ДОДАТОК А

ПРИКЛАД ПРОГРАМНОГО КОДУ

```
import argparse
import glob
import json
import os
import random
import time

import numpy as np
import tensorflow as tf


def main(settings):

    # You can use any libraries in the Docker image or installed with pip.
print('numpy', np.__version__)
print('tensorflow', tf.__version__)

# You can access the inputs at / valohai / inputs / < input_name >
    images_dir = os.path.join(settings.inputs_dir, 'images')
image_count = len(glob.glob(os.path.join(images_dir, '*.png')))
print('image count: {0}'.format(image_count))

# Here you could do any pre - processing, training and validation.
# We are just mocking something
for simplicity.
iterations = int(settings.epochs / 20)
for i in range(0, iterations):
    epoch = (i + 1) * 20
accuracy = random.random()
print(json.dumps({
    'epoch': epoch,
    'accuracy': accuracy
}))
time.sleep(0.5)

# You can store anything writing it to / valohai / outputs
# Here we are are writing some mock numbers to CSV files but
# outputs can be anything e.g.HDF5 - files, images, audion, video.
print('writing weights and biases...')
weights_path = os.path.join(settings.outputs_dir, 'model-weights.csv')
with open(weights_path, 'w') as f:
    weights_list = [random.random() for _ in range(100)]
weights_str = ', '.join(map(str, weights_list))
f.write(weights_str)

biases_path = os.path.join(settings.outputs_dir, 'model-biases.csv')
with open(biases_path, 'w') as f:
    biases_list = [random.random() for _ in range(100)]
biases_str = ', '.join(map(str, biases_list))
f.write(biases_str)
print('done writing weights and biases')

def cli():
    parser = argparse.ArgumentParser()
parser.add_argument('--inputs_dir', type = str,
    default = os.getenv('VH_INPUTS_DIR', './inputs'))
parser.add_argument('--outputs_dir', type = str,
```

```
    default = os.getenv('VH_OUTPUTS_DIR', './outputs'))
parser.add_argument('--epochs', type = int,
    default = 200)
settings = parser.parse_args()
main(settings)


if __name__ == "__main__":
    cli()
-step:
    name: generate - images
image: tensorflow / tensorflow: 1.5 .0 - devel - gpu
command:
    -apt update - qq -
    apt install - qqq - y--no - install - recommends lwm -
    lwm &
    -cd / valohai / inputs / imgen - binary -
    tar - xzf.
/*.tgz
      - ./imgen.x86_64 -logfile
      - export DATETIME=$(date '+%Y-%m-%d-%H-%M-%S')
      - tar -cvzf imgen-images-$DATETIME.tgz ./*.png
      - mv imgen-images-$DATETIME.tgz /outputs/
    environment-variables:
      - name: VH_XORG
        default: "1"
      - name: IMGEN_TAKE_SCREENSHOTS
        default: "1"
      - name: IMGEN_LOOP
        default: "1"
    inputs:
      - name: imgen-binary

- 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
```

ДОДАТОК Б

СЛАЙДИ ПРЕЗЕНТАЦІЇ

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Атестаційна робота магістра

Дослідження та оптимізація методів класифікації зображень з використанням синтетичних даних

Науковий керівник:
к.т.н доц. каф. ПІ **Вечур О.В.**

Виконав:
Студент групи ІПЗмзд-18-1 **Вилегжанін С.С.**

2020

1

Мета дослідження

- Дослідити можливості та алгоритми класифікацій зображень з використанням синтетичних даних
- Запропонувати алгоритми та методи для вирішення задачі надання додаткового реалізму синтетичним зображенням
- Розглянути методи оптимізації при навчанні з синтетичними зображеннями

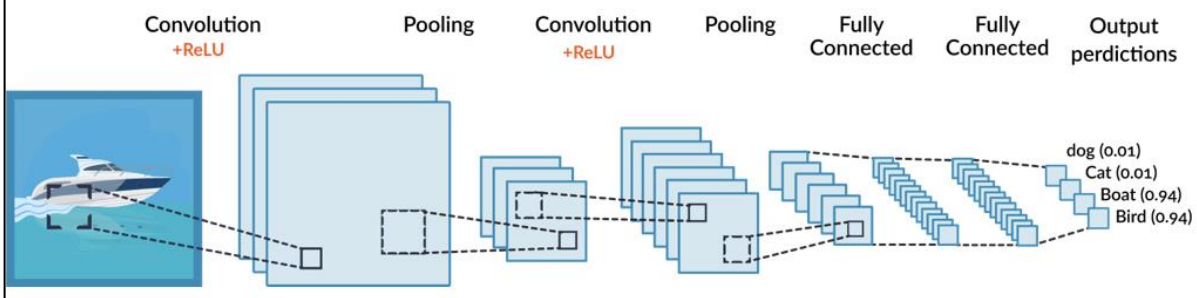
2

Практична релевантність та новизна

- Вирішити задачу конкрутентоспроможності використання синтетичних даних по відношенню до реальних даних для класифікації зображень.
- Знайти кореляції між налаштуваннями сцен та результатами навчання

3

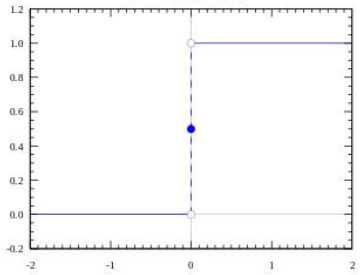
Згорткові мережі (CNN)



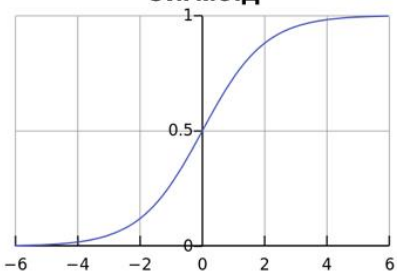
4

Функції активації

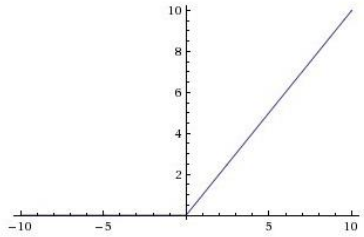
Шагова



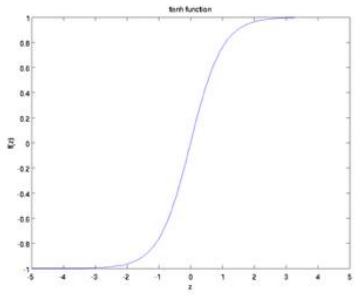
Сигмоїд



ReLU



Тан



5

Інструменти

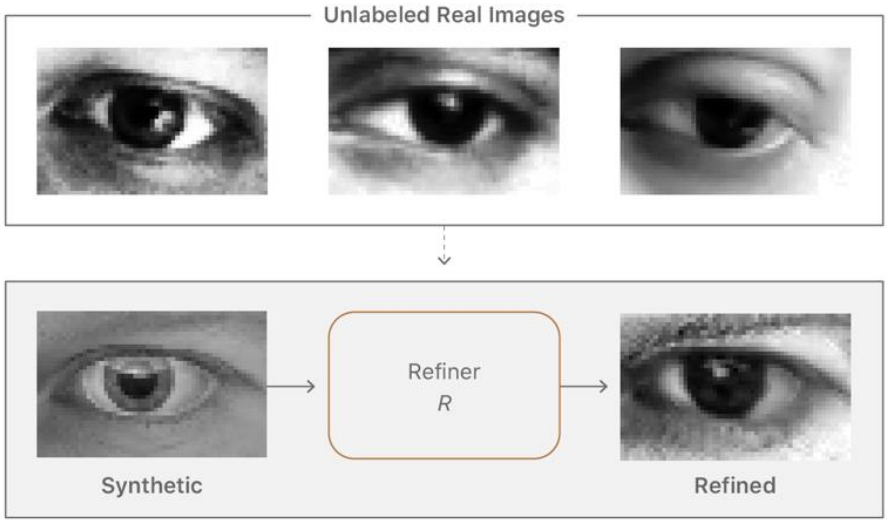
Приклад використання:

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

Tensorflow дозволяє легко інтегрувати CNN до будь-якого проекту та має зручний API для побудування багатошарових мережей

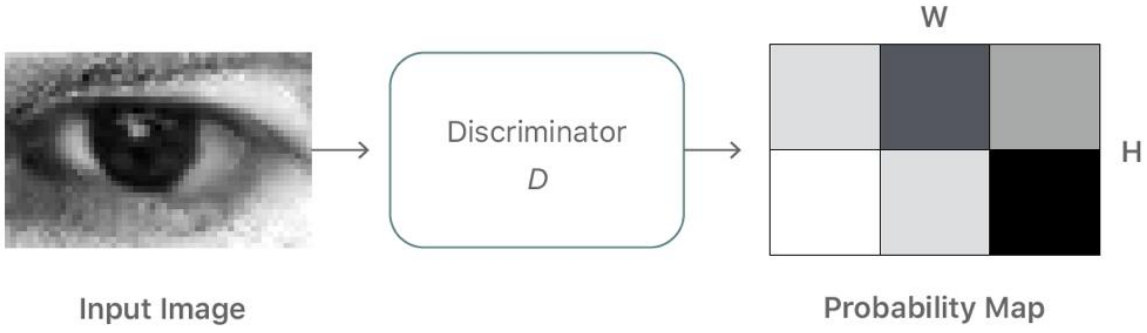
6

Вдосконалення синтетичних зображень



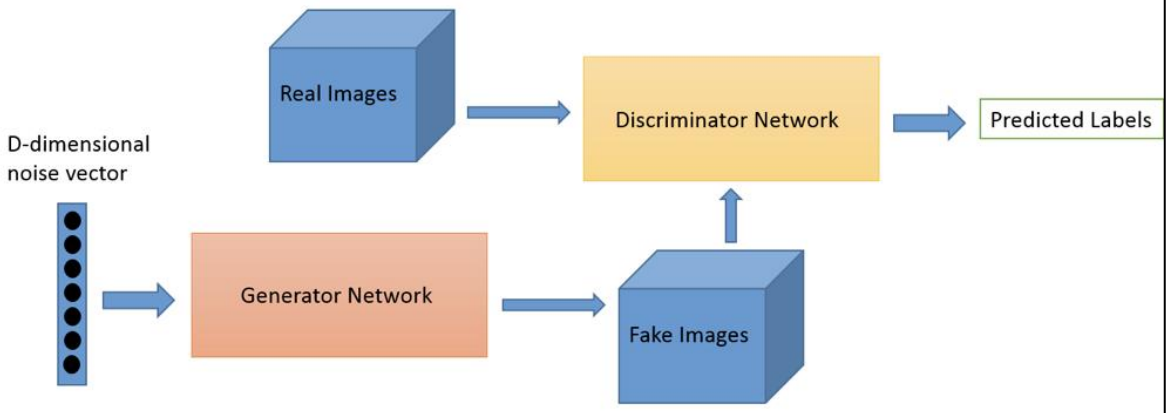
7

Запобігання артефактам



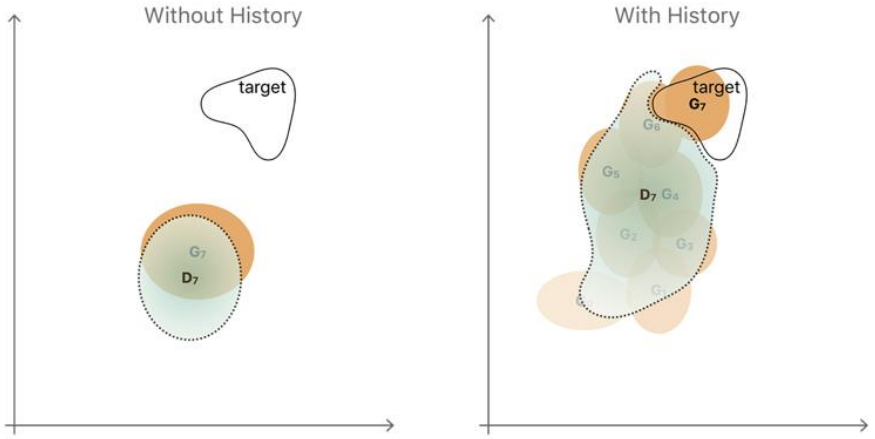
8

GAN



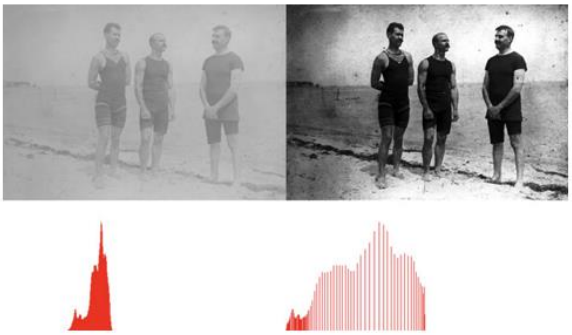
9

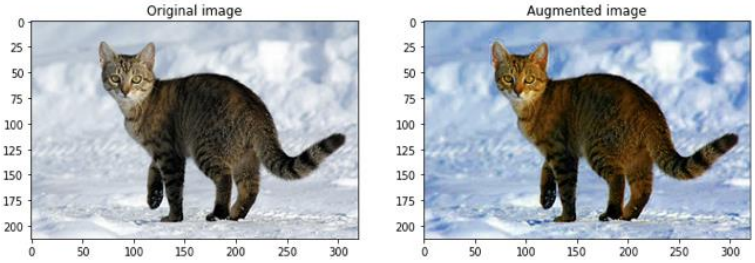
Навчання зі змаганням та використанням історії



10

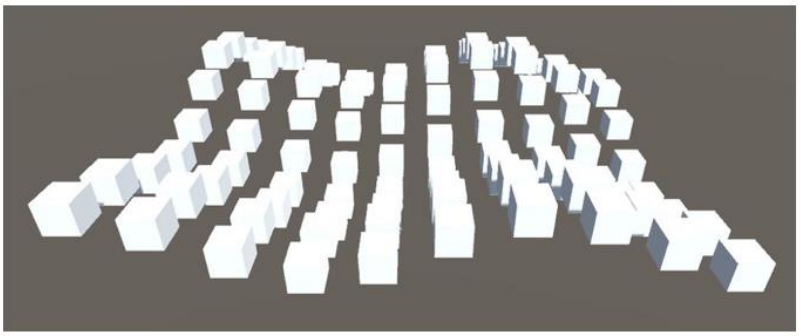
Інші методи вдосконалення

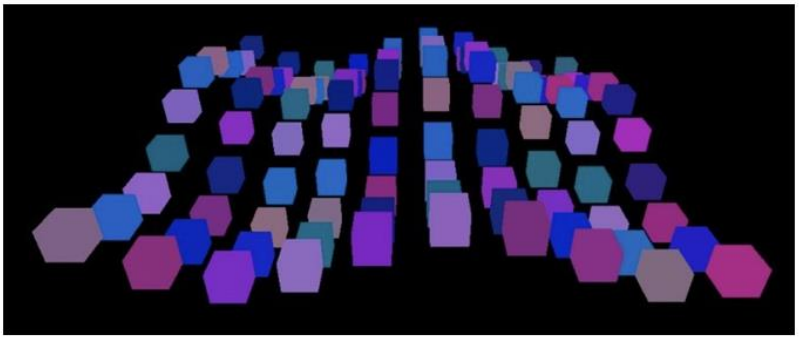




11

Сегментація





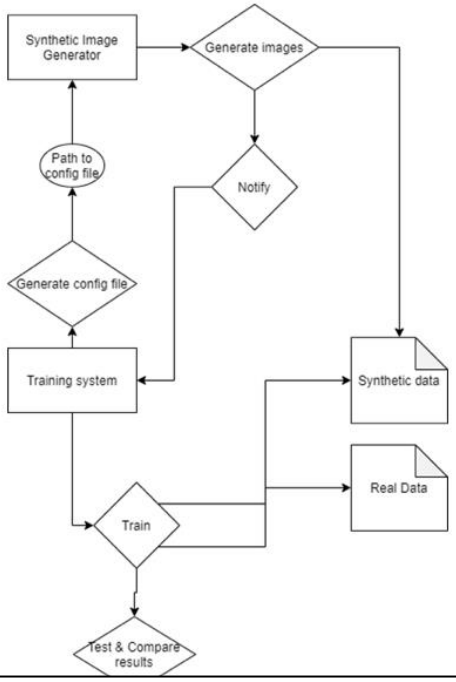
12

Використані метрики для оцінки результатів

- Точність - це міра правильних прогнозів класу, яка поділена на загальну кількість прогнозів для цього класу
- Відгук - частина правильних прогнозів для класу, поділена на кількість загальних даних для цього класу
- Швидкість рендерингу даних - кількість зображень на одиницю часу

13

Основні частини програмної системи



14

Результати прогнозів без оптимізацій

Data Set	Number of images	Accuracy %	Recall %	Render time (seconds)
1	300	~54	~73	~201
2	500	~57	~75	~289
3	700	~59	~77	~374

15

Результати з використанням зображень з більшим розширенням та кількістю даних

Data Set	Number of images	Accuracy %	Recall %	Render time (seconds)
1	600	~65	~74	~531
2	800	~69	~73	~691
3	1000	~71	~81	~801

16

Результати з використанням HDRP

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

17

Результати без використання карт нормалей та 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

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

18

Результати з використанням 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

19

Висновки

В ході дослідження були:

- Знайдені кореляції між налаштуванням генератора синтетичних зображень у ігровому движку та результатами навчання
- Вирішена задача вдосконалення синтетичних зображень з метою наблизи їх до реальних
- Виконано апробацію у вигляді публікації тезисів до конференції «Наукова думка сучасності і майбутнього»

20

ДОДАТОК В

АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ

СТВОРЕННЯ ТА РЕНДЕРІНГ СИНТЕТИЧНИХ ДАНИХ У ІГРОВИХ ДВИЖКАХ

naukam.triada.in.ua

Тихонова Олена Олександрівна

Студентка кафедри Програмної Інженерії

Харківський Національний Університет Радіоелектроніки

м. Харків

Вилегжанін Станіслав Сергійович

Студент кафедри Програмної Інженерії

Харківський Національний Університет Радіоелектроніки

м. Харків

Анотація: *Розгляд використання синтетичних даних, створених у ігрових движках для використанні при машинному навчанні.*

Ключові слова: *Синтетичний, штучний, зображення, текстура, карта, движок.*

З кожним днем сучасний світ змінюється та перетворюється. Технології «еволюціонують» та становляться все більш адаптовані під потреби конкретного користувача. Головним інструментом створення індивідуалізованого програмного і апаратного забезпечення стають системи машинного навчання та штучного інтелекту. Однак, щоб ефективно ними користуватися – потрібні дані, які б могли бути застосовані для вирішення поставленої задачі. Трапляється так, що виникають такі задачі, де дані для навчання систем, які зможуть їх вирішити, – або взагалі відсутні, або їх отримання може коштувати значних ресурсів. Прикладом однієї з таких задач може бути створення системи, яка буде аналізувати стан помешкання та визначати чи є загроза пожежі, чи ні. Для цього нам потрібні відео, власне з пожежами, але здобути їх у достатній кількості не є можливим, а відтворювати власноруч – дорого та непрактично. Механізмом, який допоможе розробнику подібних систем, стають інструменти генерації синтетичних даних. Синтетичні дані – це штучно створена інформація за допомогою певних алгоритмів та лімітів, а не з реальних подій. У рамках цієї роботи, буде розглянуто проблеми, інструменти та способи створення саме візуальних (зображення, відео) синтетичних даних. Якщо переглянути всі існуючі програмні продукти у домені графічних редакторів, систем рендерінгу, пакетів 3D моделювання та ігрових движків, найбільш яскравими виступають саме останні. Причин цьому велика кількість:

- можливість процедурно створювати сцени будь-якої складності;
- використання декількох камер та сесій одночасно;
- можливості зручно та динамічно налаштовувати параметри рендерінгу, світла та камери;
- широкий спектр оптимізаційних рішень для пошвидшення роботи;
- реалістична поведінка світла у сцені (PBR);
- ефекти пост-обробки для камери;
- створення ефектів з використанням GPU.

Тобто сучасні ігрові движки, які використовуються для створення фізично–реалістичних симуляцій у кіно та при створенні ігор – також ідеальні для створення псевдовипадкових сцен, бо мають можливості гнучко налаштовувати різноманітні параметри. Це дозволяє кожен раз генерувати нову ситуацію, що дає можливість наближатися до реалістичних результатів машинного навчання та покращувати точність прогнозів. Беручи до уваги актуальність цієї проблематики та релевантність ігрових движків та систем рендерінгу у її вирішенні, метою даних тез є пошук кореляцій різноманітних параметрів налаштування сцен та їх рендерінгу у ігровому движку при генерації синтетичних даних, а також рішень на етапі моделювання та текстурування з точністю прогнозів системи машинного навчання. Другою задачею є порівняння різних движків у цьому аспекті, а також оптимізація процесу генерації даних під певну ситуацію. Для безпосередньої перевірки результатів, є релевантним використанням популярних та перевірених часом систем та алгоритмічних баз, наприклад таких, як TensorFlow, які використовуються у великій кількості в сучасних проектах. Якщо результати покажуть достатню ефективність системи, то це доведе можливість створення більш точних моделей у випадку більш спеціалізованих програмних пакетів для машинного навчання.

Слід зазначити, що елементами наукової новизни у цьому питанні є систематизовані та зважені аспекти, які впливають на результати машинного навчання при створенні синтетичних даних, а також створення технік оптимізацій цього процесу. Практичним значенням цього нарямку є можливість впровадити системи штучного інтелекту та машинного навчання у такі домени життя, в яких наразі це неможливо. А також, застосування у наукових дослідженнях в певних галузях, покращення тестування вже існуючих програмних забезпечень та продуктів, яке дозволить розширити можливості розробників у створенні власних прототипів у обраній галузі з реального життя.

Основна проблема синтетичних наборів даних – це наблизити їх до подібності з реальним випадком використання, особливо

відео. Але справжні проблеми виникають, коли прогнози зосереджуються на дрібних деталях, таких як розпізнавання обличчя. Крім того, рендерінг довгих фотореалістичних відео або зображень може бути надзвичайно важкою операцією. Але це можна подолати, наприклад, згенерувавши карти сегментації. Моделі, які можуть бути навчені синтетичними даними, обмежені насиченістю та «багатством» даних, які ми можемо отримати. Що стосується зображень, які створені за допомогою програмного забезпечення для 3D рендерінгу, то є можливість повторення значної частини реального зображення завдяки науковому розумінню світла та фізики.

Розглянемо також деякі інші способи генерації візуальних синтетичних даних, окрім використання ігрових движків, щоб мати більш повну картину переваг саме цього методу. Вирізання та вставка одного зображення в інше виглядає як перспективна ідея в цьому плані. У цьому аспекті є можливість вирізати набір предметів та підставляти їх до різноманітних оточень. Проблема цього механізму полягає в тому, що це надзвичайно важко вирізати та вставити предмет достатньо природнім та реалістичним виглядом; тому це потребує додаткових модифікації зображення методами аугментації та вдосконалення. Це н надто важливо розміщувати об’єкти природньо в глобальному масштабі зображення, але важливо досягти локального реалізму. Сучасні класифікатори об’єктів не так критично, щоб мати об’єкт, наприклад на столі чи на підлозі; однак важливо змішати об’єкт якомога реалістичніше на локальному тлі. Навіть досить наївна вставка об’єктів може допомогти покращити виявлення об’єктів у системах машинного навчання, зробивши по суті синтетичні дані. Наступним кроком у цьому напрямку було б створити вставлені об’єкти бути узгодженими з геометрією та іншими властивостями сцени. У цьому контексті є такі окремі випадки, як наприклад – локалізація тексту, тобто виявлення об’єктів спеціально для тексту, що з’являється на зображенні. Однак, усі ці методи виглядають не достатньо ефективними та швидкими, адже потребують дуже специфічних підходів до їх імплементації. Саме

тому будемо розглядати саме ігрові движки у якості основного інструмента.

Отже, першочерговою задачею є вибір ігрового движка для створення тестових даних. Буде розглянуто лише open-source чи умовно безкоштовні рішення, бо доступ та ліцензії до комерційних та внутрішніх движків різних компаній відсутні. Движки, які працюють лише з 2D графікою (cocos2dx, godot та інші) у дослідженні розглядаться не будуть, бо поставлена задача полягає у розгляненні саме залежності фотореалістичних ефектів до результатів навчання моделей.

Основними рішеннями на ринку рішень для створення ігор є Unreal Engine 4 (далі – UE4, Unreal) та Unity. Ці движки є основними конкурентами та знаходяться у перегонах, хто з них може називатися найбільш developer-friendly. Вони дуже схожі за набором можливостей, але реалізація та пріоритети у них досить різні. Unity є рішенням більш загального призначення з точки зору крос-платформності, що не є вирішальним фактором у роглянутому випадку. UE4 -сфокусований на пристрої високого класу (PC, консолі PS та XBOX). Також, Unreal зарекомендував себе у створенні VR програм та візуальних ефектів у кіноіндустрії. Однак, однією з метрик за якою будуть оцінюватись результати є час рендерінгу та обробки зображення, тому Unity може бути швидшим у створенні даних. У цьому випадку UE4 може стати не гнучким та нерелевантним [3].

Отже, виділяємо метрики за якими будемо оцінювати навчання моделей з синтетични даними:

- точність прогнозу;
- девальвація від навчання моделей за реальними даними;
- швидкість створення даних;
- швидкість рендерінгу;
- розмір створених файлів;
- розмір одного «instance» для роботи системи (розмір вихідного коду движків може займати велику кількість місця).

Наступним етапом є виділення параметрів, які можуть впливати на результати у рамках доступних функцій движка, таких як:

- формати використаних моделей (OBJ, FBX тощо);
- деталізованість текстур;
- UV розгортка;
- розширення текстур;
- формати використаних текстур (PNG, TGA, JPEG);
- використання відео чи зображень;
- адаптивність під різні типи задач;
- використання PBR текстур;
- налаштування параметрів світла – Baked vs Real time;
- кількість джерел світла;
- налаштування постпроцесингу камери;
- налаштування позиції камери (зум, кути огляду, ширина об'єктиву);
- використання ефектів часток (підрахування на CPU / GPU);
- використовування різних пайплайнів рендерінгу (у Unity, наприклад, HDRP – High Definition Render Pipeline, LWRP – Low Weight Render Pipeline, UWRP – Universal Render Pipeline);
- параметри налаштування матеріалів.

З точки зору системи навчання моделі для класифікації зображення, розглянемо такі можливості та параметри:

- швидкість навчання;
- стиснення зображень;
- точність прогнозів (у порівнянні з іншими системами);
- можливість використовувати згорткові нейронні мережі;
- можливість сегментації зображень;
- можливість візуалізувати результати.

Розглянемо одне з цих питань. Динамічна зміна кількості полігонів можлива лише за наявністю декількох однакових моделей з різними рівнями деталізації. У загальному сенсі кількість полігонів не є важливим фактором з точки зору

класифікації зображення, бо навіть для низькополігональних моделей основна деталізація доводиться саме на текстури. Тому цей показник можна вважати вторинним. Деталізованість текстур визначається наближенням до вигляду з реального світу. Це досягається за допомогою використання PBR (Physically Based Rendering) – набору технік візуалізації, в основі яких лежить теорія, наближена до реальної теорії поширення світла.Для того, щоб рендер движка міг інтерпретувати текстуру,як фізично коректну, потрібно створити не тільки карту кольору (альбедо), але ще й карту нормалей, карту жорсткості, карту висоти, карту металевості та карту оклюзії. Аналіз та рендерінг з використанням цих карт займає набагато більше часу та пам'яті, ніж з використанням hand-paint, де світло та тіні вмальовуються на пряму в карту кольору. Але, якщо є потреба в саме реалістичному зображенні, то іншого способу, окрім використання PBR, у даний момент немає. Тому, слід проаналізувати етапи та структуру фізично коректного рендерінгу та знайти можливі оптимізації які можуть знадобитися у ході дослідження.Текстура (карта) альбедо визначає для кожного пікселя колір поверхні або базову відбивну здатність, якщо цей піксель металевий. Карта нормалей дозволяє поставити для кожного фрагмента поверхні свою нормаль, щоб створити ілюзію, що ця поверхня більш випукла. Зазвичай усі інші карти є похідними від карти нормалей та можуть бути згенеровані за її допомоги, бо саме вона надає основну візуальну форму та деталізацію. Карта жорсткості вказує, наскільки жорстка поверхня знаходиться в основі пікселя текстури. Обране значення з текстури жорсткості визначає статистичну орієнтацію мікрограней поверхні. На жорсткішій поверхні виходять більш широкі і розмиті відображення, тоді як на гладкій поверхні вони сфокусовані та чіткі. Карта жорсткості інтерпретується у якості хаотичного розподілу мікрограней серед яких частина світла поглинається. Можна сказати, що ці мікрограні представлені у якості маленьких зеркал серед яких розсіюється вхідний потік світла. Результатом такого розміщення є те, що промені світла розсіюються в різних напрямках на жорсткіших поверхнях, що призводить до більш яскравого блиску. І навпаки на гладких поверхнях променій з більшою вірогідністю відіб'ються в одному

або паралельних напрямках, що дасть менш різкий відблиск. Карта металевості визначає, чи є піксель металевим, чи ні. Залежно від того, як налаштований PBR движок, можна задавати металічність, як відтінками сірого, так і тільки двома кольорами: чорним та білим. Зазвичай ця карта визначає, то наскільки сильно об'єкт може відзеркалювати світло. Чим темніший піксель тим більше світла він відбиває, надаючи поверхні блиску. Карта висоти (також відома як відображення паралакса) є концепцією, аналогічною відображенню нормалей, однак цей метод є більш складним – та, отже, більш дорогим. Такі карти зазвичай використовуються разом з картами нормалей, та часто вони застосовуються для надання додаткової чіткості поверхням [4].

Таким чином були розглянуті методи створення синтетичних даних за допомогою ігрових движків, а також можливі фактори впливу на результати машинного навчання у інших системах в контексті налаштувань такої системи генерації.

Література:

1.Medium [Електронний ресурс] / Medium – Режим доступу. https://medium.com – Загол. з екрану (дата звернення: 19.03.2020).

2.Unity Docs [Електронний ресурс] / Unity Docs – Режим доступу. https://docs.unity3d.com/Manual/ – Загол. з екрану (дата звернення: 16.03.2020).

3.Pharr, Physically Based Rendering, Third Edition: From Theory to Implementation [Текст] / Matt Pharr – Morgan Kaufmann, 2016 – 1266 с.

4.Haines, Real–Time Rendering, Fourth Edition [Текст] / Eric Haines – USA, CRC Press – 1199 с.

5.UE4 Docs [Електронний ресурс] / UE4 Docs – Режим доступу. https://docs.unrealengine.com/en–US – Загол. з екрану (дата звернення:13.03.2020).