
    name: train-model
    image: tensorflow/tensorflow:1.5.0-devel-gpu
    command:
      - ls /valohai/inputs/images/*.tgz | xargs -i tar -xzf {} -C
/valohai/inputs/images
      - python train.py {parameters}
    inputs:
      - name: images
    parameters:
      - name: epochs
        pass-as: --epochs={v}
        description: Number of steps to run the trainer
        type: integer
        default: 200
- step:
    name: worker-check
    image: tensorflow/tensorflow:1.5.0-devel-gpu
    command:
      - nvidia-smi
      - python --version
      - printenv
```

Також слід зазначити, що за допомогою цих даних налаштовується процес навчання на низькому рівні.

Тобто, приймається рішення використовувати центральний процесор чи графічний процесор для пришвидшення виконання програми, кількість потоків вона буде використовуватися, тощо.

У загальному сенсі процес взаємодії користувача (у нашому випадку це програміст, котрий буде користуватися вже навченою моделлю за допомогою створених зображень можна відобразити за допомогою Use-case діаграми.

Користувач розробник своєї системи, котра потребує використання машинного навчання та не має змоги отримати потрібні дані – обирає домен в якому йому потрібно створити зображення, додає власні ресурси, налаштовує сцену та конфігураційний файл за яким модуль-генератор синтетичних зображень буде власне створювати дані.

Власне діаграму основних способів взаємодії з програмною реалізацією можна побачити на рисунку 2.2.

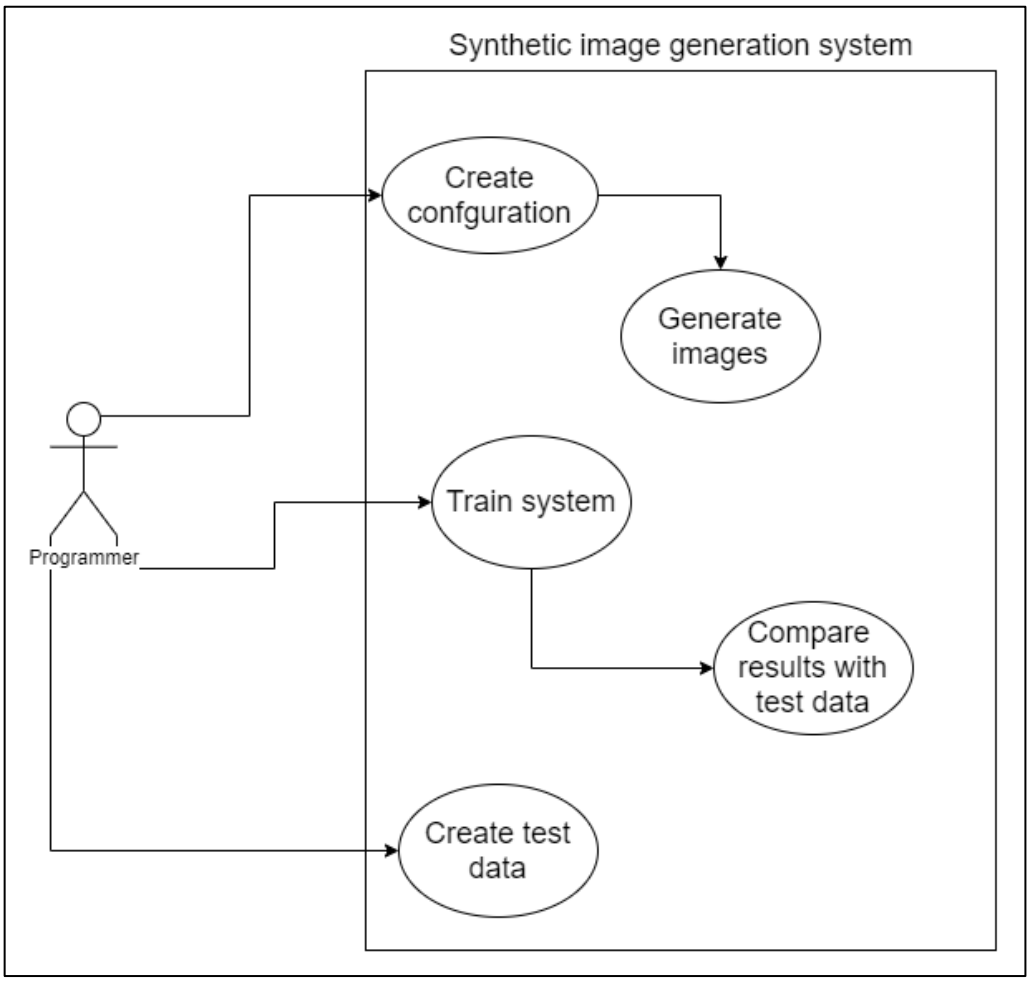

Рисунок 2.2 – Use case діаграма взаємодії користувача з системою

Можливими діями для користувача за схемою є:

– Create Configuration – можливість створити конфігурацію;

– Generate Images – запустити процес генерації за створеним файлом;

– Train System – запустити процес тренування за цими даними;

– Compare results with test data – порівняти результат з тестовими даними.

Також, суттєво зазначити, що цю утиліту можна використовувати, як модуль Jupyter Notebook. Велика кількість розробників використовує цей програмний продукт для створення демо та аналізу даних. Дана програмна система може

постачатися у якості розширення до нього. Приклад використання у якості розширення можна побачити на рисунку 2.3.

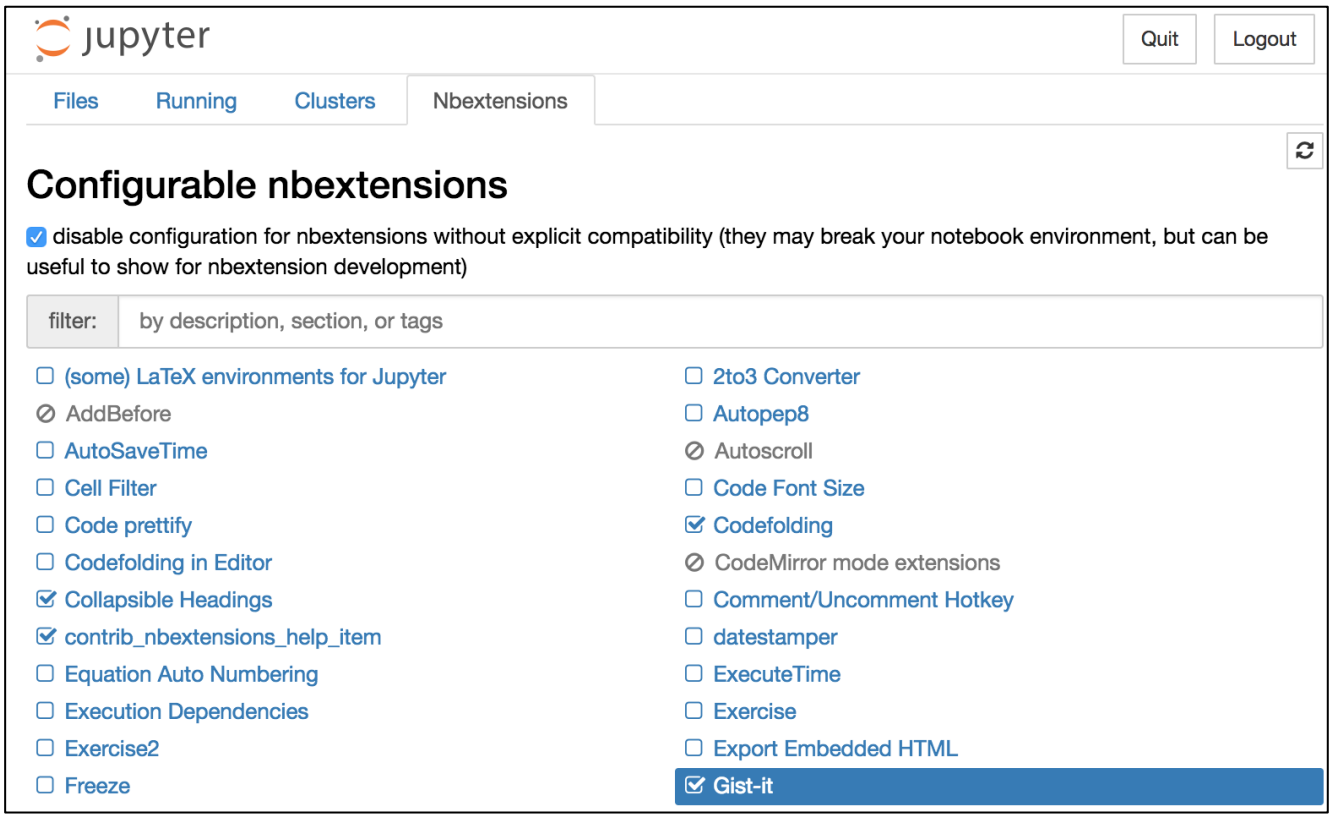

Рисунок 2.3 – Використання утиліти у якості розширення Jupyter Notebook

Отже, підбиваючи підсумки можна стверджувати, що була створена програмна система з декількох взаємодіючих модулів – генератор зображень та тренувальна система, процеси яких взаємодіють між собою підчас виконання та за допомогою конфігурацій. Можна ствердити, що подібну систему можна використати для підтвердження запропонованих гіпотез та проаналізувати результати дослідження.

3 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

3.1 Визначення метрик порівняння результатів

Отже, сформувавши практичний хід роботи для класифікації зображень, отриманих в результаті синтетичної генерації, можемо зробити заміри та порівняти з моделями навченими за даними з реального світу. Для цього візьмемо дані безпосередньо з CIFAR–10 бази (заздалегідь тренована модель), наприклад для відрізнення авто на зображенні. Але перед тим як використовувати отримані зображення їх слід підготувати, а саме:

- зчитати зображення з диска чи оперативної пам’яті процесу генерації;
- розшифрувати вміст цих зображень за форматом та перетворити їх у формат сітки відповідно до їх вмісту RGB;
- перетворити їх у матриці з плаваючою комою;
- зменшити масштаб значень матриці від значень 0 до 255 до значень між 0 і 1, оскільки нейронні мережі працюють краще з значно малими вхідними значеннями.

Ще одним з аспектів, який слід зазначити, якщо дане дослідження виявиться релевантним в аспекті генерації синтетичних даних, – це гнучкість у побудовані інфраструктури. Що мається на увазі, – це розмір розгорнутого движку, можливість це зробити на Linux–дистрибутивах операційної системи, тощо. Unity у цьому питанні значно програє, бо код не знаходиться у відкритому доступі та ми не можемо вирізати модулі якими ми не користуємося. У UE4 така можливість є. Для порівняння, розгорнутий Unity 2019.3.f1 версії важить переважно 4,74 гігабайти та у стані незапущеної сцени потребує 1,5 – 2 ГБ оперативної пам’яті, для UE4 – у оптимізованій версії движку ці значення можуть сягати 500 Мб оперативної пам’яті та місця на диску. Також, у рамках побудування оптимізованого рішення слід розглянути можливості обміну статусами програми, яка навчає модель та безпосередньо створює дані. Існує дві можливості налаштування з’єднання між

системами – дати простір у рамках операційної та файлової систем (що значно пришвидшить обмін інформацією) або за допомогою певного протоколу.

З точки зору оптимізації, саме перший варіант є оптимальним. Після того, як система відповідальна за навчання закінчує обробку зображення, вона робить скоригований запит з потрібними налаштуванням (наприклад прибрати світло чи додати новий ефект), але в цьому випадку ми не матимемо можливості масштабувати їх окремо. Для того щоб оцінити, наскільки добре працює модель, нам потрібні деякі види показників. Часто в машинному навчанні під час роботи з алгоритмами класифікації ми використовуємо точність (Accuracy) та відклик (Recall). Визначаючи ці показники, ми використовуємо такі терміни:

– true positive, справжнє позитивне (TP) – зображення, яке було вірно спрогнозоване та належне певному класу.
– true negative, справжнє негативне (TN) – зображення, вірно передбачене таким, що не належить до певного класу.
– false positive, помилково позитивне (FP) – зображення, неправильно передбачене, яке належне до певного класу.
– false negative, помилково негативне (FN) – зображення, неправильно передбачене, що не належить до конкретного класу.

Точність або Accuracy визначається, як кількість правильно класифікованих точок даних, поділених на загальну кількість точок даних, це наведено на рисунку 3.1.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Рисунок 3.1 – Визначення точності

Влучність або Precision – це міра правильних прогнозів класу, яка поділена на загальну кількість прогнозів для цього класу, що наведено на рисунку 3.2.

Високе значення точності вказує на малу кількість помилкових позитивних результатів.

$$Precision = \frac{TP}{TP + FP}$$

Рисунок 3.2 – Визначення влучності

Відгук або Recall – частина правильних прогнозів для класу, поділена на кількість загальних даних для цього класу, що наведено на рисунку 3.3.

$$Recall = \frac{TP}{TP + FN}$$

Рисунок 3.3 – Визначення відгуку

Високе значення відгуку може вказувати на малу кількість помилково–негативних передбачень під час класифікації зображення.

Отже, визначившись з метриками, за якими будуть оцінюватись результати експериментів, та інструментами для навчання моделі, у якості об’єкта класифікації зображення було обрано визначення пожежі в автомобілі, бо це має прикладне значення, адже отримати велику кількість даних для класифікації пожеж досить складно [9].

Для задачі порівняння ефективності моделі тренованої за реальними та синтетичними – порівняємо результати з заздалегідь навченою моделлю CERIF–10 у Tensorflow для детекції автомобілів. Також, для кожного з шагів буде порівнюватись час, який був потрібен для створення зображень. Це також напряму впливаэ на релевантність апробації дослідження, адже це напряму впливає на витрати на інтеграцію подібної системи.

Визначаючи найкращий метод створення синтетичних даних, важливо спочатку врахувати, який тип синтетичних даних ми прагнемо мати. Можна вибрати дві широкі категорії, кожна з яких має свої переваги та недоліки:

- повністю синтетичні – це дані, які не містять інформації з реального світу. Це означає, що повторна ідентифікація будь–якої однієї одиниці майже неможлива. Прикладом є наведений вище спосіб створення даних за допомогою рендерінгу у ігровому движку з використанням 3D оточення;
- частково синтетичні – дані, в яких замінюються тільки чутливі до класифікації об’єкти. Це призводить до зменшення залежностей моделі та не потребує широкої динамічності побудування системи генерації. Прикладом такої генерації може бути «вставка» синтетично створеного об’єкта в уже існуючу сцену.

Основною задачею обраного дослідження є саме отримання можливості створювати гіпотетичні ситуації, які досить складно та дорого або навіть практично неможливо, відтворити у реальному світі Через це, вибір у цьому питанні – це створення повністю синтетичних даних.

Саме тому, у якості інструмента генерації синтетичних даних буде використано, створена в іншій частині цього дослідження «Дослідження та оптимізація алгоритмів створення, рендерінгу та обміну синтетичними даними з системами машинного навчання», програмна система.

3.2 Порівняння системи з різними конфігураціями сцен

Отже, маючи інформацію про те, які зміни, до налаштувань процедури генерації синтетичних даних можуть впливати на точність прогнозу, будуть зроблені поступові зміни та кореляції у реальному часі.

В якості перевірки навченої моделі буде використано набір з 10 – 20 зображень обраних заздалегідь.

Змінні, які будуть контролюватись та оцінюватись у сцені є такими:

– кількість джерел світла;
– сумарна інтенсивність світла;
– типи тіней (Soft / Hard);
– використання запеченого чи світла у реальному часі;
– позиція та кут нахилу камери;
– наявність та кількість ефектів частинок;
– використанні чи невикористання декількох карт з текстур;
– різне розширення текстур;
– змішування при тренуванні додаткових реальних зображень;
– використання пост–обробки камери;
– використання результуючих зображень різного розширення.

Початковими були визначені такі значення для налаштування сцени:

– кількість джерел світла одночасно – 4;
– інтенсивність – ~8l;
– тип тіней – Soft;
– baked світло;
– UWRP, як пайплайн рендерінгу;
– ротація – x від –10 до 10;
– позиція камери у ігровому світі + або – (15,5,15);
– clipping planes = 0.3;
– тип камери = Perspective;
– використання Ambient occlusion карти – true;
– використання Normal карти – true;
– використання Roughness карти – true;
– використання пост–обробки – Volumetric Bloom – true, HDR – false;
– розширення зображення 1280x720;

– ефекти вогню та ефект диму в одному екземплярі з текстури 1024x1024;

– змінення навколишнього оточення – false.

Слід відмітити використання такого інструменту, як HDR. У стандартній візуалізації, значення червоного, зеленого та синього для пікселя представлений дробом у діапазоні 0..1, де 0 – являє нульову інтенсивність та 1 – являє собою максимальну інтенсивність для пристрою відображення.

Хоча, це легко використовувати, воно не відображає саме те, як працює освітлення на сцені з реального світу. Людське око має тенденцію пристосовуватися до місцевих умов освітлення, тому предмет, який у мало освітленій кімнаті виглядає білим, насправді може бути менш яскравим, ніж предмет, який при повному денному світлі виглядає сірим.

Крім того, людське око більш чутливе до різниці яскравості в нижньому кінці діапазону, ніж у високому кінці. Більш переконливих візуальних ефектів можна досягти, якщо візуалізація адаптована таким чином, щоб діапазони значень пікселів точніше відображали рівні світла, які були б наявними в реальній сцені.

Хоча, ці значення в кінцевому підсумку потрібно буде віднести до діапазону, доступного на пристрої відображення, будь–які проміжні розрахунки (наприклад, рендерінг зображення Unity) дадуть більш достовірні результати. Дозвіл внутрішнього представлення графіки використовувати значення поза діапазоном 0..1 – це є власне суть візуалізації високого динамічного діапазону (HDR).

Отже, для початку будуть порівнені результати стандартних налаштувань системи разом з зазделегідь навченою моделлю з реальними даними. Для цього випадку буди отримані такі результати експерименту.

Початкові результати для заздалегідь підготовленої моделі з Tensorflow можна побачити в таблиці 3.1. Вони були обрані, як еталонні.

Таблиця 3.1 – Метрики у заздалегідь навченої моделі

Data Set	Number of images	Accuracy %
1	300	~91

Кінець таблиці 3.1

Data Set	Number of images	Accuracy %
2	500	~93
3	700	~97

Результати експерименту при виконанні навчання нашої модель та валідації результатів можна побачити у таблиці 3.2.

Таблиця 3.2 – Початкові значення з використанням синтетичного набору

Data Set	Number of images	Accuracy %	Recall %	Render time (seconds)
1	300	~54	~97	~201
2	500	~57	~93	~289
3	700	~59	~95	~374

Можна легко побачити, що результати істотно різні. Спробуємо переконфігурувати змінні, які за нашими гіпотезами є найвпливовішими. Слід змінити розширення зображень до 1920x1080 та кількість зображень.

Результат наступного експерименту можна побачити у таблиці 3.3.

Таблиця 3.3 –Значення з вищим розширенням та кількістю зображень

Data Set	Number of images	Accuracy %	Recall %	Render time (seconds)
1	600	~65	~84	~531
2	800	~69	~73	~691
3	1000	~71	~81	~801

Точність прогнозів значно покращились, але за рахунок збільшення кількості зображень також майже вдвічі збільшився час рендерінгу та навчання моделі.

Наступним кроком спробуємо додати використання HDR для пост–обробки камери. Результат експерименту можна побачити у таблиці 2.4.

Таблиця 3.4 –Значення з використанням HDR

Data Set	Number of images	Accuracy %	Recall %	Render time (seconds)
1	300	~57	67	~269
2	500	~63	75	~399
3	700	~64	77	~516

Істотної різниці немає, але час генерації даних був збільшений на 34%. Робимо висновок, що HDR не грає високої ролі у контексті побудування нашої моделі, надалі не будемо його використовувати.

Спробуємо прибрати послідовно мапи AO, Normal та Roughness з усіх об’єктів у сцені.

Для відсутньої карти AO результати можна побачити у таблиці 3.5.

Таблиця 3.5 –Значення без використання карти AO

Data Set	Number of images	Accuracy %	Recall %	Render time (seconds)
1	300	~53	88	~162
2	500	~56	87	~213
3	700	~57	91	~354

Дійсно, через те, що сконфігурована сцена не має великих щілин у об’єктах (наприклад, як на камені) та не є дуже деталізованою, тому ця карта є надлишковою, у всіх наступних експерементах не будемо її використовувати.

Наступним кроком – для відсутньої карти Normal. Результати можна побачити у таблиці 3.6.

Таблиця 3.6 –Значення без використання карти Normal

Data Set	Number of images	Accuracy %	Recall %	Render time (seconds)
1	300	~42	67	~142
2	500	~44	54	~203
3	700	~48	64	~264

Можна побачити, що карта нормалей э важливим елементом текстури об`єктів. Її відсутність істотно впливає на точність прогнозу, тому надалі у всіх експериментах вона буде присутня.

Далі перевіримо наскільки впливає незначна зміна оточення навколо цільового об’єкта на показники навчання моделі. Результати можна побачити у таблиці 3.7.

Таблиця 3.7 –Значення з використанням динамічної зміни оточення

Data Set	Number of images	Accuracy %	Recall %	Render time (seconds)
1	300	~63	77	~221
2	500	~67	79	~275
3	700	~71	81	~361

Додавання різноманіття у оточення диверсифікує вихідні зображення та система стає більш адаптованою під зображення різного класу. Набір для валідації моделі відповідає цьому опису, тому надалі використовуємо саме таку конфігурацію.

Далі спробуємо змінити усі текстури на текстури з розширенням 2048x2048, що має надати додаткової деталізації у сцені. Результати експерименту можна побачити у таблиці 3.8.

Таблиця 3.8 –Значення з використанням текстур з розширенням 2048x2048

Data Set	Number of images	Accuracy %	Recall %	Render time (seconds)
1	300	~64	77	~401
2	500	~69	79	~580
3	700	~72	81	~899

Можна спостерігати, що точність прогнозу майже не змінилася, однак зарезервована оперативна пам`ять процесу генерації збільшилася майже на 50% та значно зменшилася швидкість генерації зображень. Якщо це не є проблемою для конкретної задачі – це є валідним способом покращення результатів, але в нашій ситуації ми надалі продовжимо використовувати текстури розширенням 1024x1024.

Щодо налаштувань позицій камери – дана опція повністю залежить від спостерігаємо об’єкту. Найкращих результатів навчання модель досягає, коли більше ніж 80% об’єкту знаходиться на зображенні. Досить різка зміна відстані камери від об’єкту допомагає покращити класифікацію на різних дистанціях, але лише за рахунок, якщо для кожного типу дистанції було згенеровано достатньо зображень.

Виділивши основотвірні параметри які безпосередньо впливають на результати навчання, наступним кроком застосуємо техніки вдосконалення та сегментації зображення.

У контексті вирішення задачі з детекції автомобіля та пожежі з точки зору сегментації слід відокремити границі об’єктів зображення – потрібно окреслити форму автомобіля, надати його основним часткам та навколишнім об’єктам унікальні кольори. Вогонь та дим мають мати також унікальний кольоровий градієнт у залежності до дистанції від джерела займання. Створення подібних сегментованих масок дозволить нам влучно додатково класифікувати такі аспекти як рівень небезпеки для водія, тощо. Дійсно, процес розділення зображення на

унікальні частини дозволив нам відкрити додаткові напрямки у аналізі зображення, а також безпосередньо дозволив пришвидшити процес навчання.

Однак, цей процес значно збільшив час на рендерінг зображень, бо для кожної копії зображення слід мати сегментовану версію. Час який знадобився для генерації цих даних можна побачити у таблиці 3.9.

Таблиця 3.9 – Значення з використанням маскування сегментованих зображень

Data Set	Number of images	Accuracy %	Recall %	Render time (seconds)
1	300	~62	74	~701
2	500	~76	76	~950
3	700	~78	81	~1069

Не дивлячись на значне збільшення часу генерації, сегментація зображень допомагає розкрити подібну систему у більш узагальненому вигляді та з новими можливостями. Тому її слід використовувати у всіх наступних експериментах в рамках дослідження.

З точки зору надання додаткового реалізму до зображенням програмним шляхом у рамках поставлених експериментів використано не було, адже це є окремим об’єктом для розгляду та не є цільовою задачею проведеного дослідження.

ВИСНОВКИ

Резюмуючи результати та хід проведеного дослідження у рамках даної роботи, слід зазначити, що було проведено аналіз різноманітних математичних та алгоритмічних рішень у контексті класифікації зображень. Серед типів нейронних мереж було обрано ефективну та релевантну в аспекті роботи з синтетичними візуальними даними, а саме – згорткову мережу. Через те, що за своєю суттю згорткова мережа може будуватися з шарів було побудовано таку модель, що підходила до обраної предметної області – визначення автомобіля та пожежі в автомобілі.

Були розглянуті різноманітні функції активації (що безпосередньо впливають на транзитивність даних між шарами), їх переваги та недоліки, було обрано оптимальний тип для нашого випадку. Додатково було розглянуті можливості додавання синтетичних об’єктів до зображень з реального життя.

Слід зауважити, що була створена програмна реалізація, яка дозволила експериментально підтвердити вплив усіх перелічених метрик на точність прогнозів та удосконалення якості навчання моделі. За допомогою створеної системи була апробована практична значимість цього дослідження, а саме можливість навчати низку моделей у конкретній предметній галузі за допомогою набору з повністю синтетичних даних.

У контексті інфраструктурних оптимізаційних рішень було проведено порівняння різних ігрових движків, їх можливостей у масштабуванні свого функціоналу заради покращення продуктивності.

У якості продовження роботи над цим дослідження, наступним кроком може стати балансування параметрів генерації під конкретні задачі, а також автоматизація цього процесу, інтеграція нових програмних рішень з точки зори рендерінгу або створення власного повністю адаптованого продукту, дослідження методів та механізмів удосконалення зображення під час його генерації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Medium. URL: https://medium.com (дата звернення: 01.05.2020).

2. Tensorflow. URL: https://www.tensorflow.org/ (дата звернення: 01.05.2020).

3. Apollo Auto Data Set. URL: http://apollo.auto/ (дата звернення: 01.05.2020).

4. mc.ai. URL: https://mc.ai/ (дата звернення: 01.05.2020).

5. A. Vechur, A. Chayka, Y. Chemodakov. The component method of scene analysis and object recognition: стаття. США, IEEE, 2003 - 5 с.

6. Richard Hoptroff, Lucy Mosquera, Khaled El Emam. Practical Synthetic Data Generation: книга. США, Каліфорнія: O'Reilly Media, Inc., 2020 – 543 с.

7. Aurélien Géron. Hands–on Machine Learning with Scikit–Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems: книга. США, Каліфорнія: O'Reilly Media, Inc., 2019 – 932 с.

8. А. Л. Єрохін, А. С. Нечипоренко, А. С. Бабій, О. П. Турута. Применение глубоких сверточных нейронных сетей для классификаци риноманометрических данных: стаття. Україна: ХНУРЕ, 2016 – 5 с.

9. Habibi Aghdam, Hamed, Jahani Heravi, Elnaz. Guide to Convolutional Neural Networks: A Practical Application to Traffic–Sign Detection and Classification: книга. США: Springer, 2017 – 308 с.

10. Ragav Venkatesan. Convolutional Neural Networks in Visual Computing: A Concise Guide: книга. США, Флоріда: CRC Press, 2018 – 198 с.

11. С. Ю. Шабанов, Ю. С. Новіков. Представление знаний сложного структурируемого объекта в задачах диагностирования с использованием моделей: стаття. Україна: Вісник НТУ «ХПІ», 2016 – 4 с.

12. Antonio Gulli, Sujit Pal. Deep Learning with Keras. Implementing deep learning models and neural networks with the power of Python: книга. Великобританія, Бірмінгем: Packt Publishing Ltd., 2017 – 318 с.