Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ТА ДОСЛІДЖЕННЯ ВИКОРИСТАННЯ KERAS IMAGEDATAGENERATOR ДЛЯ РОЗШИРЕННЯ ДАНИХ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-18-1

Улупов Г.С.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Кузьомін О.Я,
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Улупову Георгію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та дослідження використання Keras ImageDataGenerator для розширення даних
затверджена наказом університету від 16 травня 2022 року № 541 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 24 травня 2022р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, персональний комп'ютер на операційній системі Windows 10, мова програмування Python. середовище розробки Jupyter Notebook.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд методів збільшення даних.
2. Архітектура моделі для розширення даних за допомогою Keras.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми розширення даних за допомогою Keras ImageDataGenerator,

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

No з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	12.04.2022	
2	Аналіз завдання, підбір літератури	13.04.22-15.04.22	
3	Аналіз літератури з досліджуваної проблеми	16.04.22-17.04.22	
4	Аналіз підходів для розширення даних за допомогою використання Keras ImageDataGenerator	26.04.22-30.04.22	
5	Розробка методу розширення даних за допомогою використання Keras ImageDataGenerator	01.05.22-14.05.22	
6	Програмна реалізація	15.05.22-23.05.22	
7	Оформлення пояснювальної записки	24.05.22-26.05.22	
8	Перевірка на плагіат	06.06.22	
9	Рецензування	06.06.22	
10	Підготовка презентації та доповіді	06.06.22-14.06.22	
11	Занесення роботи в електронний архів	14.06.22	
12	Попередній захист кваліфікаційної роботи	14.06.22	

Дата видачі завдання 18 квітня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Кузьомін О.Я,_
(підпис) (посада, прізвище,ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 62 с., 1 табл., 17 рис., 16 ліст., 25 джерел.

PYTHON, РОЗШИРЕННЯ ДАНИХ, IMAGEDATAGENERATOR, КОМП'ЮТЕРНИЙ ЗІР, KERAS.

Об’єктом роботи є дані які будуть збільшені.

Метою роботи є розширення даних за допомоги використання Keras ImageDataGenerator.

Використано методи розширення даних за допомогою Keras DataImageGenerator. Проведено дослідження методів створення набору даних і розширення наявного набору даних. Досліджено метод збільшення даних на місці або на льоту. Проведено дослідження методу поєднання генерації наборів даних і розширення на місці.

У результаті роботи здійснена три експерименти зі збільшенням даних даних за допомогою Keras.

PYTHON, DATA EXTENSION, IMAGEDATAGENERATOR, COMPUTER VISION, KERAS.

The object of the work is the data that will be increased.

The aim of the work is to expand the data by using Keras ImageDataGenerator.

Data extension methods using Keras DataImageGenerator are used. A study of methods for creating a data set and expanding the existing data set. The method of increasing data on the spot or on the fly has been studied. A study of the method of combining data set generation and field expansion was performed.

As a result, three experiments were performed to increase data data using Keras.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CNN − convolutional neural network (згорткова нейронна мережа)

RNN − recurrent neural network (рекурентна нейронна мережа)

NN − neural network (нейронна мережа)

HSV − hue saturation value (колірний тон, насиченість, значення кольору)

ВСТУП

Збільшення даних охоплює широкий спектр методів, що використовуються для створення «нових» навчальних вибірок із вихідних шляхом застосування випадкових тремтінь і збурень (але водночас гарантуючи, що мітки класів даних не змінюються).

Мета цієї роботи при застосуванні аугментації даних — підвищити узагальненість моделі. У цьому нам буде допомагати бібліотека Keras. Keras - це високорівневий API нейронної мережі, Keras написаний на чистому Python і заснований на бекенд Tensorflow, Theano і CNTK [1]. Keras був створений для підтримки швидких експериментів і може швидко перетворювати наші ідеї на результати. Треба використовувати Keras якщо в нас є наступні вимоги:

− просте та швидке прототипування (keras має високу модульність, мінімалістичність і розширюваність);

− підтримує CNN та RNN або їх комбінацію;

− плавне перемикання ЦП та ГП.

Актуальність роботи полягає у тому що ми при застосуванні аугментації даних, підвищуємо узагальненість моделі. Враховуючи, що наша мережа постійно отримує нові, дещо змінені версії вхідних даних, мережа може вивчати більш надійні функції. Під час тестування не застосовується збільшення даних, а просто оцінюється навченість мережі на основі не модифікованих даних тестування — у більшості випадків буде підвищення точності тестування, можливо, за рахунок невеликого зниження точності навчання.

1 ЗАДАЧА ЗБІЛЬШЕННЯ ДАНИХ

1.1 Концепція збільшення даних

Адріан Роузброк проводив опитування щодо розуміння людей концепції збільшення даних. Він запитував що з наведеного впливає на збільшення даних?

− Додавання більше тренувальних даних;

− зміна навчальних даних;

− обидві відповіді;

− не знаю.

Результати опитування щодо концепції збільшення даних показано на рисунку 1.1.

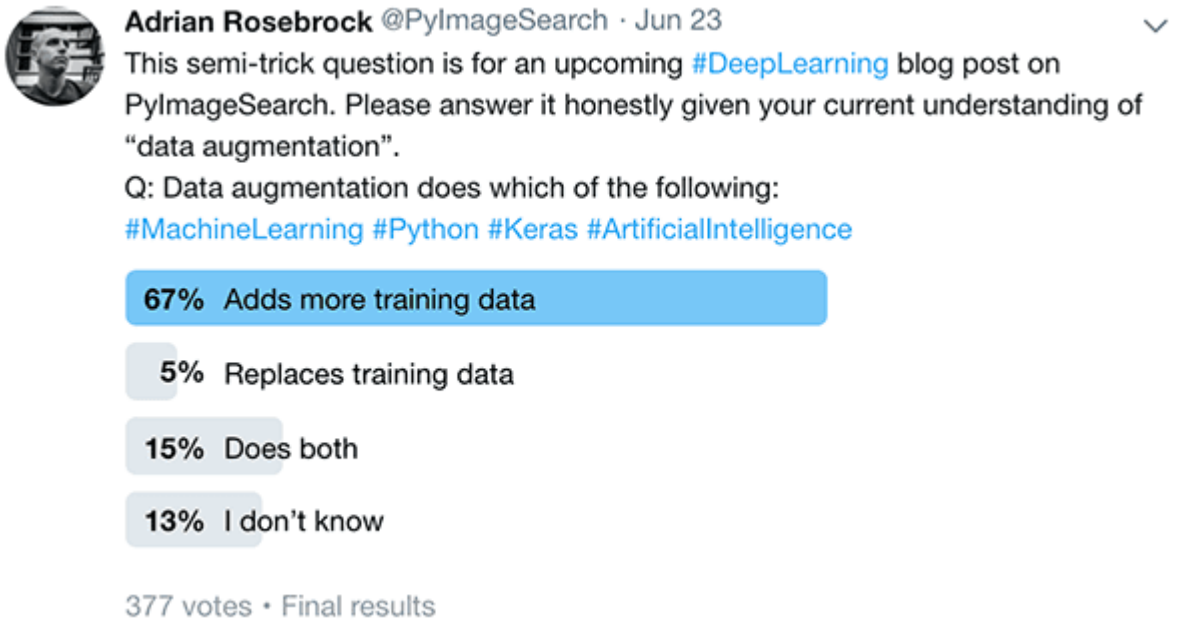

Рисунок 1.1 – Результати опитування щодо концепції збільшення даних

Лише 5% респондентів відповіли на це підступне питання «правильно» (принаймні, якщо ви використовуєте клас ImageDataGenerator Keras). Знову ж таки, це підступне питання, тому це несправедлива оцінка, але ось угода: Хоча слово «доповнити» означає зробити щось «більше» або «збільшити» щось (у цьому випадку дані), клас Keras ImageDataGenerator насправді працює за допомогою:

– прийняття партії зображень, які використовуються для навчання;

– взяти цей пакет і застосувати серію випадкових перетворень до кожного зображення в пакеті (включаючи випадковий поворот, зміну розміру, зріз тощо);

– заміна вихідної партії новою, випадково перетвореною партією;

–навчання CNN на цьому випадково перетвореному пакеті (тобто самі вихідні дані не використовуються для навчання).

Клас Keras ImageDataGenerator не є «адитивною» операцією. Він не бере вихідні дані, випадково перетворює їх, а потім повертає як вихідні дані, так і перетворені дані.

Натомість ImageDataGenerator приймає вихідні дані, випадковим чином перетворює їх і повертає лише нові, перетворені дані.

Технічно всі відповіді правильні, але єдиний спосіб дізнатися, чи правильне дане визначення збільшення даних, — це контекст його застосування.

Щоб дізнатися більше про розширення даних, у тому числі про використання класу ImageDataGenerator Keras, було проведено це дослідження.

1.2 Що таке збільшення даних

Збільшення даних охоплює широкий спектр методів, що використовуються для створення «нових» навчальних вибірок із вихідних

шляхом застосування випадкових тремтінь і збурень (але водночас гарантуючи, що мітки класів даних не змінюються).

Наша мета при застосуванні аугментації даних — підвищити узагальненість моделі.

Враховуючи, що мережа постійно отримує нові, дещо змінені версії вхідних даних, мережа може вивчати більш надійні функції.

Під час тестування не застосовується збільшення даних, а просто оцінюється навченість мережі на основі не модифікованих даних тестування — у більшості випадків є можливість помітити підвищення точності тестування, можливо, за рахунок невеликого зниження точності навчання.

1.3 Приклад збільшення даних

Розглянемо рисунок 1.2 на ньому можна побачити дві вибірки: ліворуч вибірка з 250 точок даних які відповідають нормальному розподілу, а праворуч вибірка з 250 точок даних які відповідають нормальному розподілу з додаванням невеликої кількості випадкового «джиттера» [5].

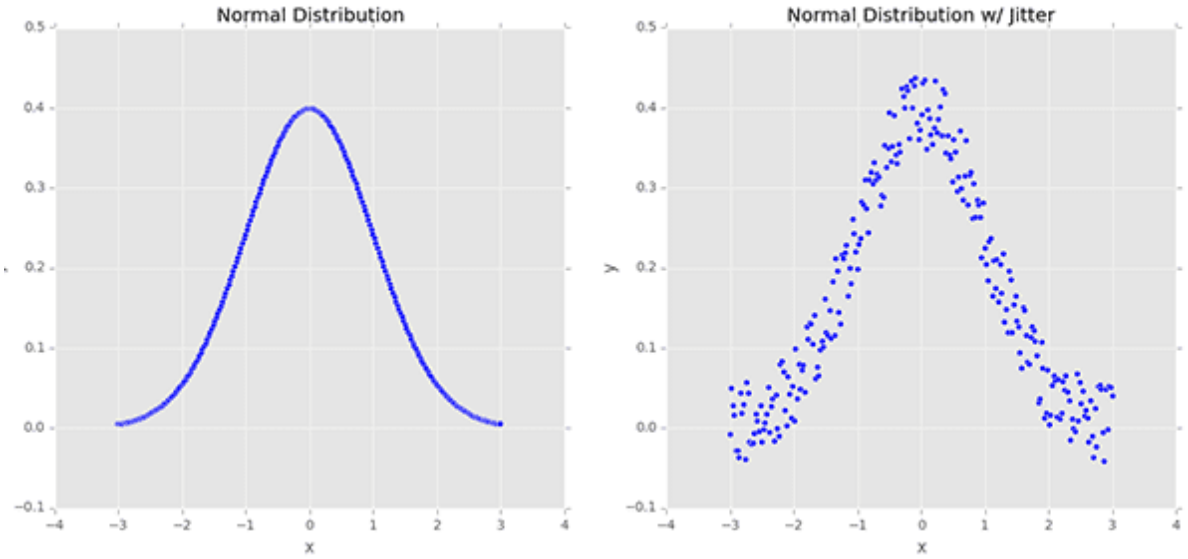

Рисунок 1.2 – Вибірка нормального розподілу та вибірка нормального розподілу з додаванням «джиттера».

Розглянемо вибірку без додавання «джиттера» на рисунку 1.2 нормального розподілу з нульовим середнім значенням та одиницею дисперсії. Навчання моделі машинного навчання на цих даних може призвести до того, що буде можливість точно змоделювати розподіл, однак у реальних програмах дані рідко слідують за таким гарним, акуратним розподілом.

Замість цього, щоб підвищити узагальненість класифікатора, можемо спочатку випадковим чином змінювати точки вздовж розподілу, додаючи деякі випадкові значення ∈ взято з випадкового розподілу (праворуч).

Після цього графік все ще має приблизно нормальний розподіл, але це не ідеальний розподіл, як ліворуч.

Модель, навчена на цих модифікованих, доповнених даних, швидше за все, буде узагальнюватися на приклади точок даних, які не включені в навчальний набір.

1.4 Постановка задачі

Об’єктом роботи є дані які будуть збільшені.

Метою роботи є розширення даних за допомоги використання Keras ImageDataGenerator. Будуть розглянуті основні методи та типи збільшення даних. Також будуть проведені експерименти з використанням Keras DataImageGenerator для розширення даних із застосуванням аугментації даних, для підвищення узагальненості моделі.

У ході експериментів буде створений згенерований набір даних, після чого будуть використані три типи збільшення даних, та два скрипти Python.

Під час експерименту планується використовувати среду розроботки Jupyter Notebook, та мову програмування Python. Реалізація ЗМН буде використана завдяки бібліотеки TensorFlow в тандемі з Keras.

2 ОСНОВНІ МОДЕЛІ ЗБІЛЬШЕННЯ ДАНИХ

2.1 Комп'ютерний зір і збільшення даних

У комп’ютерному зорі аугментація даних виконує випадкові маніпуляції із зображеннями. Зазвичай він застосовується в трьох сценаріях, які обговорюються в цій роботі.

У контексті комп’ютерного зору збільшення даних є природним.

Наприклад, можливо отримати доповнені дані з вихідних зображень, застосовуючи прості геометричні перетворення, такі як випадкові:

– переклади;

– обертання;

– зміни масштабу;

– стрижка;

– горизонтальні (а в деяких випадках і вертикальні) перекидання.

Застосування невеликої кількості перетворень до вхідного зображення дещо змінить його зовнішній вигляд, але не змінить мітку класу, що робить збільшення даних дуже природним і легким методом для задач комп’ютерного зору.

2.2 Типи збільшення даних

Існує три типи розширень даних, з якими є можливість зіткнутися, застосовуючи поглиблене контекстне навчання до програм комп’ютерного зору.

Яке саме визначення збільшення даних є «правильним», повністю залежить від контексту проекту/набору експериментів.

2.2.1 Створення набору даних і розширення набору даних

Розширення даних складається з генерування/розширення набору даних. Це менш поширена форма збільшення даних (рис. 2.1).

Перший тип розширення даних - це те, що називають генерацією набору даних або розширенням набору даних .

Моделі машинного навчання, і особливо нейронні мережі, можуть вимагати досить багато навчальних даних, але що робити, якщо не має дуже багато навчальних даних?

Буде розглянуто найтривіальніший випадок, коли є лише одне зображення , і потрібно застосувати розширення даних для створення цілого набору зображень на основі цього одного зображення.

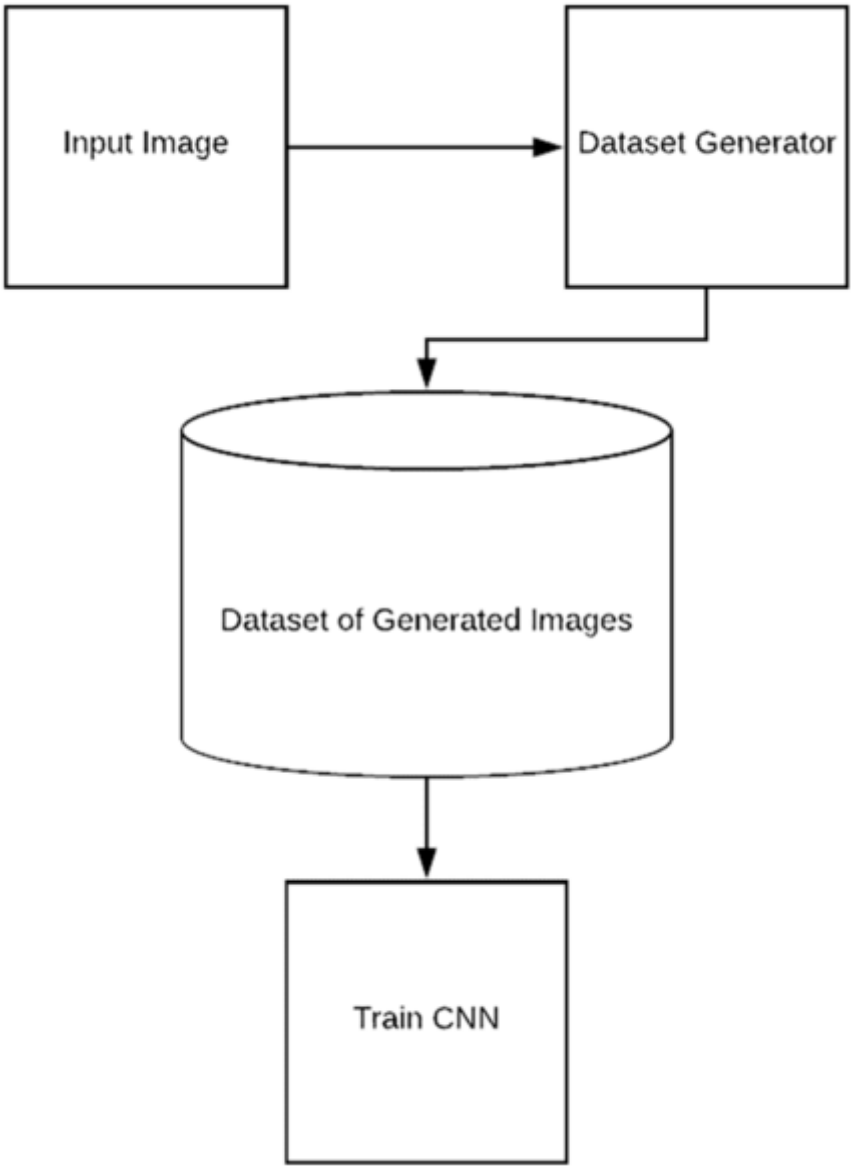

Рисунок 2.1 – Розширення з генеруванням/розширенням набору даних

Щоб виконати це завдання, необхідно по-перше завантажити вихідне вхідне зображення з диска, потім треба довільно трансформувати вихідне зображення за допомогою серії випадкових перекладів, поворотів тощо. Після цього треба взяти перетворене зображення та записуємо його назад на диск. Другий та третій крок треба повторити загалом N разів.

Ймовірність того, що буде більше однієї фотографії, висока – ймовірно, їх більше 10 чи 100, і тепер постає мета — створити цей менший набір із 1000 фотографій, щоб можливість його вивчати.

У таких ситуаціях, можливо, варто дослідити розширення набору даних і створення набору даних.

Але з цим підходом є проблема — здатність моделі до узагальнення, була збільшена не точно.

Так, були збільшені навчальні дані, завдяки створенню додаткових прикладів, але всі ці приклади базуються на дуже маленькому наборі даних.

Треба мати на увазі, що нейронна мережа настільки хороша, як і дані, на яких вона була навчена.

Не можливо розраховувати на навчання NN на невеликій кількості даних, а потім очікувати, що вона узагальнить дані, на яких вона ніколи не навчалася і яких ніколи раніше не бачила.

Якщо стоїть задача про створення набору даних і розширення набору даних, слід зробити крок назад і замість цього витратити час на збір додаткових даних або пошук методів поведінкового клонування (а потім застосування типу розширення даних, описаного в розділі «Поєднання генерації наборів даних та збільшення на місці»).

2.2.2 Збільшення даних на місці/на льоту

Другий тип розширення даних складається з пакетних маніпуляцій із зображеннями на льоту. Це найпоширеніша форма збільшення даних за допомогою Keras. Цей тип даних називається розширенням даних на місці або збільшенням даних на льоту . Другий тип збільшення даних є те, що Керас ImageDataGenerator інструменти класу.

Використовуючи цей тип розширення даних, є можливість переконатися, що мережа, коли її навчають, бачить нові варіації наших даних у кожну епоху.

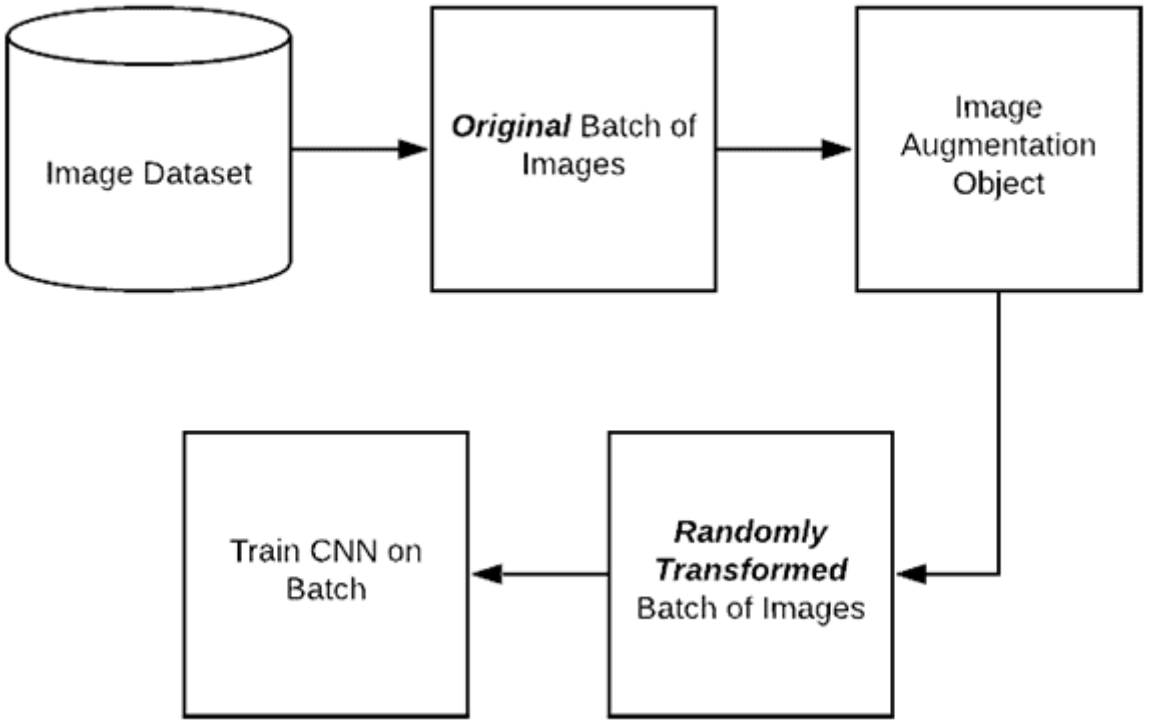

Рисунок 2.2 – Процес застосування додавання даних на місці

Крок № 1: вхідна партія зображень представлена до ImageDataGenerator

Крок №2 : ImageDataGenerator перетворює кожне зображення в пакеті за допомогою серії випадкових перекладів, поворотів тощо.

Крок №3: випадково перетворений пакет потім повертається до функції виклику.

Є два важливі моменти, на які треба звернути увагу:

– the ImageDataGenerator не повертає як вихідні дані, так і перетворені дані - клас повертає лише випадково перетворені дані;

– це називається збільшенням даних «на місці» та «на льоту», оскільки це збільшення здійснюється під час навчання (тобто не генеруємо ці приклади завчасно/перед навчанням).

Коли модель навчається, є можливість думати про ImageDataGenerator клас як «перехоплення» вихідних даних, випадкове їх перетворення, а потім повернення в нейронну мережу для навчання, при цьому NN поняття не має, що дані були змінені.

Багато людей вважають, що клас ImageDateGenerator Keras є «адитивною операцією», схожою на наступну (неправильну) фігуру, яка зображена на рисунку 2.3

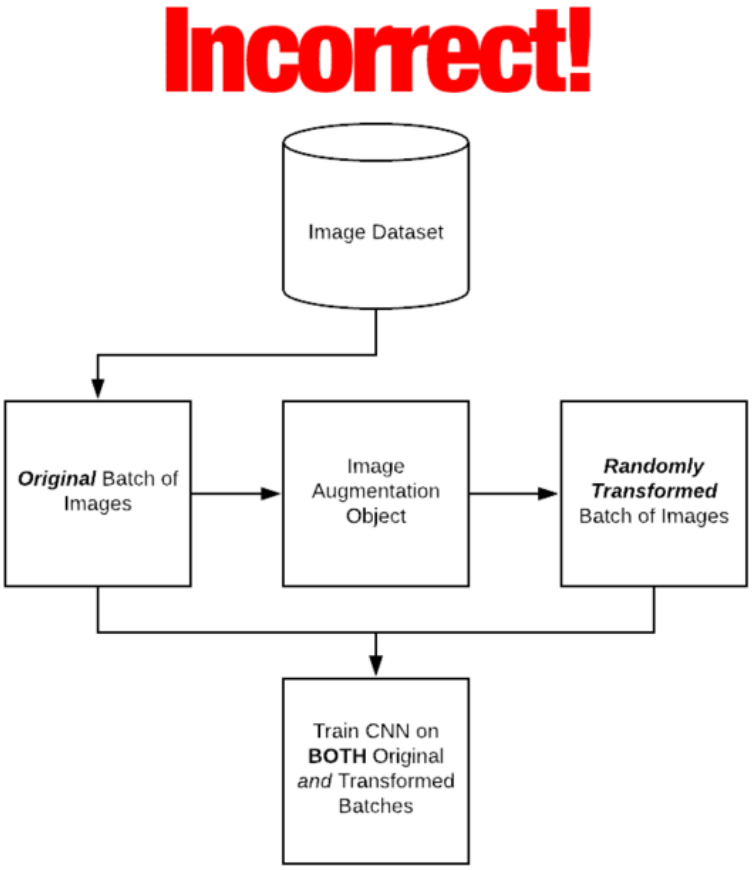

Рисунок 2.3 – Як не працює розширення даних Keras

На ілюстрації вище ImageDataGenerator приймає вхідний пакет зображень, випадковим чином перетворює пакет, а потім повертає як оригінальний пакет, так і змінені дані — знову ж таки, це не те, що Keras ImageDataGenerator робить. Натомість, ImageDataGenerator клас поверне лише випадково перетворені дані.

Треба мати на увазі, що вся суть техніки збільшення даних, описаної в цьому розділі, полягає в тому, щоб забезпечити, щоб мережа бачить «нові» зображення, які вона ніколи не «бачила» раніше в кожну епоху .

Якби було включено вихідні навчальні дані разом із доповненими даними до кожного пакету, то мережа «бачила» б оригінальні навчальні дані кілька разів, фактично порушивши ціль. По-друге, загальна мета збільшення даних полягає в тому, щоб підвищити узагальнюваність моделі. Для досягнення цієї мети «заміняємо» навчальні дані випадково перетвореними, доповненими даними.

На практиці це призводить до моделі, яка працює краще на даних перевірки/тестування, але, можливо, дещо гірше працює з навчальними даними (через зміни в даних, спричинені випадковими перетвореннями).

2.2.3 Поєднання генерації наборів даних і розширення на місці

Остаточний тип розширення даних має на меті поєднання як генерування наборів даних, так і розширення на місці — є можливість побачити цей тип збільшення даних під час виконання поведінкового клонування . Чудовий приклад поведінкового клонування можна побачити в програмах для самокерованих автомобілів. Створення наборів даних для самокерованих автомобілів може бути надзвичайно трудомістким і дорогим — вихід із проблеми полягає в тому, щоб замість цього використовувати відеоігри та симулятори водіння автомобіля. Графіка відеоігор стала

настільки реалістичною, що саме використано у наступному прикладі. Behavioral Cloning — зробіть так, щоб автомобіль їздив як ви.

Ця стаття містить думки для Udacity Self-Driving Car Nanodegree. Код та технічні деталі можна знайти [5] .

У цьому проекті для Udacity Self-Driving Car Nanodegree розроблено глибокий CNN, який може керувати автомобілем у симуляторі, наданому Udacity. CNN веде автомобіль автономно по колії. Мережа тренується на зображеннях із відеопотоку, який був записаний під час керування автомобілем. Таким чином, CNN клонує поведінку людини за кермом.

Цілі/кроки цього проекту такі:

– використовуйте симулятор для збору даних про хорошу поведінку водія;

– build, нейронна мережа згортки в Keras, яка прогнозує кути повороту за зображеннями;

– навчайте та перевіряйте модель за допомогою набору для навчання та перевірки;

– перевірте, чи модель успішно їде по першій колії, не з’їжджаючи з дороги;

– підсумуйте результати у письмовому звіті.

Проект включає в себе такі файли:

– model.py, що містить скрипт для створення та навчання моделі;

– drive.py для керування автомобілем в автономному режимі;

– model.h5, що містить навчену нейронну мережу згортки;

– це README.md, ця стаття та image_transformation_pipeline.ipynb для пояснення.

Додатково потрібно завантажити та розпакувати симулятор самокерованого автомобіля Udacity (використовувалася версія 1). Щоб запустити код, треба запустити симулятор у autonomous mode, відкрити іншу оболонку та ввести python drive.py model.h5

Щоб навчити модель, спочатку треба створити довідник ./data/mydata, об’їжджайте автомобіль training modeпо колії та зберегти дані в цьому каталозі. Потім модель навчається шляхом введення тексту python model.py

Решта цього README.mdмістить детальну інформацію про використану модель.

Імітований автомобіль оснащений трьома камерами, одна зліва, одна в центрі та одна справа від водія, які забезпечують зображення з цих різних точок огляду. Тренувальна доріжка має гострі повороти, з’їзди, в’їзди, мости, частково відсутні смуги та змінні умови освітлення. Існує додаткова тестова траса зі зміною висот, ще більш різкими поворотами та нерівностями. Таким чином, дуже важливо, щоб CNN не просто запам’ятовував першу доріжку, а й узагальнив невидимі дані, щоб добре працювати на тестовій доріжці. Розроблена тут модель була підготовлена виключно на тренувальній трасі і завершує тестовий трек. Основна проблема полягає в перекосі та упередженості набору даних. Нижче показана гістограма кутів повороту рульового колеса, записаних під час руху посередині дороги протягом кількох кіл (рис. 2.4). Це також дані, які використовуються для навчання. Перекіс вліво-вправо є менш проблематичним і його можна усунути, перевертаючи зображення та кути повороту одночасно. Однак навіть після балансування лівого та правого кутів більшу частину часу кут повороту під час нормальної їзди невеликий або дорівнює нулю, і, таким чином, створюється ухил у напрямку руху прямо. Але найважливіші події – це ті, коли машині потрібно різко повернути.

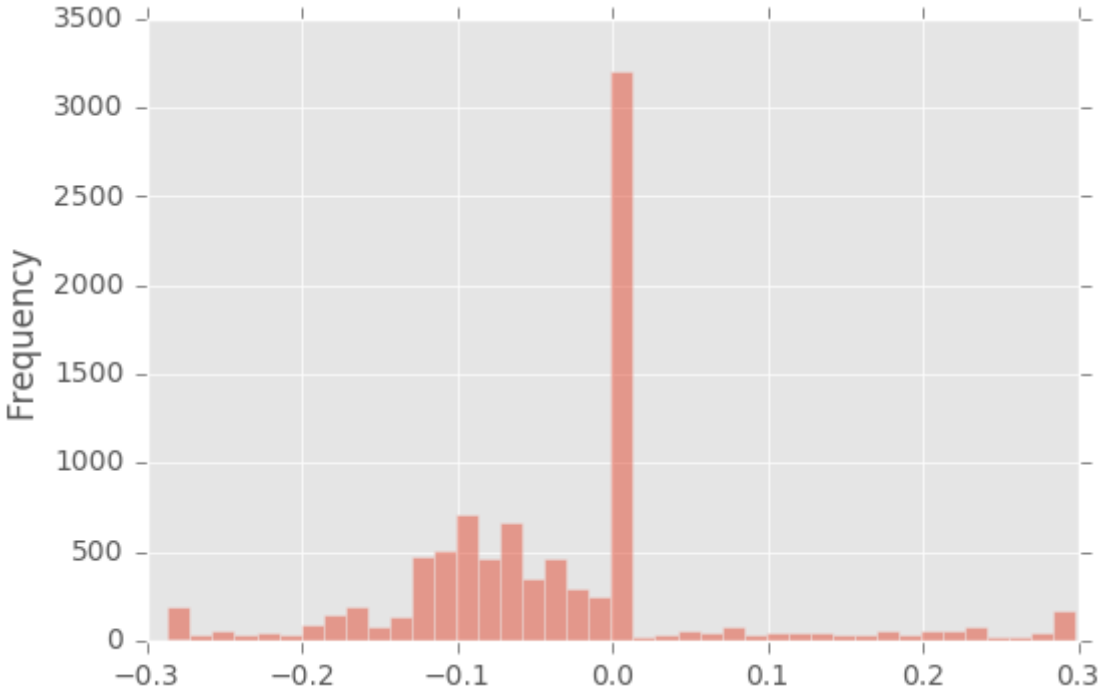

Рисунок 2.4 – Гістограма кутів повороту рульового колеса

Не враховуючи цього ухилу до нуля, автомобіль швидко залишає трасу. Один із способів протистояти цій проблемі — це навмисне дозволити автомобілю занестися на узбіччя і почати відновлення в самий останній момент. Однак правильні великі кути повороту нелегко створити таким чином, тому що навіть тоді більшу частину часу автомобіль їде прямо, за винятком короткого моменту, коли водій уникає аварії або автомобіль виїжджає з дороги.

2.2.3.2 Архітектура моделі

Архітектура CNN була успішно використана для прогнозування кута повороту симулятора. Серед них архітектура CNN від NVIDIA або архітектура comma.ai , які були успішно використані, наприклад, це

представлення . У цій статті Вівек Ядав запропонував рішення цієї проблеми керування на основі розумного використання розширення даних. Ця заява спирається на отримані там ідеї, але відрізняється архітектурою мережі та методами розширення.

У всіх вищезгаданих архітектурах одна змінна - поточний кут повороту - прогнозується як число з реальним значенням. Таким чином, проблема не є класифікацією, а завданням регресії. Ми побудуємо подібну архітектуру, яка передбачає одне реальне значення, але було б цікаво подивитися, як працює дискретизована версія.

Для архітектури мережі використовується CNN, яка розвинулась на основі попереднього представлення для класифікації дорожніх знаків з високою (97,8%) точністю. Проте ми внесли деякі важливі зміни.

Мережа починається з шару попередньої обробки, який приймає зображення форми 64x64x3. Кожне зображення нормалізується до діапазону [-1,1], інакше попередня обробка не виконується. Після вхідного шару йдуть 4 згорткових шари. Активації ReLU використовуються по всій мережі. Перші два згорткові шари використовують ядра розміру k=(8,8) із кроком s=(4,4) та 32 та 64 канали відповідно. Наступний згортковий шар використовує k=(4,4) ядра, крок s=(2,2) і 128 каналів. В останньому згортковому шарі використовується k=(2,2), крок s=(1,1) і знову 128 каналів. Після згорткових шарів йдуть два повністю пов’язані шари з активацією ReLU, а також регуляризацією випадання безпосередньо перед шарами. Останній шар — це один нейрон, який забезпечує прогнозований кут повороту.

Таблиця 2.1 – Результати порівнянь

Шар (тип)	**Вихідна форма**	**Параметр №**	**Пов'язаний з**
lambda_1 (лямбда)	(Немає, 64, 64, 3)	0	лямбда_вхід_1[0][0]
convolution2d_1 (Convolution2D)	(Немає, 16, 16, 32)	6176	лямбда_1[0][0]

Продовження таблиці 2.2

Шар (тип)	**Вихідна форма**	**Параметр №**	**Пов'язаний з**
activation_1 (Активація)	(Немає, 16, 16, 32)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(Немає, 4, 4, 64)	131136	активація_1[0][0]
relu2 (Активація)	(Немає, 4, 4, 64)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(Немає, 2, 2, 128)	131200	relu2[0][0]
activation_2 (Активація)	(Немає, 2, 2, 128)	0	convolution2d_3[0][0]
convolution2d_4 (Convolution2D)	(Немає, 2, 2, 128)	65664	активація_2[0][0]
activation_3 (Активація)	(Немає, 2, 2, 128)	0	convolution2d_4[0][0]
flatten_1 (зрівняти)	(Немає, 512)	0	активація_3[0][0]
dropout_1 (випуск)	(Немає, 512)	0	flatten_1[0][0]
щільний_1 (Щельний)	(Немає, 128)	65664	випадання_1[0][0]
activation_4 (Активація)	(Немає, 128)	0	щільний_1[0][0]
dropout_2 (випуск)	(Немає, 128)	0	активація_4[0][0]
густий_2 (щільний)	(Немає, 128)	16512	випадання_2[0][0]
густий_3 (щільний)	(Немає, 1)	129	щільний_2[0][0]

Усі обчислення проводилися на системі Ubuntu 16.04 з процесором Intel i7 та NVIDIA GTX 1080.

Через проблеми з генерацією важливих подій відновлення вручну була використана стратегія розширення. Необроблені дані про тренування були зібрані, керуючи автомобілем якомога плавніше прямо посередині дороги протягом 3-4 кіл в одному напрямку. Були змодельовані події відновлення шляхом перетворення (зміщення, зрізи, обрізки, яскравість, перевороти) записаних зображень за допомогою бібліотечних функцій з OpenCV з відповідними змінами кута повороту. Остаточні навчальні зображення потім генеруються партіями по 200 на льоту з 20000 зображень на епоху. Генератор

Python створює нові навчальні пакети, застосовуючи вищезгадані перетворення з відповідно виправленими кутами повороту.

Проведені операції є:

– вибрано випадковий навчальний приклад;

– камера (ліва, права, центральна) вибирається випадковим чином;

– випадковий зсув: зображення зрізається по горизонталі, щоб імітувати вигин дороги;

– випадкове обрізання: ми випадковим чином обрізаємо кадр із зображення, щоб імітувати зміщення автомобіля від середини дороги (на цьому кроці виконується також зниження дискретизації зображення до 64x64x3);

– випадковий переворот: щоб переконатися, що повороти ліворуч і праворуч відбуваються однаково часто;

– випадкова яскравість: для імітації різних умов освітлення.

На кроках 1-4 кут повороту регулюється з урахуванням зміни зображення. Об’єднання цих операцій веде до практично нескінченної кількості різноманітних навчальних прикладів. Зміни кута повороту, що відповідають кожній із зазначених вище операцій, визначали вручну шляхом дослідження результату кожного перетворення та використання тригонометрії. Наприклад, корекція кута, що відповідає горизонтальній трансформації зсуву, повинна бути пропорційна куту зсуву.

2.1.3.3 Епохи та підтвердження

З метою перевірки 10% навчальних даних (близько 1000 зображень) було затримано. Для перевірки використовуються лише зображення центральної камери. Через кілька епох (~ 10) втрата підтвердження та навчання завершується. Втрата перевірки стабільно становить приблизно

половину втрат при навчанні, що вказує на недостатню підготовку, однак із застереженням, що дані навчання та перевірки не беруться з однієї вибірки: немає розширення даних для даних перевірки. Більш надійна, хоча й не автоматична метрика полягає в перевірці продуктивності мережі, дозволяючи їй керувати автомобілем на другій колії, яка не використовувалася під час навчання.

У ході навчання машина обійшла тренувальний трек майже відразу після введення генератора. Однак він був недостатньо реактивним, щоб завершити тестовий трек. Налаштування параметрів корекції кута в розділі розширення коду і перенавчання мережі протягом приблизно 10 епох усунули проблему, і автомобіль освоїв тестовий трек.

Треба зауважити, що автомобіль успішно відновлюється з критичних ситуацій, хоча подій відновлення не зафіксовано. Усі події відновлення були згенеровані синтетично шляхом спотворення зображень автомобіля, що рухається посеред дороги, як описано вище.

Висновки: використовуючи послідовне збільшення зображення з відповідними оновленнями кута повороту вдалося навчити нейронну мережу відновлювати автомобіль після екстремальних подій, як-от раптово з’являються криві зміни умов освітлення, виключно моделюючи такі події на основі звичайних даних про водіння. Мета — навчити згорткову нейронну мережу (CNN) керувати автомобілем у симуляторі, наданому Udacity. Автомобіль оснащений трьома камерами, які забезпечують відеопотоки та записують значення кута повороту, швидкості, газу та гальма. Кут повороту – єдине, що потрібно передбачити, але більш просунуті моделі також можуть захотіти передбачити газ і гальмо. Виявляється, це завдання регресії, яке дуже відрізняється від звичайних застосувань CNN для цілей класифікації.

Дані про тренування, були зібрані завдяки керуваню автомобілем тренувальним треком на рівній місцевості. Ефективність CNN можна перевірити, дозволивши автомобілю їздити автономно по тій же колії або в

ідеалі по другій колії, яка значно вітряніша з крутими пагорбами і яку не слід використовувати для тренувань. Нижче наведено кілька фотографій з різних камер на автомобілі на тренувальній трасі.

Рисунок 2.5 – Фотографія з камери на автомобілі

Рисунок 2.6 – Фотографія з камери на автомобілі

Під час руху автомобіля в нормальних умовах кут повороту керма в більшості випадків дуже близький до нуля. Це добре видно на необроблених даних про навчання. Нижче наведено кути повороту, які були записані під час руху автомобіля по трасі, залишаючись якомога ближче до середини смуги.

Це всі дані, які я зібрав для підготовки остаточної моделі (~9000 зображень або проїзд 3–4 кола).

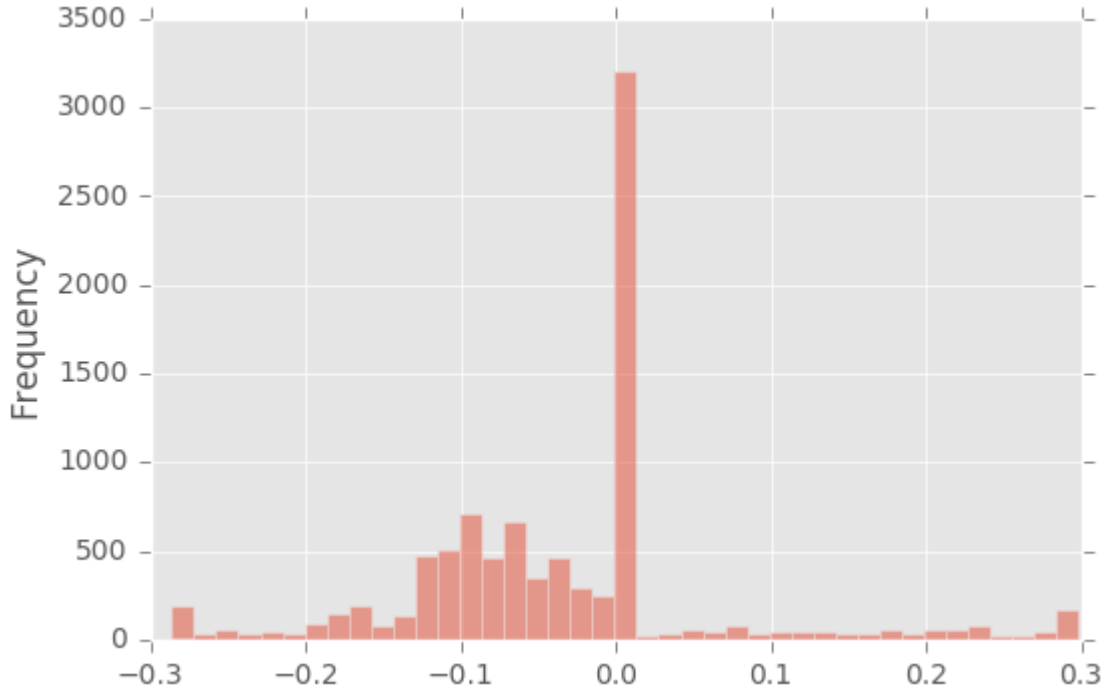

Рисунок 2.7 – Гістограма кутів повороту рульового колеса

Гістограма кутів повороту: діапазон [-1,1] відповідає діапазону кутів ±25°

Перекіс вліво/вправо пов’язаний з рухом автомобіля по колії лише в одному напрямку, і його можна усунути, перегортаючи кожне записане зображення та відповідний йому кут повороту. Більш клопітним є упередження щодо їзди прямо: рідкісні випадки, коли реєструється великий кут повороту, також є найважливішими, якщо автомобіль має залишитися на дорозі. Одним з можливих рішень було б дозволити автомобілю занестися на край дороги і відновитися до того, як станеться аварія. Такий варіант був протестований, але це рішення було незадовільним, тому що в такому

випадку автомобіль все ще їде прямо більшу частину часу — напрямок «з траси» — з кількома великими кутами повороту, посипаними зверху. В результаті CNN, навчений на таких даних, як правило, навіть не завершує трену навчання, якщо тільки дані навчання не є «правильними». Була отримана CNN, щоб їздити на машині по тренувальній трасі таким чином.

Іншим рішенням було б проводити вибірку подій з екстремальними кутами частіше, ніж з малими кутами. Однак, оскільки вони рідкісні, може знадобитися зібрати велику кількість навчальних даних для моделі, щоб уникнути переобладнання.

Для тренувань машина була керована максимально плавно прямо посеред дороги. Причиною цього була ідея постійно записувати ідеальний кут повороту. Потім події відновлення були змодельовані шляхом спотворення та обрізання записаних зображень камери та регулювання кута повороту. Таким чином, об’єднуючи спотворення зображень, випадкові корекції яскравості та обрізання, можна було б створити практично нескінченну кількість навчальних зображень із невеликих навчальних даних, які я зібрав. Це спрацювало на диво добре.

Були використані зображення з усіх трьох камер. Зображення, зроблені з бічних камер, схожі на паралельні переклади автомобіля. Щоб врахувати те, що я був поза центром, був налаштований кут повороту для зображень, зроблених з бічних камер, таким чином: ігноруючи спотворення перспективи, можна вважати, що якщо бічні камери розташовані приблизно на 1,2 метра від центру, і автомобіль повинен повернутися до середині дороги в межах наступних 20 метрів поправка на рульове керування має становити приблизно 1,2/20 радіан (з використанням $\tan(\alpha)\sim\alpha$). Це виявилося досить потужним засобом, щоб змусити автомобіль уникати узбіч дороги.

Рисунок 2.8 – Випадковий вид камери

На рисунку 2.8 кут повороту складає: -2,8°, виправлений: -3,4° .На наступному етапі кожне зображення зрізали по горизонталі. Пікселі в нижній частині зображення були фіксовані, а верхній ряд був випадковим чином переміщений ліворуч або праворуч. Кут повороту був змінений пропорційно куту зсуву. Це призвело до того, що криві доріжки з’являлися в тренувальному наборі так само часто, як і прямі.

Рисунок 2.9 – Випадковий зсув: виправлений кут повороту: -1,0°

Поки що всі трансформації були виконані на повних зображеннях 320x160 пікселів, які надходять від камер. На наступному етапі було вибрано

вікно розміром 280x76 пікселів, яке усунуло капот із нижньої частини зображення та обрізало частину над горизонтом на рівній місцевості у верхній частині. Для кожного зображення розташування цього вікна було випадковим чином зміщено від центру вздовж осей x і y на ±20 і ±10 пікселів відповідно. Кут повороту був змінений пропорційно бічному зміщенню врожаю. Таким чином, також були змодельовані зображення автомобіля, що перебуває на горбистій місцевості, і можна було отримати більшу різноманітність зображень, ніж доступна з бічних камер. Потім розмір результату був змінений до 64x64 пікселів.

Рисунок 2.10 – Випадковий вид, виправлений кут повороту: -2,1°

Нарешті, кожне зображення було випадковим чином перевернуто (по горизонталі) з однаковою ймовірністю, щоб повороти ліворуч і праворуч з’являлися однаково часто. Також яскравість регулювалася випадковим чином.

Рисунок 2.11 – Випадкова яскравість і поворот: виправлений кут повороту: 2,1°

Об’єднання всіх цих перетворень можна ефективно виконувати пакетами, які генеруються на льоту під час навчання із записаних зображень.

Архітектура моделі та навчання CNN, яка була обрана, є досить стандартною CNN, що складається з 4 згорткових шарів з активаціями ReLU, за якими слідують два повністю пов’язані шари з регуляризацією випадання. Нарешті один нейрон сформував вихід, який передбачав кут повороту. Варто відзначити відсутність шарів об’єднання. Причина уникнення об’єднання шарів полягала в тому, що рівні об’єднання роблять вихід CNN певною мірою інваріантним до зсувів вхідних даних, що бажано для завдань класифікації, але контрпродуктивно для утримання автомобіля посеред дороги. Мережа в системі Ubuntu 16.04 була навчена завдяки, використовувані графічного процесора NVIDIA GTX 1080. Для будь-якого заданого набору гіпер параметрів втрати зазвичай перестали зменшуватися через кілька епох (200 000 зображень кожна).

Результати стали приємним сюрпризом. Автомобіль пройшов не тільки тренувальний, але навіть тестовий трек. Нижче наведено результати для одного з найперших втілень цієї моделі, до налаштування гіперпараметрів.

Отримавши дані про тренування, є можливість повернутися назад і застосувати розширення даних типу №2 (тобто збільшення даних на місці/на льоту) до даних, які були зібрані за допомогою симуляції.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Налаштування середовища розробки

Щоб налаштувати систему для цієї роботи, були використані ці посібники [12,13].

Будь-який підручник допоможе налаштувати систему з усім необхідним програмним забезпеченням для цієї роботи в зручному віртуальному середовищі Python.

3.2 Структура проекту

Перш ніж зануритися в код, треба спочатку розглянути структуру каталогів для проекту. Проект розроблений у Jupyter Notebook, він складається з семи каталогів та дев’яти файлів Структуру проекту представлена на лістингу 3.1

Лістинг 3.1 – Структуру каталогів для проекту

```
$ tree --dirsfirst --filelimit 10
.
├── dogs_vs_cats_small
│   ├── cats [1000 entries]
│   └── dogs [1000 entries]
├── generated_dataset
│   ├── cats [100 entries]
│   └── dogs [100 entries]
├── pyimagesearch
│   ├── __init__.py
│   └── resnet.py
├── cat.jpg
├── dog.jpg
├── plot_dogs_vs_cats_no_aug.png
├── plot_dogs_vs_cats_with_aug.png
├── plot_generated_dataset.png
├── train.py
└── generate_images.py

7 directories, 9 files
```

По-перше, є два каталоги наборів даних , які не слід плутати:

– *dogs_vs_cats_small/* : підмножина популярного набору даних змагань Kaggle Dogs vs. Cats . У моїй підмножині присутні лише 2000 зображень (1000 на клас) (на відміну від 25 000 зображень для завдання);

– *generated_dataset/* : було створено цей згенерований набір даних за допомогою *cat.jpg* і *dog.jpg* зображення, які знаходяться в батьківському каталозі. Був використаний тип збільшення даних №1 для автоматичного створення цього набору даних і заповнення цього каталогу зображеннями.

Далі є *pyimagesearch* модуль, який містить реалізацію класифікатора ResNet CNN.

Було розглянуто два скрипти Python:

– *train.py*: використовується для навчання моделей як для типу №1, так і для типу №2 (і за бажанням типу №3 , якщо цього бажає користувач)

методів збільшення даних. Було проведено три навчальні експерименти, у результаті яких був кожен з трьох *сюжет*.png* файли в папці проекту;

– *generate_images.py*: використовується для створення набору даних з одного зображення за допомогою типу №1 .

3.3 Реалізація сценарію навчання

У решті цієї роботи було проведено три експерименти:

– створення набору даних за допомогою розширення набору даних і навчання на ньому CNN;

– використання підмножини набору даних Kaggle Dogs vs. Cats і тренування CNN без розширення даних;

– повторення другого експерименту, але цього разу зі збільшенням даних.

– усі ці експерименти будуть виконані за допомогою одного сценарію Python.

Для початку треба відкрити *train.py.*

Лістинг 3.2 – Імпорт необхідних пакетів

```
# set the matplotlib backend so figures can be saved in the background
import matplotlib
matplotlib.use("Agg")

# import the necessary packages
from pyimagesearch.resnet import ResNet
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import cv2
import os
```

По рядках 2-18 імпортуються необхідні пакети. У Рядку 10 ImageDataGenerator робить імпорт із бібліотеки Keras — класу для розширення даних.

Розберем аргументи командного рядка :

Лістинг 3.3 – Аргументи командного рядка

```
20. # construct the argument parser and parse the arguments
21. ap = argparse.ArgumentParser()
22. ap.add_argument("-d", "--dataset", required=True,
23.    help="path to input dataset")
24. ap.add_argument("-a", "--augment", type=int, default=-1,
25.    help="whether or not 'on the fly' data augmentation should be used")
26. ap.add_argument("-p", "--plot", type=str, default="plot.png",
27.    help="path to output loss/accuracy plot")
28. args = vars(ap.parse_args())
```

Скрипт приймає три аргументи командного рядка через термінал:

− набір даних: шлях до набору вхідних даних;

− збільшення: чи слід використовувати розширення даних «на льоту» за замовчуванням цей метод не виконується;

− сюжет: шлях до вихідного графіка історії навчання.

Перейдемо до ініціалізації гіперпараметрів і завантаження даних зображення:

Лістинг 3.4 – Ініціалізації гіперпараметрів і завантаження даних зображення.

```
30. # initialize the initial learning rate, batch size, and number of
31. # epochs to train for
32. INIT_LR = 1e-1
33. BS = 8
34. EPOCHS = 50
35.
36. # grab the list of images in our dataset directory, then initialize
37. # the list of data (i.e., images) and class images
38. print("[INFO] loading images...")
39. imagePaths = list(paths.list_images(args["dataset"]))
40. data = []
41. labels = []
42.
43. # loop over the image paths
44. for imagePath in imagePaths:
45.     # extract the class label from the filename, load the image, and
46.     # resize it to be a fixed 64x64 pixels, ignoring aspect ratio
47.     label = imagePath.split(os.path.sep)[-2]
48.     image = cv2.imread(imagePath)
49.     image = cv2.resize(image, (64, 64))
50.
51.     # update the data and labels lists, respectively
52.     data.append(image)
53.     labels.append(label)
```

Гіперпараметри навчання, включаючи початкову швидкість навчання, розмір пакету та кількість епох для навчання, ініціалізуються в рядках 32-34 . Звідки було взято рядки 39-53 шляхи зображення, завантаженні зображення та заповнені дані та списки. Єдина попередня обробка зображення, яка виконана на цьому етапі, — це змінювання розміру кожного зображення до 64×64 пікселів.

Далі було завершено попередню обробку, та закодовано мітки а також розділив дані (лістинг 3.5).

Лістинг 3.5 – Кодування міток та розділ даних

```
55. # convert the data into a NumPy array, then preprocess it by scaling
56. # all pixel intensities to the range [0, 1]
57. data = np.array(data, dtype="float") / 255.0
58.
59. # encode the labels (which are currently strings) as integers and then
60. # one-hot encode them
61. le = LabelEncoder()
62. labels = le.fit_transform(labels)
63. labels = to_categorical(labels, 2)
64.
65. # partition the data into training and testing splits using 75% of
66. # the data for training and the remaining 25% for testing
67. (trainX, testX, trainY, testY) = train_test_split(data, labels,
68.     test_size=0.25, random_state=42)
```

У рядку 57 дані перетворені в масив NumPy, а також масштабовані всі інтенсивності пікселів до діапазону [0, 1] . На цьому попередня обробка завершена. Звідти виконується «один гаряче кодування» етикетки (рядки 61-63). Цей спосіб кодування етикетки призводить до масиву, який може виглядати так:

Лістинг 3.6 – Отриманий масив

```
1. array([[0., 1.],
2.        [0., 1.],
3.        [0., 1.],
4.        [1., 0.],
5.        [1., 0.],
6.        [0., 1.],
7.        [0., 1.]], dtype=float32)
```

Для цієї вибірки даних є дві кішки ([1. , 0.]) і п'ять собак ([0. , 1]), де етикетка, що відповідає зображенню, позначена як «гаряча». Звідти проходить розподіл даних на навчання та тестування поділяється, позначаючи 75% даних для навчання та решту 25% для тестування (рядки 67 і 68).

Потім проходить ініціалізація об'єкту для розширення даних (ліст. 3.7).

Лістинг 3.7 – Ініціалізування об’єкта розширення даних

```
70. # initialize an our data augmenter as an "empty" image data generator
71. aug = ImageDataGenerator()
```

Рядок 71 ініціалізує порожній об’єкт розширення даних (тобто жодне збільшення не виконуватиметься). Це стандартна операція цього сценарію.

Після цього перевіряємо, чи змінюється значення за замовчуванням за допомогою --augment аргументу командного рядка.

Лістинг 3.8 – Перевірка на збільшення даних

```
73. | # check to see if we are applying "on the fly" data augmentation, and
74. | # if so, re-instantiate the object
75. | if args["augment"] > 0:
76. |     print("[INFO] performing 'on the fly' data augmentation")
77. |     aug = ImageDataGenerator(
78. |         rotation_range=20,
79. |         zoom_range=0.15,
80. |         width_shift_range=0.2,
81. |         height_shift_range=0.2,
82. |         shear_range=0.15,
83. |         horizontal_flip=True,
84. |         fill_mode="nearest")
```

Рядок 75 перевіряє, чи виконується збільшення даних. Якщо так, то проходить повторна ініціалізація об’єкту розширення даних із випадковими параметрами перетворення (рядки 77-84). Як вказують параметри, випадкові повороти, масштабування, зсуви, зрізи та перевороти будуть виконуватися під час збільшення даних на місці/на льоту.

Збираємо та навчаємо нашу модель, як показано на лістингу 3.9

Лістинг 3.9 – Навчання моделі

```
86. | # initialize the optimizer and model
87. | print("[INFO] compiling model...")
88. | opt = SGD(lr=INIT_LR, momentum=0.9, decay=INIT_LR / EPOCHS)
89. | model = ResNet.build(64, 64, 3, 2, (2, 3, 4),
90. |     (32, 64, 128, 256), reg=0.0001)
91. | model.compile(loss="binary_crossentropy", optimizer=opt,
92. |     metrics=["accuracy"])
93. |
94. | # train the network
95. | print("[INFO] training network for {} epochs...".format(EPOCHS))
96. | H = model.fit(
97. |     x=aug.flow(trainX, trainY, batch_size=BS),
98. |     validation_data=(testX, testY),
99. |     steps_per_epoch=len(trainX) // BS,
100. |    epochs=EPOCHS)
```

Рядки 88-92 будують *ResNet* модель з використанням оптимізації стохастичного градієнтного спуску та зниження швидкості навчання. Використовуються "binary_crossentropy" втрати для цієї 2-класної задачі. Якщо було би більше двох класів, треба було обов'язково користуватися "categorial_crossentropy".

Рядки 96-100 потім навчають модель. *Aug* об'єкт обробляє пакетне збільшення даних (хоча не треба забувати, що файл *aug* об'єкт виконуватиме збільшення даних, лише якщо *--augment* було встановлено аргументом командного рядка).

Модель оцінена та настає час надрукувати статистичні дані та згенерувати графік історії навчання. Код генерації графіка можна побачити на лістингу 3.10

Лістинг 3.10 – Код генерування графіка навчання

```
102. # evaluate the network
103. print("[INFO] evaluating network...")
104. predictions = model.predict(x=testX.astype("float32"), batch_size=BS)
105. print(classification_report(testY.argmax(axis=1),
106.    predictions.argmax(axis=1), target_names=le.classes_))
107.
108. # plot the training loss and accuracy
109. N = np.arange(0, EPOCHS)
110. plt.style.use("ggplot")
111. plt.figure()
112. plt.plot(N, H.history["loss"], label="train_loss")
113. plt.plot(N, H.history["val_loss"], label="val_loss")
114. plt.plot(N, H.history["accuracy"], label="train_acc")
115. plt.plot(N, H.history["val_accuracy"], label="val_acc")
116. plt.title("Training Loss and Accuracy on Dataset")
117. plt.xlabel("Epoch #")
118. plt.ylabel("Loss/Accuracy")
119. plt.legend(loc="lower left")
120. plt.savefig(args["plot"])
```

Рядок 104 робить прогнози для тестового набору з метою оцінки. Звіт про класифікацію друкується через рядки 105 і 106 . Звідти рядки 109-120 генерують і зберігають графік навчання точності/втрат.

3.4 Створення набору даних із розширенням даних і Keras

У моєму першому експерименті виконано розширення набору даних за допомогою розширення даних за допомогою Keras.

Набір даних містив 2 класи, і спочатку набір даних тривіально містив лише одне зображення на клас кішка та одне зображення на клас собака. Було використано розширення даних типу №1, щоб створити новий набір даних із 100 зображеннями на клас кішка та ще зі 100 зображеннями на клас собака.

Рисунок 3.1 допоможе зрозуміти, як проходить створення прикладу набору даних за допомогою Keras, який виконує випадкові маніпуляції з зображенням.

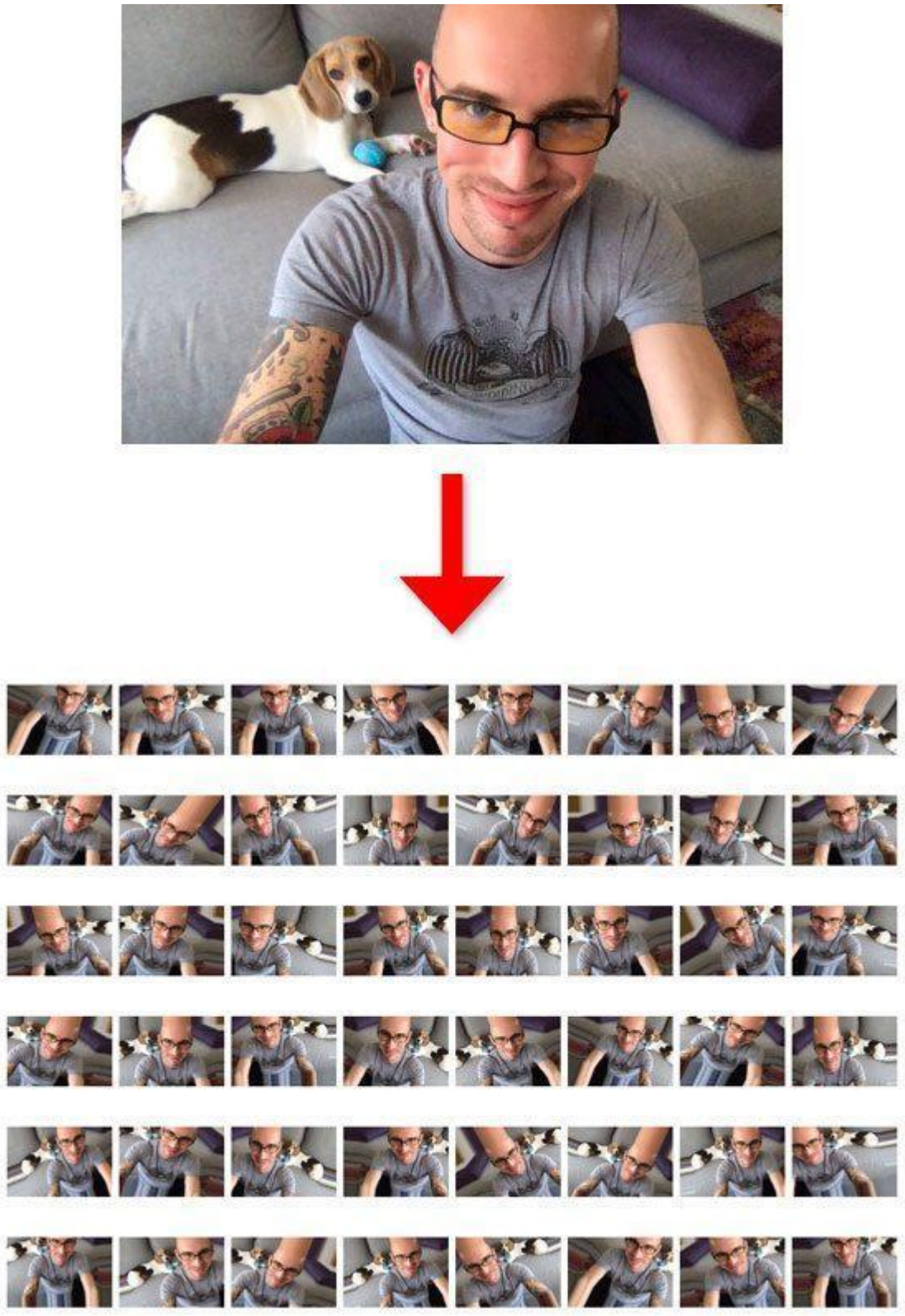

Рисунок 3.1 – Збільшення даних за допомогою Keras виконує випадкові маніпуляції із зображеннями.

Перш ніж тренувати нашу CNN, спочатку було потрібно створити приклад набору даних. З розділу 3.2 «Структура проекту» вище відомо, що два приклади зображень у кореневому каталозі: cat.jpg і dog.jpg. Були використані ці приклади зображень, щоб створити 100 нових навчальних зображень на клас (усього 200 зображень).

Щоб побачити, як можливо використовувати розширення даних для створення нових прикладів, треба відкрийти файл *generate_images.py*

Лістинг 3.11 – Код використуваня розширення даних для створення нових прикладів

```
1.  # import the necessary packages
2.  from tensorflow.keras.preprocessing.image import ImageDataGenerator
3.  from tensorflow.keras.preprocessing.image import img_to_array
4.  from tensorflow.keras.preprocessing.image import load_img
5.  import numpy as np
6.  import argparse
7.
8.  # construct the argument parser and parse the arguments
9.  ap = argparse.ArgumentParser()
10. ap.add_argument("-i", "--image", required=True,
11.    help="path to the input image")
12. ap.add_argument("-o", "--output", required=True,
13.    help="path to output directory to store augmentation examples")
14. ap.add_argument("-t", "--total", type=int, default=100,
15.    help="# of training samples to generate")
16. args = vars(ap.parse_args())
```

Рядки 2-6 імпортують необхідні пакети. Наші ImageDataGenerator імпортується в рядок 2 і буде обробляти збільшення даних за допомогою Keras. Звідти було розібрано три аргументи командного рядка:

– --image: шлях до вхідного зображення. Ми створимо додаткові випадкові мутовані версії цього зображення;

– --output: шлях до вихідного каталогу для зберігання прикладів збільшення даних;

– --total: кількість зразків зображень для створення.

Потім необхідно завантажити необхідне зображення і ініціалізувати об'єкт збільшення даних, код завантаження та ініціалізації можна побачити на лістингу 3.12

Лістинг 3.12 – Код завантажження та ініціалізації об'єкта збільшення даних

```
18. # load the input image, convert it to a NumPy array, and then
19. # reshape it to have an extra dimension
20. print("[INFO] loading example image...")
21. image = load_img(args["image"])
22. image = img_to_array(image)
23. image = np.expand_dims(image, axis=0)
24.
25. # construct the image generator for data augmentation then
26. # initialize the total number of images generated thus far
27. aug = ImageDataGenerator(
28.     rotation_range=30,
29.     zoom_range=0.15,
30.     width_shift_range=0.2,
31.     height_shift_range=0.2,
32.     shear_range=0.15,
33.     horizontal_flip=True,
34.     fill_mode="nearest")
35. total = 0
```

Зображення завантажується та готується до розширення даних через рядки 21-23 . Завантаження та обробка зображень здійснюється за допомогою функціональних можливостей Keras (тобто технології OpenCV не використовуються). Звідти проходить ініціалізація файл ImageDataGenerator об'єкт. Цей об’єкт полегшить виконання випадкових обертань, масштабування, зсувів, зсувів та переворотів вхідного зображення.

Далі проходить створення генератор Python і запустив його в роботу, поки не будуть створені всі потрібні зображення:

Лістинг 3.13 – Код створення генератору Python

```
37. # construct the actual Python generator
38. print("[INFO] generating images...")
39. imageGen = aug.flow(image, batch_size=1, save_to_dir=args["output"],
40.     save_prefix="image", save_format="jpg")
41.
42. # loop over examples from our image data augmentation generator
43. for image in imageGen:
44.     # increment our counter
45.     total += 1
46.
47.     # if we have reached the specified number of examples, break
48.     # from the loop
49.     if total == args["total"]:
50.         break
```

ImageGen було використано для випадкового перетворення вхідного зображення (рядки 39 і 40). Цей генератор зберігає зображення як файли .jpg у вказаний вихідний каталог, що міститься в ньому args["output"]. Нарешті, з’явилась змога переглянути приклади з генератора даних зображень і підрахувати їх, поки не буде досягнута необхідна кількості зображень.

Щоб запустити *generate_examples.py* скрипт треба переконайтеся, що використовується розділ «Завантаження» підручника, щоб завантажити вихідний код і приклади зображень.

Візуалізацію генерації набору даних за допомогою розширення даних можна побачити на рисунку 2.3 — треба звернути увагу, було прийнято одне вхідне зображення, а потім створено 100 нових прикладів навчання (48 з яких візуалізуються) з цього єдиного зображення.

3.5 Згорткові нейронні мережі

Згорткові нейронні мережі – клас штучних нейронних мереж, який став домінуючим в різних завданнях комп'ютерного зору та викликає інтерес в самих різних областях, включаючи медицину, а саме діагностування різних захворювань. ЗНМ були натхненні на біологічних процесах того, що модель з'єднання між нейронами нагадує організацію зорової кори тварини [14].

Окремі нейрони кори відповідають на стимули тільки в обмеженій області зорового поля, відомої як рецептивне поле. Далі чутливі поля різних нейронів частково перекриваються, так що вони покривають все поле зору. CNN є прикладом ШНМ, яка при своєму навчанні використовує метод навчання з учителем.

Ідея ЗНМ полягає в тому, що чергуються згоркові шари та шари субдискретизації. При цьому чергуванні частина нейронів деякого шару CNN можуть використовувати однакові ваги. Такі нейрони об'єднуються в карту ознак. Кожний нейрон такої карти пов'язан з частиною нейронів попереднього шару.

В порівнянні зі звичайними ШНМ в ЗНМ виділяють декілька видів шарів, кожен з яких має свої властивості:

– згортковий шар;

– шар субдискретизації;

– повноз'єднаний шар.

Розглянемо кожен з цих шарів більш детально.

Згортковий шар отримав свою назву від назви математичної операції – згортки. Згортка – це математична операція, яка застосовується до двох функції та в результаті якої отримується третя функція, яка в свою чергу може бути модифікацією однієї з первинних функцій. Наведемо математичне означення цієї операції:

Нехай f та g: $\mathbb{R}n \to \mathbb{R}$ дві функції, інтегровані відносно міри Лебега в просторі $\mathbb{R}n$. Тоді їхньою згорткою називається функція $f*g$: $\mathbb{R}n \to \mathbb{R}$, яка визначається формулою:

$$(f*g)(x) \stackrel{\text{def}}{=} \int f(y)g(x-y)dy = \int f(x-y)g(y)dy \; \mathbb{R}n \; \mathbb{R}n. \qquad (3.1)$$

Якщо розглядати згортку в аналізі зображень, то в даному випадку під згорткою мається на увазі операція обчислення нового значення обраного

пікселя зображення, враховуючи значення оточуючих пікселів. Для отримання результати згортки використовують матрицю, яка називається ядром згортки. Зазвичай ядро згортки – це матриця $n \times n$, де n– непарне число. Під час обчислення нового значення обраного пікселя ядро згортки «прикладається» своїм центром до даного пікселя, при цьому оточуючі пікселі теж накриваються ядром. Далі розраховується сума, доданками якої є добутки значень пікселів на значення комірки ядра, що накрила даний піксель. Потім отримана сума ділиться на суму всіх елементів ядра згортки. Після останньої дії ми отримуємо нове значенням обраного пікселя. Якщо застосувати згортку до кожного пікселя зображення, то в результаті вийде якийсь ефект, що залежить від обраного ядра згортки. На рисунку 3.2 наведено приклад операції згортки для зображення:

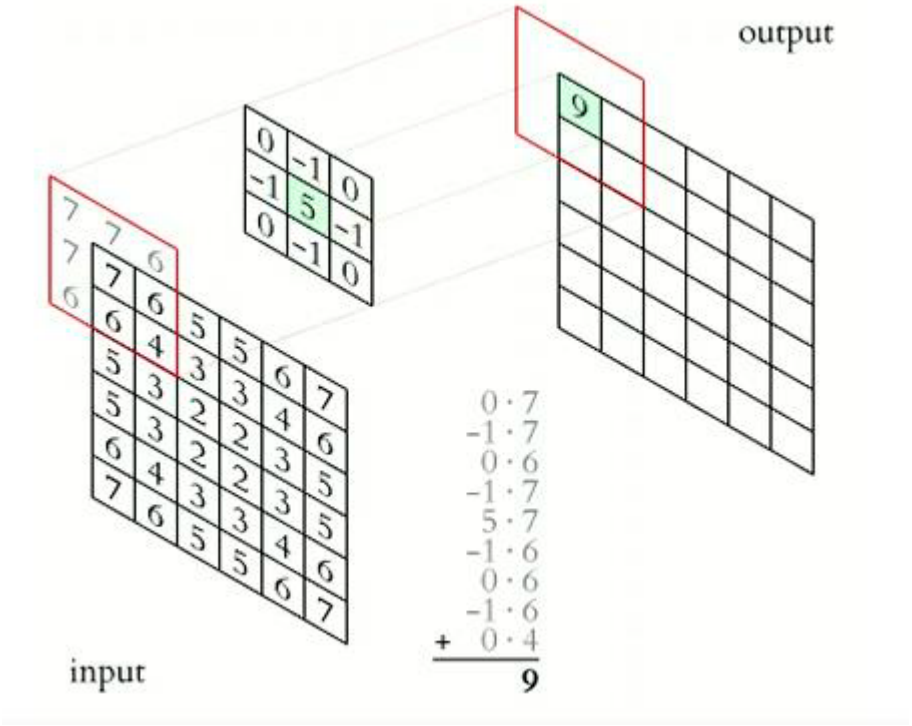

Рисунок 3.2 Приклад операції згортки для зображення

Шар згортки є фундаментальним компонентом архітектури CNN, який виконує витяг ознак, що зазвичай складається з комбінації лінійних і

нелінійних операцій, тобто операції згортки і функції активації. Процес навчання моделі ЗНМ щодо згорткового шару полягає в тому, щоб ідентифікувати ядра, які найкраще працюють для даного завдання, на основі заданого набору навчальних даних. Ядра - це єдині параметри, які автоматично вивчаються в процесі навчання в згортковому шарі, але розмір ядер, кількість ядер, відступи і крок - це гіперпараметри, які необхідно встановити до початку процесу навчання. Головне призначення одного згорткового шару – це виділення простих шаблонів у вході за допомогою фільтрів, які навчаються. Розташовуючи згорткові шари один за одним або комбінуючи з іншими типами шарів, отримуємо те, що зі зростанням глибини мережі буде зростати і абстрактність та складність ознак, що виділяються [15]. Наприклад, якщо фільтри налаштовані для того, щоб виділяти прямі лінії на зображенні, то після першого згорткового шару мережа розпізнає тільки прямі лінії, а вже після другого такого шару ще й комбінації цих ліній. Також зауважимо, що після кожної операції згортки CNN застосовує перетворення ReLU до карти ознак, привносячи в модель нелінійність.

Наступний шар ЗНМ, який ми розглянемо, – це шар субдискретизації або друга назва – шар пулінгу. Основною функцією такого шару є зниження розміру представлення даних після чергового шару для того, щоб зменшити кількість параметрів та обчислень в мережі, а також щоб уникнути перенавчання. Подібно згортковому шару, операція пулінгу пропускає фільтр по всьому входу, але різниця в тому, що цей фільтр немає ваг. Замість цього ядро застосовує функцію агрегування до значень в приймаючому полі, заповнюючи вихідний масив. Є два основних типи пулінгу: максимальний (MaxPooling) та середній (AveragePooling). MaxPooling – це коли фільтр переміщається по входу, він вибирає піксель з максимальним значенням для відправки у вихідний масив. До речі, цей підхід, як правило, використовується частіше, ніж середній пулінг. AveragePooling – в міру того, як фільтр переміщається по входу, він обчислює середнє значення в

приймаючому полі для відправки в вихідний масив. Приклади розрахунку вихідної матриці для шарів MaxPooling та AveragePooling зображено на рисунку 3.3:

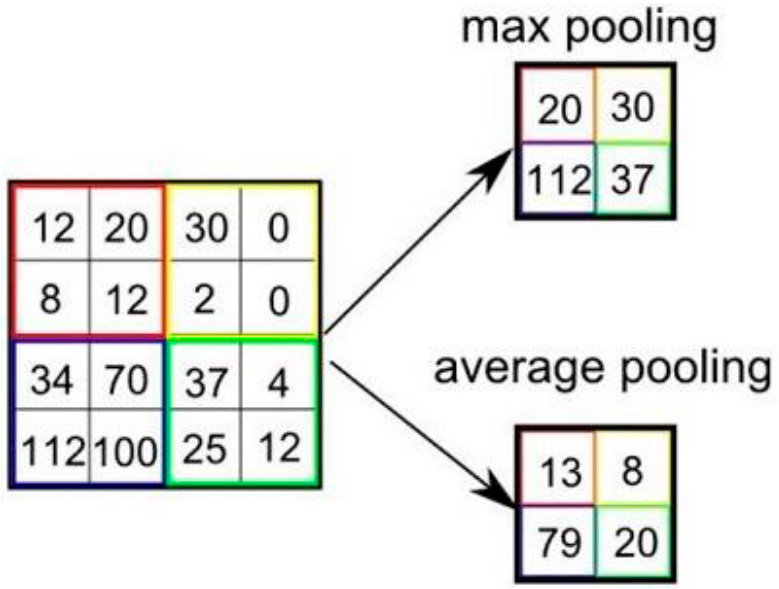

Рисунок 3.3 Приклади розрахунку вихідної матриці для шарів MaxPooling та Average Pooling

Вихідні карти ознак останнього шару згортки або пулінгу зазвичай згладжуються, тобто перетворюються на одновимірний масив чисел (або вектор) і з'єднуються з одним або декількома повноз’єднаними шарами, також відомими як щільні шари, в яких кожен вхід пов'язаний з кожним виходом вагою, яка навчається. Після того, як об'єкти, витягнуті згортоковими шарами і шарами субдискретизації, створені, вони зіставляються підмножиною повноз’єднаних шарів з кінцевими вихідними даними мережі, такими як ймовірності для кожного класу в задачах класифікації, тобто повноз’єднаний шар виконує завдання класифікації на основі функцій, витягнутих через попередні шари, і їх різних фільтрів. Останній повноз’єднаний шар зазвичай має таку саму кількість вихідних вузлів, що і кількість класів, та

використовує функцію активації SoftMax, яка була описана в попередньому пункті.

Отже, об’єднуючи між собою згорткові шари, шари пулінгу, повноз’єднані шари з функціями активації отримуємо архітектуру ЗНМ. Наведемо приклад найпростішої архітектури CNN (рисунок 3.4):

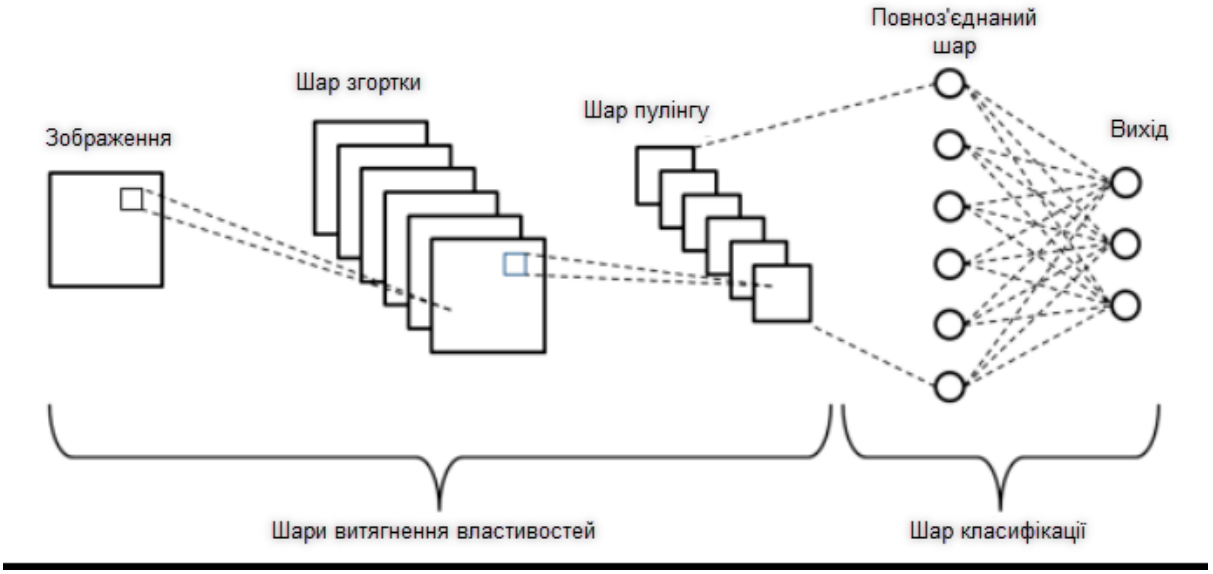

Рисунок 3.4 Архітектура простої згорткової нейронної мережі

3.6 Експеримент №1: результати генерації набору даних

Після того як будуть знайдені всі необхідні дані для проведення першого експерименту. Необхідно згенерувати набор даних, як показано на лістингу 3.14.

Лістинг 3.14 – Код генерації набору даних

```
$ python train.py --dataset generated_dataset --plot plot_generated_dataset.png
[INFO] loading images...
[INFO] compiling model...
[INFO] training network for 50 epochs...
Epoch 1/50
18/18 [==============================] - 1s 60ms/step - loss: 0.3191 - accuracy: 0.9220 - val_loss: 0.1986 - val_accuracy: 1.0000
Epoch 2/50
18/18 [==============================] - 0s 9ms/step - loss: 0.2276 - accuracy: 0.9858 - val_loss: 0.2044 - val_accuracy: 1.0000
Epoch 3/50
18/18 [==============================] - 0s 8ms/step - loss: 0.2839 - accuracy: 0.9574 - val_loss: 0.2046 - val_accuracy: 1.0000
...
Epoch 48/50
18/18 [==============================] - 0s 9ms/step - loss: 0.1770 - accuracy: 1.0000 - val_loss: 0.1768 - val_accuracy: 1.0000
Epoch 49/50
18/18 [==============================] - 0s 9ms/step - loss: 0.1767 - accuracy: 1.0000 - val_loss: 0.1763 - val_accuracy: 1.0000
Epoch 50/50
18/18 [==============================] - 0s 8ms/step - loss: 0.1767 - accuracy: 1.0000 - val_loss: 0.1758 - val_accuracy: 1.0000
[INFO] evaluating network...
              precision    recall  f1-score   support

        cats       1.00      1.00      1.00        25
        dogs       1.00      1.00      1.00        25

    accuracy                           1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50
```

Результати показують, що була отримана 100% точність без зусиль. Це можна побачити на графіку який зображений на рисунку 3.2.

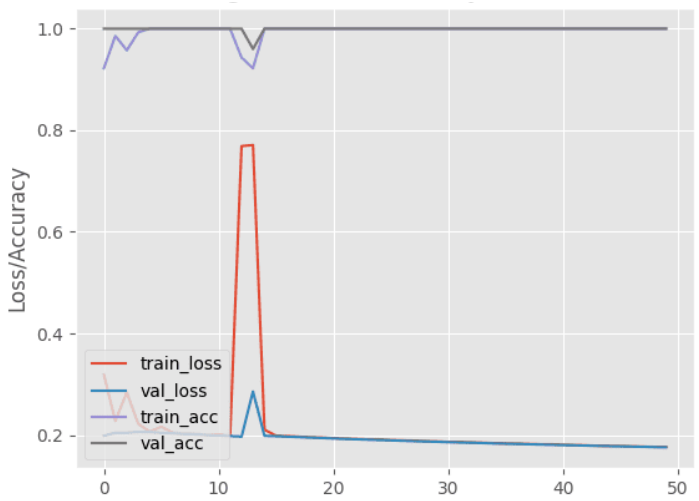

Рисунок 3.2 – Графік збільшення даних

Звичайно, це тривіальний, надуманий приклад. На практиці ніхто не буде брати лише одне зображення, а потім створюєте набір даних із 100 або 1000 зображень за допомогою розширення даних. Натомість буде набір даних із 100 зображень, а потім буде застосована генерація набору даних до цього набору даних — але знову ж таки, метою цієї роботи було продемонструвати на простому прикладі, щоб була можливість краще зрозуміти цей процес.

Більш популярна форма збільшення даних (на основі зображень) називається збільшенням даних на місці . При виконанні аугментації на місці наш Keras ImageDataGenerator буде: Прийняти партію вхідних зображень. Довільно трансформуйте вхідний пакет. Поверніть перетворену партію в мережу для навчання. Буде досліджено, як розширення даних може зменшити переобладнання та збільшити здатність нашої моделі до узагальнення за допомогою двох експериментів.

Щоб виконати це завдання, було використано підмножину набору даних Kaggle Dogs vs. Cats. Потім була навчена варіація ResNet з нуля на цьому наборі даних із розширенням даних і без нього.

3.7 Експеримент №2: отримання базової лінії

У цьому експерименті не було використано розширення даних, це можна побачити на лістингу 3.15.

Лістинг 3.15 – Код отримання базової лінії

```
$ python train.py --dataset dogs_vs_cats_small --plot plot_dogs_vs_cats_no_aug.png
[INFO] loading images...
[INFO] compiling model...
[INFO] training network for 50 epochs...
Epoch 1/50
187/187 [==============================] - 3s 13ms/step - loss: 1.0743 - accuracy: 0.5134 - val_loss: 0.9116 - val_accuracy: 0.5440
Epoch 2/50
187/187 [==============================] - 2s 9ms/step - loss: 0.9149 - accuracy: 0.5349 - val_loss: 0.9055 - val_accuracy: 0.4940
Epoch 3/50
187/187 [==============================] - 2s 9ms/step - loss: 0.9065 - accuracy: 0.5409 - val_loss: 0.8990 - val_accuracy: 0.5360
...
Epoch 48/50
187/187 [==============================] - 2s 9ms/step - loss: 0.2796 - accuracy: 0.9564 - val_loss: 1.4528 - val_accuracy: 0.6500
Epoch 49/50
187/187 [==============================] - 2s 10ms/step - loss: 0.2806 - accuracy: 0.9578 - val_loss: 1.4485 - val_accuracy: 0.6260
Epoch 50/50
187/187 [==============================] - 2s 9ms/step - loss: 0.2371 - accuracy: 0.9739 - val_loss: 1.5061 - val_accuracy: 0.6380
[INFO] evaluating network...
              precision    recall  f1-score   support

        cats       0.61      0.70      0.65       243
        dogs       0.67      0.58      0.62       257

    accuracy                           0.64       500
   macro avg       0.64      0.64      0.64       500
weighted avg       0.64      0.64      0.64       500
```

Подивившись на необроблений звіт про класифікацію, ви побачите, що ми отримуємо 64% точності , але є проблема, яку можна побачити, якщо подивитися на графік сюжету, пов’язаний з нашим тренуванням на рисунку 3.3

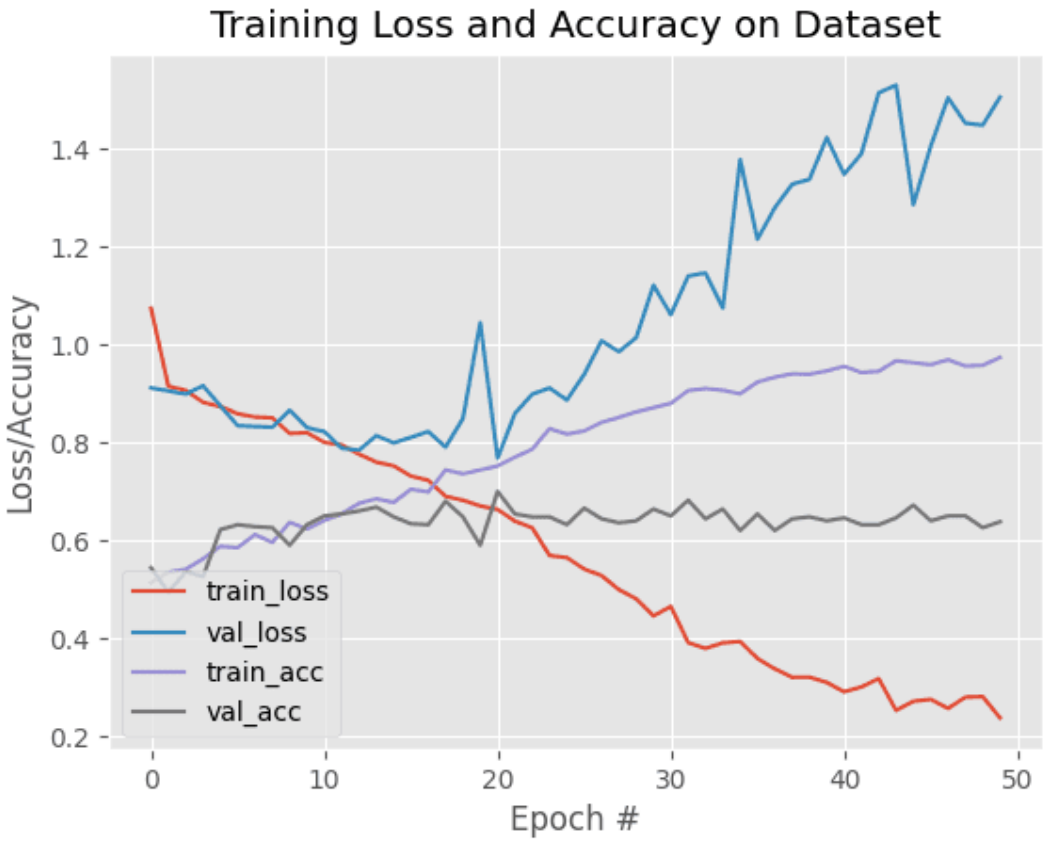

Рисунок 3.3 – Графік сюжету тренувань для другого експерименту

Для експерименту №2 ми не виконували збільшення даних. У результаті виходить сюжет із сильними ознаками переобладнання. Відбувається різке переобладнання — приблизно в епоху 15 ми бачимо, що наша втрата перевірки починає зростати, а втрата навчання продовжує падати. До 20 епохи зростання втрат підтвердження особливо помітно.
Така поведінка свідчить про переобладнання.

Рішення полягає в тому, щоб зменшити потужність моделі та виконати регуляризацію.

3.8 Експеримент №3: покращення результатів

У третьому експерименті було досліджено, як розширення даних може діяти як форма регуляризації.

Лістинг 3.16 – Код покращення даних за допомогою збільшення даних

```
[INFO] loading images...
[INFO] performing 'on the fly' data augmentation
[INFO] compiling model...
[INFO] training network for 50 epochs...
Epoch 1/50
187/187 [==============================] - 3s 14ms/step - loss: 1.1307 - accuracy: 0.4940 - val_loss: 0.9002 - val_accuracy: 0.4860
Epoch 2/50
187/187 [==============================] - 2s 8ms/step - loss: 0.9172 - accuracy: 0.5067 - val_loss: 0.8952 - val_accuracy: 0.6000
Epoch 3/50
187/187 [==============================] - 2s 8ms/step - loss: 0.8930 - accuracy: 0.5074 - val_loss: 0.8801 - val_accuracy: 0.5040
...
Epoch 48/50
187/187 [==============================] - 2s 8ms/step - loss: 0.7194 - accuracy: 0.6937 - val_loss: 0.7296 - val_accuracy: 0.7060
Epoch 49/50
187/187 [==============================] - 2s 8ms/step - loss: 0.7071 - accuracy: 0.6971 - val_loss: 0.7690 - val_accuracy: 0.6980
Epoch 50/50
187/187 [==============================] - 2s 9ms/step - loss: 0.6901 - accuracy: 0.7091 - val_loss: 0.7957 - val_accuracy: 0.6840
[INFO] evaluating network...
              precision    recall  f1-score   support

        cats       0.72      0.56      0.63       243
        dogs       0.66      0.80      0.72       257

    accuracy                           0.68       500
   macro avg       0.69      0.68      0.68       500
weighted avg       0.69      0.68      0.68       500
```

Зараз було досягнуто 69% точності , що більше, ніж попередні 64% точності. Але що важливіше, не має змоги переобладнати більше, це можливо побачити з графіка на рисунку 3.4.

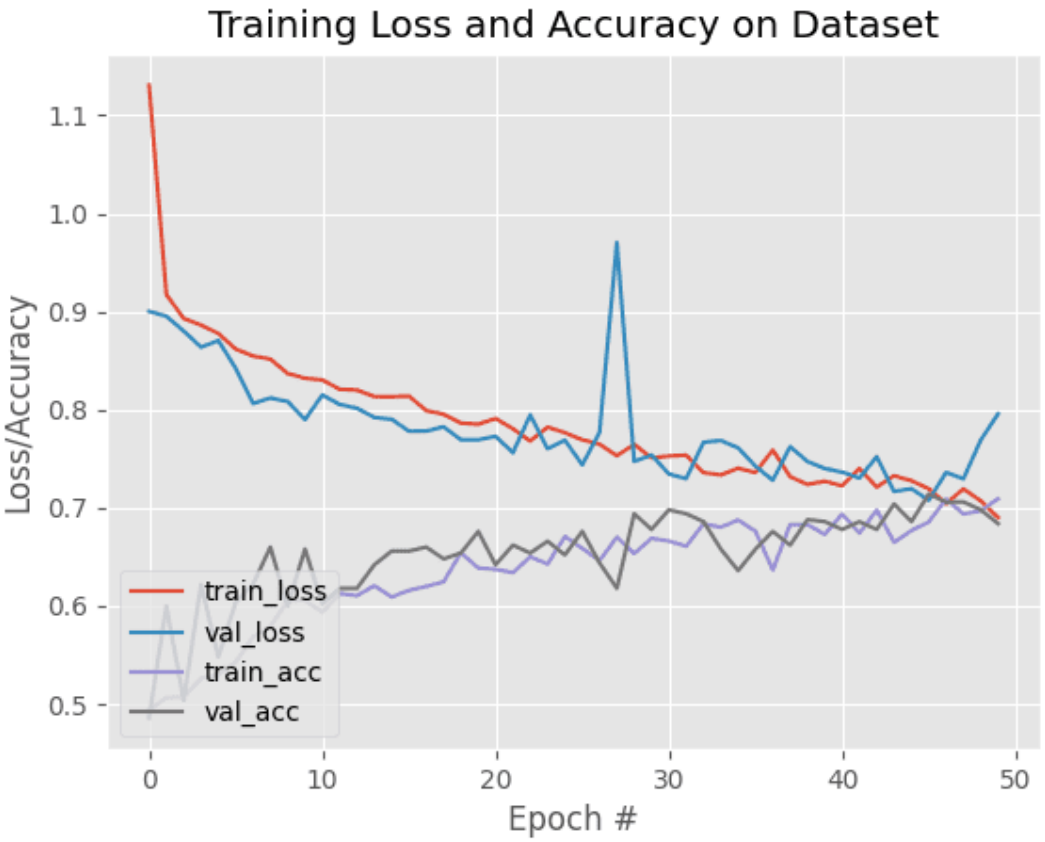

Рисунок 3.4 – Графік сюжету тренувань для третього експерименту

Для експерименту №3 виконано збільшення даних за допомогою Keras для партій зображень на місці. Навчальний графік не має ознак переобладнання цією формою регуляризації. Треба звернути увагу, як перевірка та втрата навчання падають разом із невеликими розбіжностями. Аналогічно, точність класифікації як для навчання, так і для розділів перевірки також зростає разом. Використовуючи аугментацію даних, була усунена проблема з переобладнанням. Майже у всіх ситуаціях, якщо немає вагомих причин цього не робити, то треба виконувати збільшення даних під час навчання власних нейронних мереж.

ВИСНОВКИ

У рамках кваліфікаційної роботи були проведені експерименти з розширенням даних за допомогою Keras DataImageGenerator. Була використана така мова програмування як Python, а також середовище розробки Jupyter notebook, та відкрита нейромережна бібліотека Keras, написана мовою Python. Завдяки цьому проекту ми зрозуміли, як застосувати розширення даних до власних наборів даних, використовуючи всі три методи, за допомогою Keras ImageDataGenerator.

Актуальність цього проекту полягає у тому що при застосуванні аугментації даних, змогли досягти підвищення узагальненості моделі. Враховуючи, що дана мережа постійно отримує нові, дещо змінені версії вхідних даних, мережа може вивчати більш надійні функції. Під час тестування проходить оцінка навченої мережі на основі не модифікованих даних тестування — у більшості випадків помічається підвищення точності тестування, можливо, за рахунок невеликого зниження точності навчання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is Python [Електронний ресурс] – URL: https://www.w3schools.com/python/python_intro.asp (дата звернення 19.05.2022).

2. Keras: библиотека глубокого обучения на основе Python [Електронний ресурс] – URL: https://keras-cn.readthedocs.io/en/latest (дата звернення 12.05.2022).

3. Розробка багатоагентних структур для вирішення проблем медичної системи діагностування / О.Я.Кузьомін, О.О. Василенко, Свістунов І.О. // Радіоелектроніка та інформатика. 2020. №2. С. 47 – 54.

4. Розробка структур медичних агентів для вирішення проблем медичної системи діагностування /О.Я. Кузьомін, О.О. Василенко,Горшколєпов А.В. // Радіоелектроніка та інформатика. 2020. №2. С.55- 65.

5. Research of the intellectual system of knowledge search in Databases / Oleksii Vasilenko, Oleksandr Kuzomin, Bohdan Maliar //International Journal "Information Models and Analyses" Volume 7, Number 4. PP.2019. 327 – 338.

6. Research of medical diagnostic data search methods / Oleksii Vasilenko, Oleksandr Kuzomin, OleksandrShapoval // International Journal "Information Models and Analyses" Volume 7, Number 4.PP.2019. 339 – 349.

7. Intellectual Models and Means of the Biometric System Dynamics of Rinosinusite / Oleksii Vasilenko,Oleksandr Kuzomin, Tatyana Khripushina // International Journal "Information Models andAnalyses" Volume 7, Number 4. PP.2019. 350 – 361.

8. Forming Medical Database and Knowledge for Diagnostic Disease / Oleksii Vasilenko, Oleksandr KuzominVladislav Shvets. //International Journal

"Information Models and Analyses" Volume 7, Number 4. PP.2019. 362 – 372.

9. Automated Tests for Errors in Computer System ‘Environment’/ Oleksii Vasilenko, Oleksandr Kuzomin. // International Journal"Information Models and Analyses" Volume 7, Number 4. PP.2019. 373 – 384.

10. Developing methods based on text mining technology to Improve the quality and speed of automatic clustering of Documents / Oleksii Vasilenko, Oleksandr Kuzomin, Artem Mertsalov // International Journal"Information Models and Analyses" Volume 7, Number 4.PP.2019. 385 – 397.

11. Python main function [Електронний ресурс] – URL: https://www.journaldev.com/17752/python-main-function (дата звернення 14.05.2022).

12. TensorFlow 2.0 на Ubuntu [Електронний ресурс] – URL: https://pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-ubuntu/ (дата звернення 16.05.2022).

13. TensorFlow 2.0 на macOS [Електронний ресурс] – URL: https://pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-macos/ (дата звернення 16.05.2022).

14. Hubel D. H., Wiesel T. N. Receptive fields and functional architecture of monkey striate cortex. The Journal of Physiology. 1968. Vol. 195, no. 1. P. 215–243. DOI: https://doi.org/10.1113/jphysiol.1968.sp008455 (дата звернення 16.05.2022).

15. Ciresan, Dan Claudiu, et al. Flexible, high performance convolutional neural networks for image classification. Twenty-Second International Joint Conference on Artificial Intelligence. 2011. Vol. 2. P. 1237-1242.

16. Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2021). A survey of convolutional neural networks: analysis, applications, and prospects. IEEE Transactions on Neural Networks and Learning Systems.

17. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3), 211-252.

18. Kirichenko, L., Radivilova, T., & Bulakh, V. (2020). Machine learning classification of multifractional Brownian motion realizations.

19. Kirichenko, L., Radivilova, T., Bulakh, V., Zinchenko, P., & Alghawli, A. S. (2020, August). Two approaches to machine learning classification of time series based on recurrence plots. In 2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP) (pp. 84-89). IEEE.

20. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.

21. Minaee, S., Boykov, Y. Y., Porikli, F., Plaza, A. J., Kehtarnavaz, N., & Terzopoulos, D. (2021). Image segmentation using deep learning: A survey. IEEE transactions on pattern analysis and machine intelligence.

22. Xuan, X., Zhang, X., Kwon, O. H., & Ma, K. L. (2022). VAC-CNN: A Visual Analytics System for Comparative Studies of Deep Convolutional Neural Networks. IEEE Transactions on Visualization and Computer Graphics, 28(6), 2326-2337.

23. West, Jeremy; Ventura, Dan; Warnick, Sean (2007). "Spring Research Presentation: A Theoretical Foundation for Inductive Transfer". Brigham Young University, College of Physical and Mathematical Sciences. Archived from the original on 2007-08-01. Retrieved 2007-08-05.

24. Szegedy C., Vanhoucke V., Ioffe S., Shlens J., Wojna Z., Google ,University College London Computer Vision and Pattern Recognition Rethinking the Inception Architecture for Computer Vision, 2016 pp 2818-2826 DOI: https://doi.org/10.1109/CVPR.2016.308 (дата звернення 20.05.2022).

25. 22. Bengio Y., Courville A., Goodfellow I. Deep Learning. MIT Press, 2016. 800 p.