КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра дослідження операцій

Випускна кваліфікаційна робота магістра

за спеціальністю 113 «Прикладна математика»

на тему:

ЗАСТОСУВАННЯ НЕЙРОМЕРЕЖ ДО РОЗПІЗНАВАННЯ АВТОМОБІЛІВ

Студента 2 курсу магістратури

кафедри дослідження операцій

Огійчука Артема Олександровича

Науковий керівник: доцент

Проскурін Данило Павлович

Робота заслухана на засіданні кафедри дослідження операцій та рекомендована до захисту в ДЕК, протокол "_____" __ _____ 2021р.

Завідувач кафедри дослідження операцій доктор фізико-математичних наук Іксанов О.М.

Київ-2021

РЕФЕРАТ

Обсяг роботи 40 сторінок, 20 ілюстрацій, 9 джерел посилань, 2 додатки

МЕТОД НЕЙРОМЕРЕЖ, PYTHON, АВТОМОБІЛІ, РОЗПІЗНАВАННЯ, YOLO.

Об’єктом роботи є процес розробки програми, що розпізнає автомобілі за допомогою Python. Предметом роботи є проект Python в середовищі Jupyter Notebook.

Метою роботи є створення програми для розв’язання задачі розпізнавання автомобілів із застосуванням нейронних мереж.

Інструменти розроблення: безкоштовна версія середовища розробки Jupyter Notebook, мова програмування Python, бібліотеки argparse, matplotlib.pyplot, matplotlib.pyplot, scipy.io, scipy.misc, numpy, pandas, tensorflow, keras, keras.layers, keras.models, yolo_utils, yad2k.models.keras_yolo, PIL та os, модулі imshow, backend, Input, Lambda, Conv2D, load_model, Model, yolo_head, yolo_boxes_to_corners, read_anchors, read_classes, generate_colors, yolo_loss preprocess_image, preprocess_true_boxes, draw_boxes, scale_boxes, and yolo_body.

Результат роботи: виконано загальний огляд застосування нейромереж до задачі класифікації, проведено аналіз існуючих підходів розпізнавання автомобілів, розроблено програмний засіб на основі принципу YOLO, який розпізнає автомобілі на зображеннях. Програмний продукт може застосовуватись у машинах без водія, у камерах спостереження, у каптчах.

ЗМІСТ

Вступ

Оцінка сучасного стану об’єкта розробки. Одним з важливих напрямів розробки IT-компаній є штучний інтелект та “навчання” машин. Розробники хочуть, щоб їх машини бачили навколишній світ, уміли аналізувати та розпізнавати об’єкти, що вони бачать, так, як це робимо ми - люди, та й взагалі усі живі істоти. Однією з найпопулярніших технологій у сучасній IT-сфері, що допомагає машині розпізнавати об’єкти та приймати найкращі рішення, являється розробка штучних нейронних мереж. Вчені усього прогресивного світу займаються цим вже близько сімдесяти років. Але з появою більш швидких машин і великих баз даних стався прорив у цій області. Навчання - найбільш ресурсовитратна частина в розробці нейромережі. Для цього потрібні, наприклад, мільйони наборів вхідних даних і потужні машини, чого не було раніше. А зараз є і загальнодоступна об'ємна база даних зразків рукописного написання цифр MNIST, і відеокарти з високою продуктивністю, і технології прискореного навчання. Крім того, широкий розголос нейронні мережі отримали завдяки наперед навченим прототипам, які дозволили на їх базі створювати додатки і сервіси без глибокої підготовки.

Мета й завдання роботи. Метою дипломної роботи є створення програмного засобу для задачі розпізнавання автомобілів із застосуванням нейронних мереж. Для досягнення цієї мети поставлено такі завдання.

- ➔ Дослідити існуючі засоби розпізнавання автомобілів
- ➔ Розробити модель нейронної мережі для вирішення даної задачі
- ➔ Реалізувати програмний продукт

Об’єкт, методи й засоби розроблення. Об’єктом розроблення програмного засобу є процес розпізнавання автомобілів за допомогою програмного засобу.

Розробці програмного засобу передувало створення моделі нейромережі , яка забезпечить розпізнавання автомобілів.

В якості інструменту створення програмного засобу було обрано Jupyter Notebook мовою програмування Python. Використано бібліотеки argparse, matplotlib.pyplot, matplotlib.pyplot, scipy.io, scipy.misc, numpy, pandas, tensorflow, keras, keras.layers, keras.models, yolo_utils,

yad2k.models.keras_yolo, PIL та os, модулі imshow, backend, Input, Lambda, Conv2D, load_model, Model, yolo_head, yolo_boxes_to_corners, read_anchors, read_classes, generate_colors, yolo_loss preprocess_image, preprocess_true_boxes, draw_boxes, scale_boxes, and yolo_body.

Можливі сфери застосування. Програмний продукт може застосовуватись у машинах без водія, у камерах спостереження, у каптчах, а також у більш складних програмних системах, яким потрібно розпізнавати автомобілі.

Розділ 1. НЕЙРОННІ МЕРЕЖІ У ЗАДАЧАХ КЛАСИФІКАЦІЇ

1.1 Штучна нейронна мережа

Штучні нейронні мережі [1-5] — це обчислювальні системи, які створені на подобі біологічних нейронних мереж, які, в свою чергу, складають мозок тварин. Такі системи навчаються задач (поступово покращують свою продуктивність на них), розглядаючи приклади, без конкретного програмування під кожну окрему задачу.

ШНМ базується на сукупності з'єднаних вузлів, назва яких - штучні нейрони (аналогічно до біологічних нейронів у головному мозку тварин). Кожне таке з'єднання (за аналогією з синапсом) між штучними нейронами може передавати сигнал від одного до іншого. Штучний нейрон може обробляти сигнал, який він отримав, а потім сигналізувати штучним нейронам, які приєднані до нього.

В поширених реалізаціях ШНМ сигнал, який є на з'єднанні між штучними нейронами, це - дійсне число, а вихід кожного штучного нейрону, в свою чергу, обчислюється за допомогою нелінійної функції суми його входів. Штучні нейрони та їх з'єднання зазвичай мають вагу, яка міняється в перебігу навчання. Вага використовується для збільшення або зменшення сили сигналу на з'єднанні. Зручно, що штучні нейрони можуть мати такий поріг, що сигнал надсилається лише тоді, коли сукупний сигнал перетинає цей поріг. Зазвичай штучні нейрони організовано в шари. Окремі шари можуть виконувати різноманітні види перетворень своїх входів. Сигнали проходять від вхідного (першого) до вихідного (останнього) шару. При цьому можливе проходження шарів декілька разів.

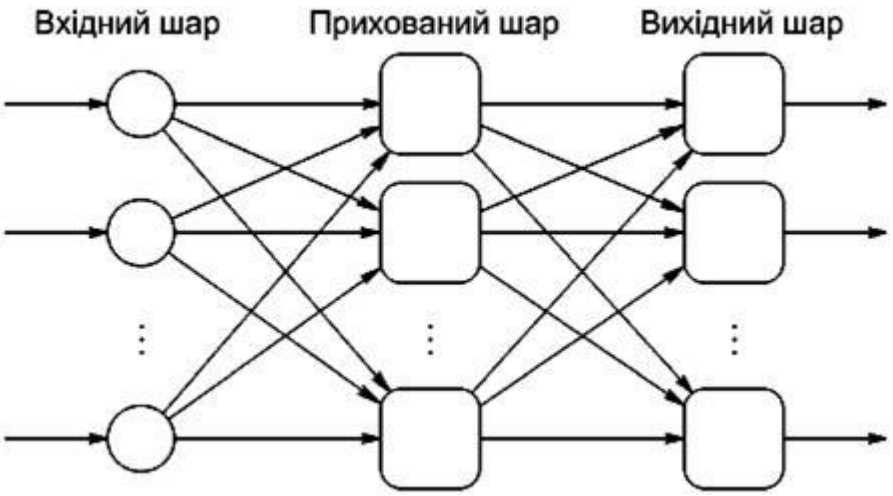

Рисунок 1. Багатошарова нейронна мережа.

Нестандартний характер обробки інформації - сильна сторона цих комплексів. Вона кодується і запам’ятовується в розподілі зв’язків між нейронами і в їх силі, а не в окремих елементах пам’яті, тому стан кожного нейрона визначається станом багатьох інших нейронів, які пов’язані з ним. Таким чином, втрата одного або декількох зв’язків в цілому не робить істотного впливу на результат роботи системи. Це, зі свого боку, забезпечує її високу надійність.

Висока «природна» стійкість до перешкод і функціональна надійність стосуються як зашумлених (спотворених) потоків інформації, так і в сенсі відмов окремих процесорних елементів. Цим забезпечуються достовірність обробки інформації і висока оперативність, а просте перенавчання і донавчання мереж дозволяють при зміні зовнішніх чинників здійснювати перехід на новий рівень вирішуваних завдань без запізнень.

Завдяки наведеним вище перевагам нейромережевої обробки даних визначаються сфери їх застосування:

– адаптивне управління і передбачення.

– класифікація інформації в реальному масштабі часу;

– обробка і аналіз зображень;

– планування, застосування сил і засобів у великих масштабах;

– автоматизована система швидкого пошуку інформації;

– розпізнавання мови незалежно від диктора;

– вирішення трудомістких задач оптимізації;

– обробка високошвидкісних цифрових потоків;

Кожен нейрон складається із входів (input signals), синаптичних ваг (synaptic weights), суматора (summing junction), функції активації (activation function) та виходу (output). Нейрон повинен мати щонайменше 1 вхід. На кожен вхід нейрона подається сигнал що може бути негативним або позитивним значенням. Виходи таких нейронів можуть бути з'єднані із входами інших нейронів. Таким чином формується мережа.

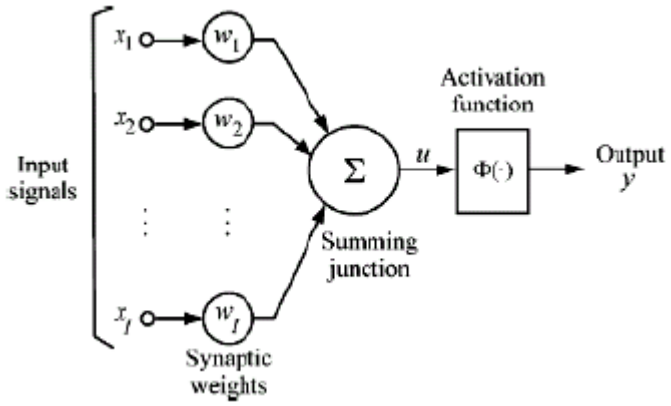

Рисунок 2. Внутрішня структура штучного нейрона.

Кожному входу нейрону співставляється синаптична вага (або просто вага). Вона також може мати негативне або позитивне значення. Суматор приймає щонайменше 2 значення: зміщення та значення для кожного входу. Значення на виході суматора є входом для функції активації та є сумою зважених входів. Функція активації визначає залежність сигналу на виході від зважених сигналів на його входах. Існує багато різних функцій активації. Кожна з них має свої переваги та недоліки. Серед них найбільш

поширені: кусково-лінійна, випрямлена лінійна (ReLU), порогова, сигмоїдальна та тангенціальна. Сигмоїдальна та тангенціальна функції дають можливість таким мережам вирішувати складні нелінійні задачі, тому що вони є нелінійними.

1.2 Застосування нейромереж для задач класифікації

Вирішення задачі класифікації є одним з найважливіших застосувань нейронних мереж.

В будь-якому завданні класифікації потрібно віднести наявні статичні зразки (рукописні цифри чи літери, звукові сигнали, характеристики фінансового становища) до певних класів. Різноманітність прикладів, що виникають в реальному світі, практично нескінченна. Ефективність класифікації залежить від способу представлення цих форм. У структурному розпізнаванні образів зразки описуються своєю структурою, подібно до того, як це робиться в граматиці мови. При статистичному підході до розпізнавання зразок задається вектором $\bar{x}$, компоненти якого є різними характеристиками зразка. Класифікатор відносить об'єкт x до того чи іншого класу C у відповідності до визначеного розбиттям N-вимірного простору, яке називається простором входів.

Універсальним засобом апроксимації функцій, який дає змогу його використовувати в вирішенні проблеми класифікації, є мережі з прямим зв'язком. Нейронні мережі виявляються найбільш ефективним способом класифікації, оскільки генерують велику кількість регресійних моделей (що можна використати при розв'язанні задачі).

Але при практичному застосуванні нейронних мереж виникає ряд проблем. Так наприклад, неможливо наперед визначити якої складності та

розміру може знадобитись мережа для найкращого результату. Також, щоб побудувати класифікатор необхідно визначити параметри, які впливають на прийняття рішення про те, до якого класу віднести зразок. При цьому можемо зіштовхнутися із ситуацією, коли кількість параметрів мала, то один і той же набір вихідних даних відповідає зразкам, які перебувають в різних класах. Тоді неможливо навчити нейронну мережу, і система не буде коректно працювати. Вихідні дані обов'язково не мають бути суперечливими. Для вирішення цього питання треба збільшити розмірність простору ознак (кількість компонент вхідного вектора). Але при збільшенні розмірності можемо зіштовхнутися із тим, що коли кількість прикладів може бути недостатньою для навчання мережі, і замість узагальнення вона просто запам'ятає приклади з навчальної вибірки і не буде коректно працювати. Отже, при визначенні ознак необхідно знайти компроміс з їх кількістю.

Наступним кроком необхідно визначити спосіб представлення вхідних даних для нейромережі, тобто визначити спосіб нормування. Нормування потрібне, тому що нейронні мережі працюють з даними, які представлені числами в діапазоні $[0, 1]$, а вихідні дані можуть мати довільний діапазон або взагалі бути нечисловими.

При наявності двох класів задача класифікації може бути вирішена за допомогою мережі з навіть одним нейроном у вихідному шарі, який може приймати одне з двох значень: нуль або один, залежно від того, до якого класу належить зразок. Також при наявності декількох класів виникає проблема, яка пов'язана з поданням цих даних для виходу мережі. Найпростішим способом представлення вихідних даних в цьому випадку є вектор, компоненти якого відповідають різним номерам класів. При цьому і-а компонента вектора відповідає і-му класу, а всі інші компоненти рівні

нулю. Тоді, наприклад, другому класу буде відповідати одиниця на другому виході мережі і нуль на інших. При інтерпретації результату зазвичай вважається, що номер класу визначається номером виходу мережі, на якому з'явилося максимальне значення.

Велике значення має правильний вибір розміру мережі. Часто буває неможливо побудувати невелику і якісну модель, а велика модель замість того, щоб виробляти апроксимацію, буде просто запам'ятовувати приклади з навчальної вибірки, що призведе до некоректної роботи класифікатора. Є два основні підходи до побудови мережі - конструктивний і деструктивний. При першому з них спочатку береться мережа мінімального розміру і поступово збільшується до моменту досягнення необхідної точності. В такому випадку на кожному кроці її заново навчають. Часто застосовують метод каскадної кореляції, суть якого полягає в тому, що після закінчення епохи відбувається коригування мережі з метою мінімізації помилки. При деструктивному підході спочатку береться мережа досить великого обсягу, і потім з неї видаляються зв'язки та вузли, що мало впливають на рішення.

Розділ 2. МЕТОДИ НАВЧАННЯ МЕРЕЖІ

2.1. Навчання з вчителем (supervised learning)

На сьогоднішній день відомо три методи навчання нейронних мереж, основою яких є особливості машинного навчання:

- навчання з вчителем (supervised learning);
- навчання без вчителя (unsupervised);
- навчання з підкріпленням (reinforcement learning).

Навчання з вчителем (supervised learning) — передбачає, що для кожного вхідного вектора (x_i) існує вектор вихідних значень (d_i). Разом ці два вектора називаються навчальною парою (x_i,d_i), а множина навчальних пар — навчальною вибіркою. Процес навчання зводиться до того, що почергового подаються на вхід нейронної мережі навчальних пар, вираховуються похибки між дійсним і бажаним значенням $\square = y - d$ та корегуються параметри мережі в бік зменшення цієї похибки.

Для вирішення задачі за допомогою методу навчання з учителем, потрібно виконати наступні кроки:

1. Потрібно визначити тип навчальних прикладів. Перш за все, користувач повинен вирішити, які дані використовувати як свою навчальну вибірку. Наприклад, якщо ви аналізуєте рукописний текст, це може бути єдина рукописна літера, ціле рукописне слово або цілий рукописний рядок.

2. Зібрати навчальну вибірку. Збирається набір вхідних об'єктів та відповідних виходів. Таким чином, навчальна вибірка повинна бути представником реального використання функції.
3. Визначити представлення ознак вводу вивченої функції. Точність вивченої функції сильно залежить від представлення об'єкту вводу. Здебільшого, об'єкт вводу перетворюється в вектор ознак. Він містить ряд ознак, що описують об'єкт. Кількість ознак не повинна бути занадто великою через прокляття розмірності; але, у той же час, вона повинна містити достатньо інформації для того, щоб результати спрогнозувалися якумога точніше.
4. Визначити структуру вивченої функції та відповідний алгоритм навчання. Наприклад, інженер може вибрати метод опорних векторів або дерева рішень.
5. Завершити дизайн. Запускаємо алгоритм навчання на зібраній навчальній вибірці. Деякі алгоритми навчання з вчителем вимагають від користувача певних контрольних параметрів. Ці параметри можна регулювати через перехресну перевірку, або оптимізуючи продуктивність на підмножині (називається набором перевірок) навчальної вибірки.
6. Оцінити точність вивченої функції. Після налаштування та навчання параметрів продуктивність результуючої функції потрібно виміряти на тестовій вибірці, яка повинна відрізнятися від навчальної вибірки.

2.2. Навчання без вчителя (unsupervised learning)

Навчання без вчителя є одним із способів машинного навчання, при вирішенні яких випробовувана система самостійно навчається виконувати поставлене завдання, роблячи це без втручання з боку експериментатора. З кібернетичної точки зору, це є одним з видів кібернетичного експерименту. Зазвичай, це підходить тільки для тих задач, в яких вже відомий опис множини об'єктів (навчальна вибірка), і необхідно виявити внутрішні взаємозв'язки, залежності, закономірності, що існують між цими об'єктами.

Навчання без вчителя часто протиставляється до навчання з учителем, коли для кожного об'єкта, що навчається, примусово задається «правильна відповідь», і потрібно знайти залежність між реакціями системи та стимулами.

Попри численні прикладні застосування, навчання з вчителем критикувалося за свою неправдоподібність, якщо дивитися з точки зору біології. Дійсно, важко уявити навчальний механізм в мозку, який зумів би порівнювати бажані і дійсні значення виходів, при цьому виконуючи корекцію за допомогою зворотного зв'язку. Припустимо існування подібного механізму в мозку. Звідки тоді виникають бажані виходи? Навчання без вчителя з цього боку є більш правдоподібною моделлю навчання в біологічній системі. Кохонен і багато інших вчених розвили цю модель. Вона не потребує цільового вектора для виходів, а, отже, не буде вимагати порівняння з встановленими ідеальними відповідями. Навчальна множина складається тільки з вхідних векторів. Навчальний алгоритм, зі свого боку, налаштовує ваги мережі так, щоб вихідні вектори виходили узгодженими, тобто щоб пред'явлення досить близьких вхідних векторів давало однакові виходи. Отже, процес навчання виділяє статистичні

властивості навчальної множини, а потім групує схожі вектори в класи. Пред'явлення на вхід вектора з цього класу дасть певний вихідний вектор, проте до навчання неможливо передбачити, який саме вихід буде вироблятися даним класом вхідних векторів. Таким чином, виходи подібної мережі повинні трансформуватися в якусь зрозумілу форму, що зумовлена процесом навчання. І це не є серйозною проблемою. Зазвичай, не складно ідентифікувати зв'язок між входом і виходом, що встановила мережа.

2.3. Навчання з підкріпленням (reinforcement learning)

Навчання з підкріпленням - це такий собі проміжний варіант двох попередніх методів . Замість «вчителя» в схему навчання вводиться блок, який називається «критика». Він відслідковує реакцію середовища на вхідний сигнал, спирається на неї і визначає евристичну похибку, яку покладено в процес навчання мережі. Вказаний метод базуєтьмч на відповідних правилах навчання, що визначають основні особливості їх застосування.

У машинному навчанні середовище зазвичай формулюється як марковський процес вирішення (МПВ), адже багато алгоритмів навчання з підкріпленням для цього контексту використовують методики динамічного програмування. Основною відмінністю між класичними методиками й алгоритмами навчання з підкріпленням є те, що останні не потребують знання про МПВ, а також те, що вони орієнтовані на великі МПВ, в яких точні методи стають просто нездійсненними.

Від стандартного навчання з учителем навчання з підкріпленням відрізняється тим, що пари правильних входів/виходів ніколи не

представляються, а недостатньо оптимальні дії явно не виправляються. Також присутній акцент на інтерактивній продуктивності, що включає в себе знаходження балансу між дослідженням (незвіданої території) та використанням (поточного знання. Компроміс між дослідженням та використанням у навчанні з підкріпленням найбільше вивчався за допомогою задачі багаторукого бандита та скінченні МПВ.

Розділ 3. МЕТОДИ РОЗПІЗНАВАННЯ АВТОМОБІЛІВ

3.1 Аналіз існуючих підходів

Класифікація. У класифікації зображень алгоритм розглядає зображення і може відповісти: автомобіль це чи ні.

Рисунок 1. Приклад класифікації.

Проблема класифікації полягає в тому, що недостатньо знати, що тут зображена машина. Ми також хочемо знати, де саме знаходиться цей автомобіль.

Класифікація з локалізацією. У цьому випадку ми не лише визначаємо, що це автомобіль, але алгоритм також відповідає за розміщення обмежувального контуру або малювання червоного прямокутника навколо положення автомобіля на зображенні.

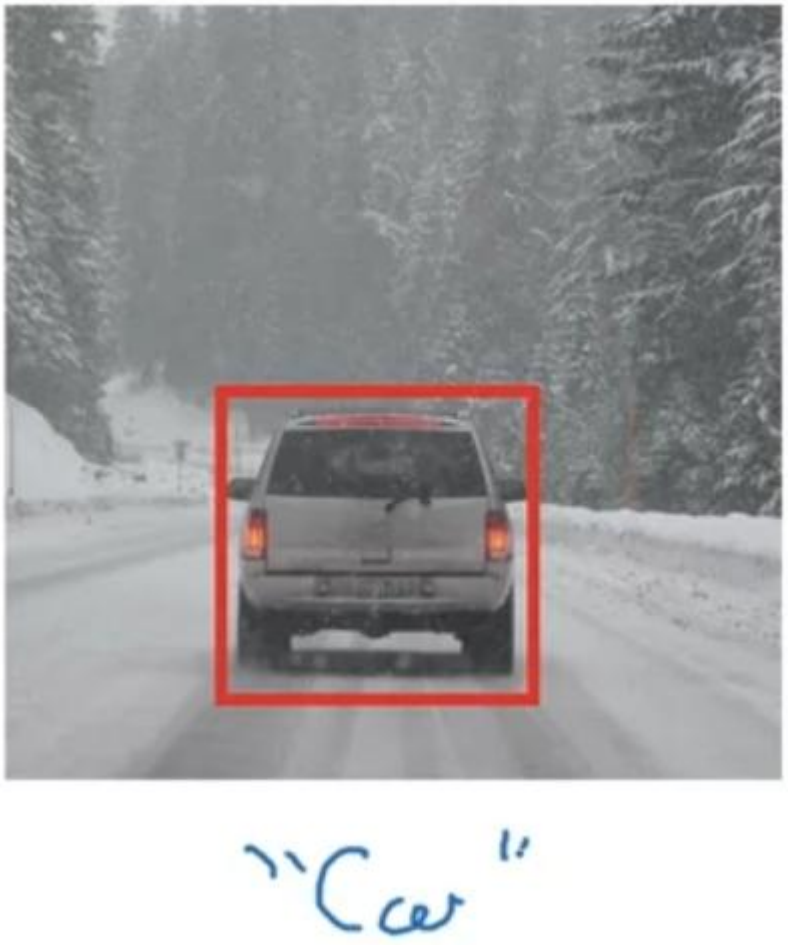

Рисунок 2. Приклад класифікації з локалізацією.

Отже, ми вирішили свою попередню проблему, але тепер ми хочемо знати, чи є на зображенні інші машини чи інші об’єкти, такі як автобуси, пішоходи, дорожні знаки тощо.

Розпізнавання за допомогою контуру, що ковзає (Sliding window detection). Візьмемо невелике прямокутне вікно і розмістимо його на зображенні, як на рисунку 3.

Рисунок 3. Контур 1 позиція 1.

У програмі, швидше за все, буде сказано, що на маленькому червоному квадраті немає машини. У алгоритмі розпізнавання за допомогою контуру, що ковзає, ви потім передаєте як вхід друге зображення, обмежене цим червоним контуром, трохи зміщеним, як на рисунку 4.

Рисунок 4. Контур 1 позиція 2.

Потім ви продовжуєте рухатись, доки контур не проковзає по кожному положенню зображення. Ідея полягає в тому, що ви в основному проходите кожну область такого розміру, передаєте безліч маленьких обрізаних зображень і класифікуєте нуль або один для кожної позиції як певний крок. Одна така ітерація і є контуром, що ковзає по всьому зображенню. Потім ви повторюєте це, але тепер використовуєте більший контур. Отже, тепер ви берете трохи більший регіон і запускаєте цей регіон, як на рисунку 5.

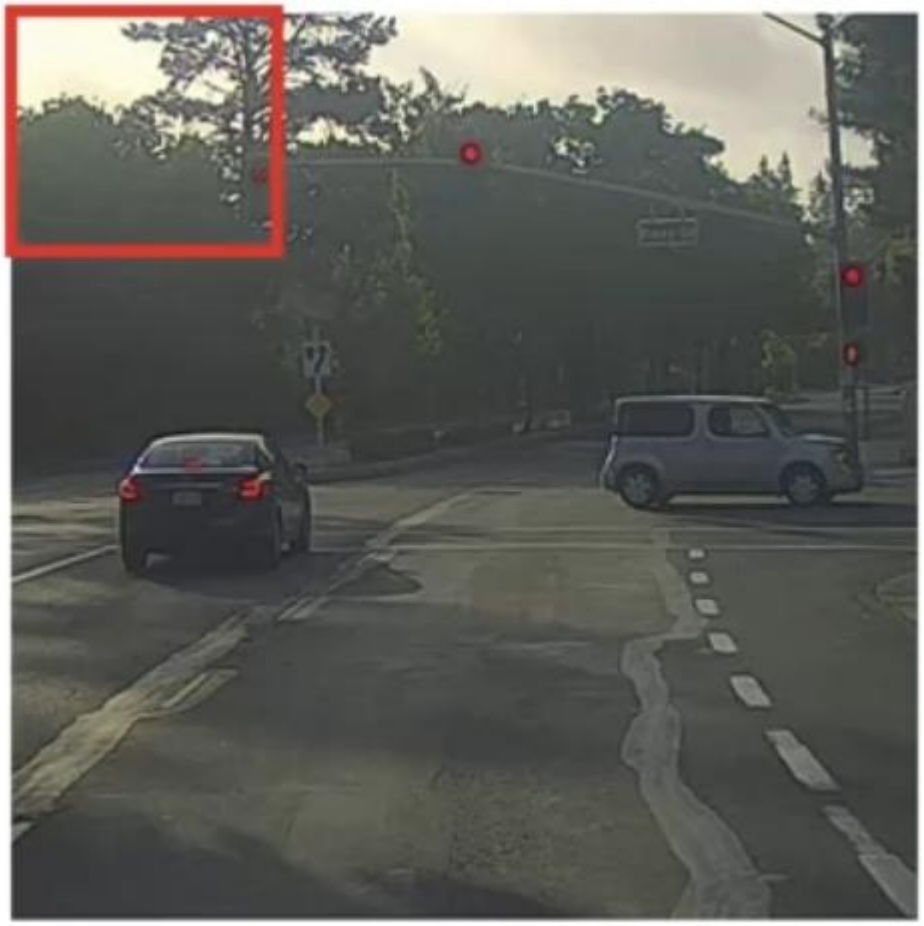

Рисунок 5. Контур 2.

Потім ми повторюємо наш алгоритм для вікна 2 так само, як для вікна 1. Потім знову беремо більший контур і знову повторюємо алгоритм.

Величезним недоліком Sliding Window Detection є обчислювальні витрати, тому що нам потрібно взяти багато вікон і пройти багато регіонів.

3.2 Обчислювальний експеримент та аналіз його результатів

Я розробив програмне забезпечення, яке включає нейронну мережу, яка може розпізнавати автомобілі на зображеннях розміром 720x1270 (які попередньо обробляються на зображення 608x608). Для виявлення об'єктів я обрав модель YOLO [6,7].

"Ти дивишся лише один раз" (You Only Look Once - YOLO) - популярний алгоритм, оскільки він досягає високої точності, а також може працювати в режимі реального часу. Цей алгоритм "дивиться лише один раз" на зображення в тому сенсі, що для прогнозування йому потрібно лише один прохід вперед, що проходить через мережу. Після видалення немаксимальних об’єктів (далі - Non-max Suppression), він потім виводить розпізнані об'єкти разом з обмежувальними рамками.

Якщо існує 80 класів, які ми хочемо, щоб детектор об’єктів розпізнав, ми можемо представити мітку класу як ціле число від 1 до 80, або як 80-мірний вектор (із 80 числами), один компонент якого 1, а решта з яких 0.

Я зібрав декілька зображень у папку та позначив їх, намалювавши обмежувальні рамки навколо кожного знайденого автомобіля. Ось приклад того, як виглядають мої обмежувальні рамки:

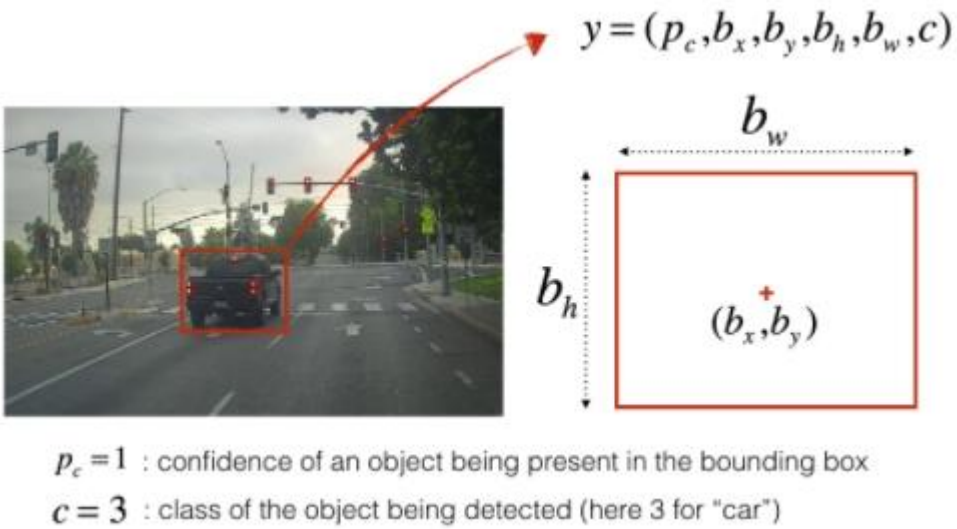

Рисунок 1. Означення рамки.

- **Вхідні дані (input)** - це пакет зображень, і кожне зображення має форму (m, 608, 608, 3)

- **Вихідні дані (output)** - це список обмежуючих рамок разом із розпізнаними класами. Кожна обмежувальна рамка представлена 6 числами $(p_c, b_x, b_y, b_h, b_w, c)$як пояснено вище. Якщо ми розширили c у 80-мірний вектор, кожен обмежувальний квадрат потім представляється 85 числами.
- **Якірні рамки (anchor boxes)** вибираються шляхом вивчення навчальних даних, щоб вибрати розумні співвідношення висоти / ширини, які представляють різні класи. Я вибрав 5 якірних рамок.
- Архітектура YOLO: ЗОБРАЖЕННЯ (m, 608, 608, 3) -> DEEP CNN -> КОДУВАННЯ (m, 19, 19, 5, 85).

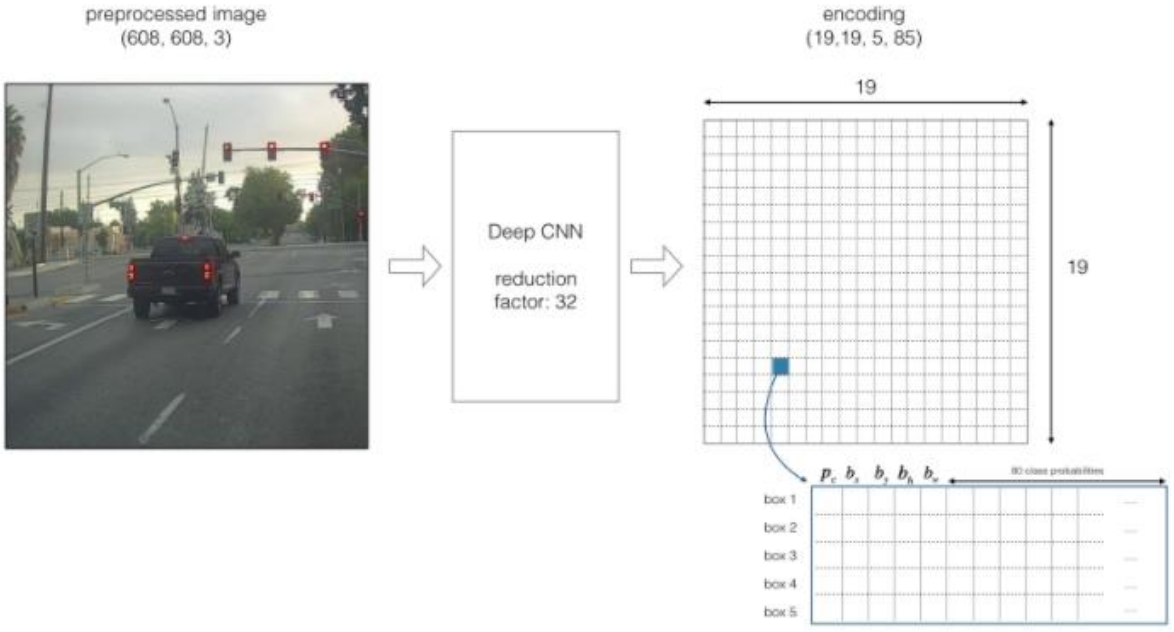

Рисунок 2. Архітектура кодування для YOLO.

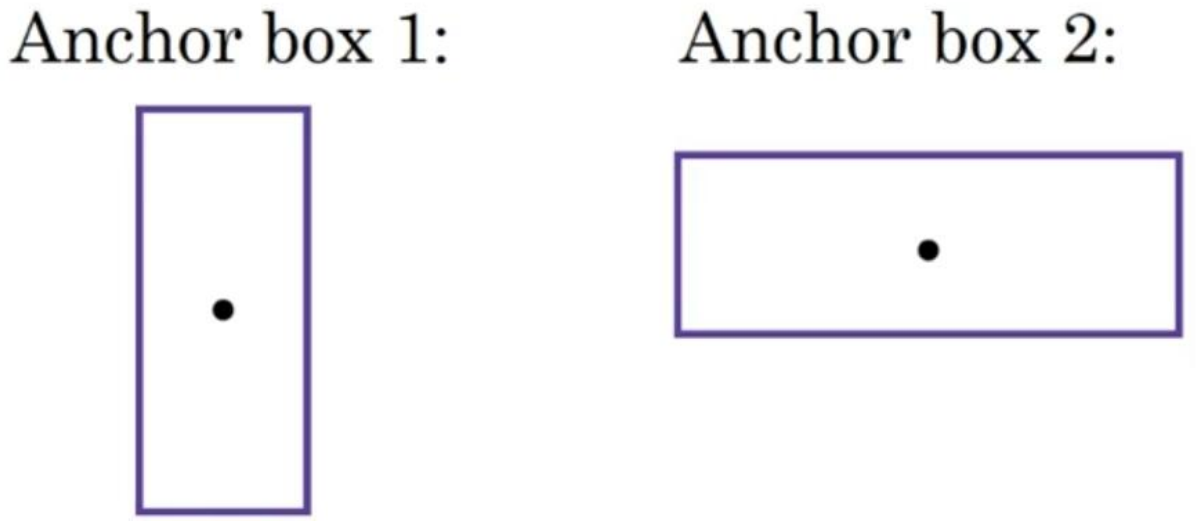

Рисунок 3. Приклади двох різних якірних рамок.

Оскільки ми використовуємо 5 якірних рамок, кожна з 19 комірок x19 таким чином кодує інформацію про 5 рамок. Якірні рамки визначаються лише їх шириною та висотою. Для простоти ми згладжуємо два останні виміри кодування фігури (19, 19, 5, 85), тому вихід Глибокої ЗНМ є (19, 19, 425).

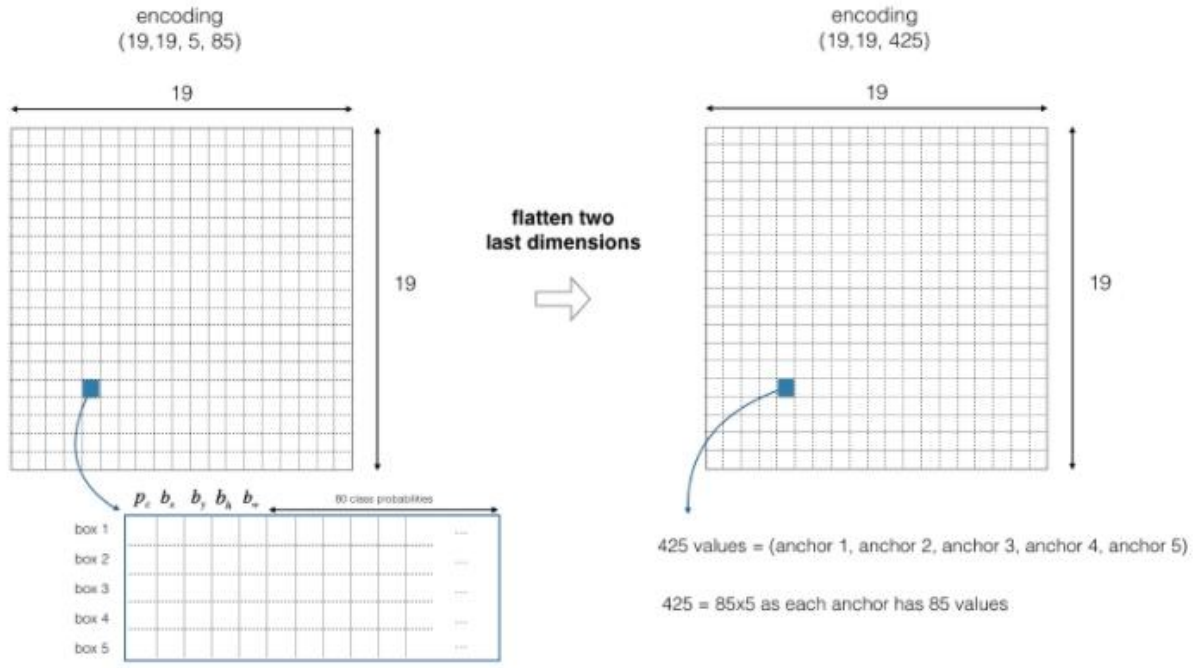

Рисунок 4. Вирівнювання останніх двох вимірів.

Тепер для кожної рамки (кожної комірки) ми обчислюємо наступний елементний продукт і витягуємо ймовірність того, що рамка містить певний клас.

Оцінка в класі становить $score_{c,i} = p_c \times c_i$: ймовірність наявності об'єкта p_c множимо на ймовірність того, що об'єкт є певним класом c_i.

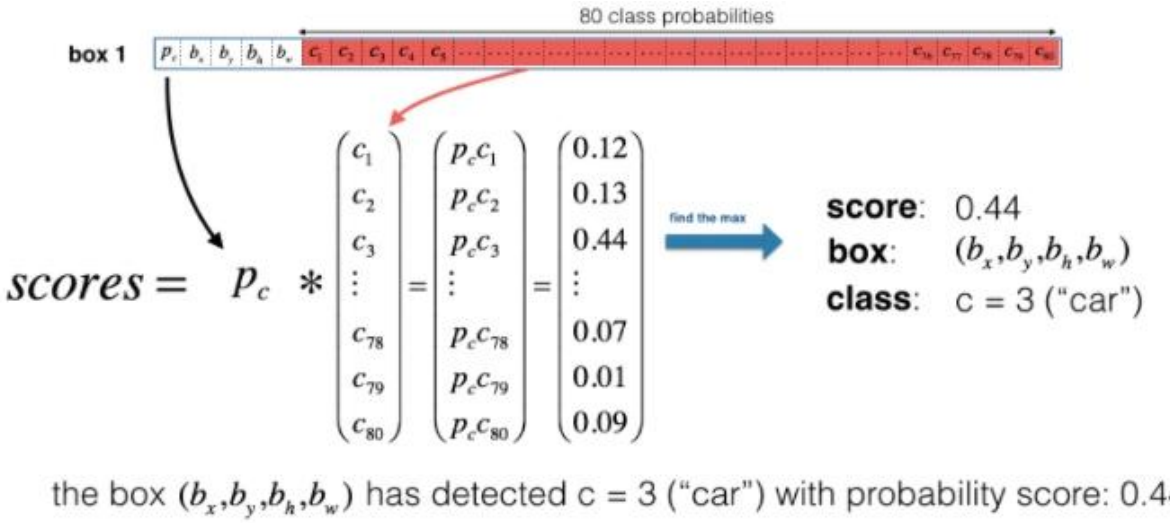

Рисунок 5. Пошук класу, виявленого кожною рамкою.

Ми можемо візуалізувати вихідні дані YOLO, побудувавши графіки обмежувальних рамок. Це призводить до такої візуалізації:

Рисунок 6. Кожна комірка дає нам 5 коробок. Загалом модель передбачає: 19x19x5 = 1805 коробок, просто подивившись один раз на

зображення (один прохід вперед через мережу)! Різні кольори позначають різні класи.

На малюнку вище лише побудовані ящики - це ті, для яких модель призначила високу ймовірність, але це все ще занадто багато ящиків. Ми хотіли б зменшити результати роботи алгоритму до значно меншої кількості виявлених об'єктів.

Для цього ми використовуємо “non-max suppression”. Зокрема, ми виконуємо такі кроки:

- Позбудемося рамок з низьким балом. Це означає, що поле не надто впевнено визначає клас або через низьку ймовірність будь-якого об’єкта, або через низьку ймовірність цього конкретного класу.
- Виберемо лише одну рамку, коли декілька вікон перекриваються між собою і виявляють один і той же об’єкт.

Рисунок 7. У цьому прикладі модель передбачила 3 машини, але насправді це 3 передбачення того самого автомобіля. Запуск “non-max suppression” вибере лише найбільш точний (з найвищою вірогідністю) з 3 полів.

“Non-max suppression” використовує дуже важливу функцію, яка називається "Перетин над Об’єднанням", ("Intersection over Union" або IoU).

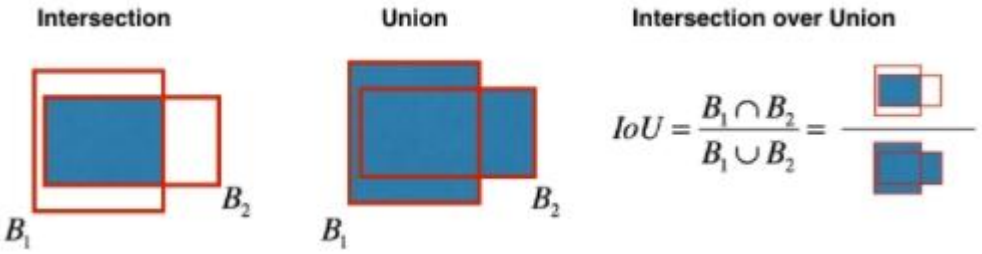

Рисунок 8. Означення "Перетину над Об’єднанням".

Код програми наведено в додатку А.

Навчання моделі YOLO займає дуже багато часу і вимагає досить великого набору позначених обмежувальних рамок для великого діапазону цільових класів.

Я використовував існуючу попередньо навчену модель Keras YOLO. Ці ваги були взяті з офіційного веб-сайту YOLO [9] і були перетворені за допомогою функції, написаної Алланом Зеленером [8].

Для тестування я використав кілька зображень, як на рисунках 9 та 10.

Рисунок 9. Тестове зображення А.

Рисунок 10. Тестове зображення Б.

В результаті тестування я отримав вихідні зображення, як на рисунках 11 і 12.

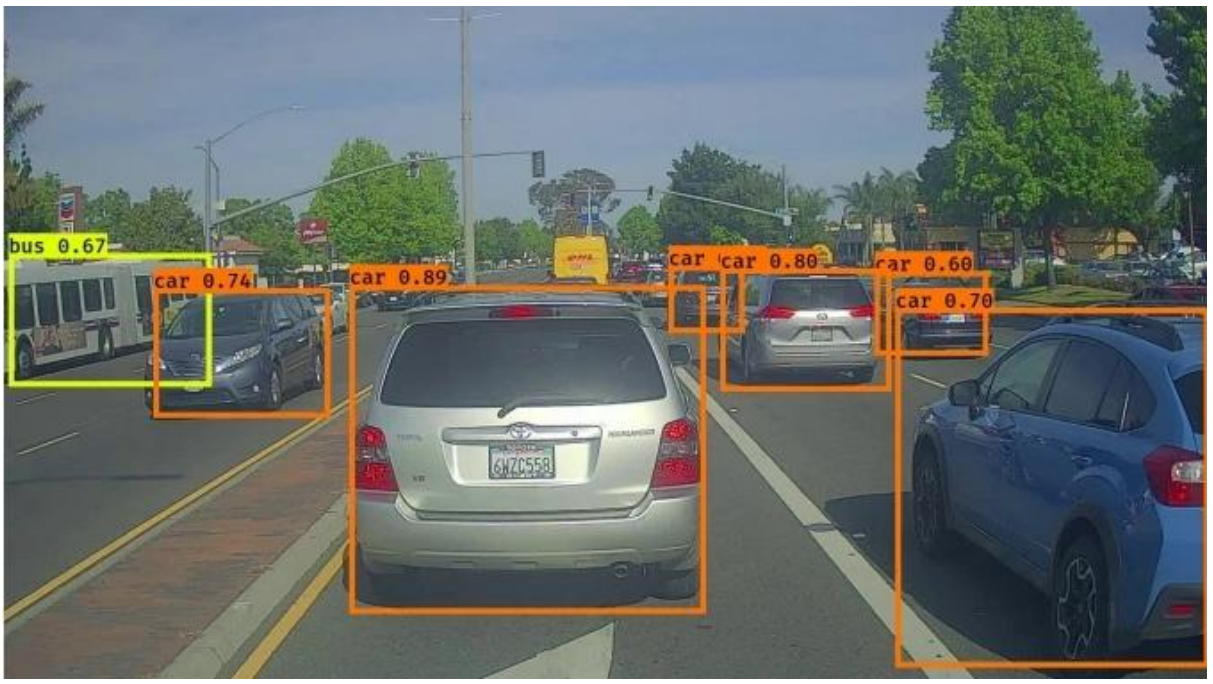

Рисунок 11. Вихідне зображення А.

Рисунок 12. Вихідне зображення Б.

Звичайно, на зображеннях є інші автомобілі, але програма знайшла найбільш очевидні з показником IoU 0,6-0,89, що є хорошим результатом.

3.3 Порівняльний аналіз

Порівняння мого програмного забезпечення з розглянутими аналогами: програма з моделлю YOLO може виявити автомобіль, визначити його локалізацію, як це роблять моделі класифікації та класифікації з локалізацією. Але наша програма може зробити це для кількох автомобілів, а не лише для одного. Крім того, модель YOLO набагато швидша і точніша, ніж модель Sliding Window.

Висновки

В результаті виконання дипломної роботи розглянуто застосування штучних нейронних мереж до задачі класифікації та методи навчання нейронної мережі, проаналізовано існуючі підходи до вирішення задачі розпізнавання автомобілів.

Програмний продукт, здатний виявляти автомобілі, був розроблений за допомогою нейронної мережі, яка використовує метод YOLO для їх виявлення. Досягнута хороша точність виявлення автомобілів, особливо тих, які розташовані ближче до переднього фону і без перешкод.

Даний програмний засіб реалізовано для розпізнавання автомобілів, проте він є універсальним - за його допомогою можливо розпізнавати будь-які об’єкти.

Даний програмний засіб може застосовуватись у машинах без водія, у камерах спостереження, у каптчах, а також у більш складних програмних системах, яким потрібно розпізнавати автомобілі.

Для розробленого продукту планується розширити навчальну вибірку, покращити точність розпізнавання та впровадити класифікацію автомобілів за їхніми моделями.

Перелік джерел посилання

1. Wasserman, Philip D. (1993). Advanced methods in neural computing. Van Nostrand Reinhold.
2. Haykin, Simon S. (1999). Neural networks : a comprehensive foundation. Prentice Hall.
3. Fahlman, S.; Lebiere, C (1991). The Cascade-Correlation Learning Architecture.created for National Science Foundation.
4. Хайкин С. Нейронные сети, полный курс [Текст] / Саймон Хайкин. – 2-е изд., перед. – М. : Вильямс, 2008. - 1103 с.
5. Jinru L. Fault diagnosis of piston compressor based on Wavelet Neural Network and Genetic Algorithm [Текст] / Jinru L., Yibing L., Keguo Y.: Proceedings of the 7th World Congress on Intelligent Control and Automation. – 2008.
6. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi - You Only Look Once: Unified, Real-Time Object Detection (2015).
7. Joseph Redmon, Ali Farhadi - YOLO9000: Better, Faster, Stronger (2016).
8. Allan Zelener - YAD2K: Yet Another Darknet 2 Keras.
9. The official YOLO website (https://pjreddie.com/darknet/yolo/)

ДОДАТОК А

Код програми

Комірка 1:

```
import pandas as pd
import scipy.io
import tensorflow as tf
from matplotlib.pyplot import imshow
from yad2k.models.keras_yolo import yolo_loss, yolo_boxes_to_corners, yolo_body, yolo_head, preprocess_true_boxes
from yolo_utils import generate_colors, draw_boxes, preprocess_image, read_anchors, read_classes, scale_boxes
import os
import argparse
import matplotlib.pyplot as plt
from keras.layers import Conv2D, Input, Lambda
import scipy.misc
import PIL
import numpy as np
from keras.models import Model, load_model
from keras import backend as K

%matplotlib inline
```

Комірка 2:

```
def yolo_boxes_filter(box_class_probs, box_confidence, boxes, threshold = .6):
    """This function filters YOLO boxes using thresholding on object and class confidence.

    Arguments:
    box_confidence -- tensor of shape (19, 19, 5, 1)
    boxes -- tensor of shape (19, 19, 5, 4)
    box_class_probs -- tensor of shape (19, 19, 5, 80)
    threshold -- real value, if [ highest class probability score < threshold], then get rid of the corresponding box

    Returns:
    scores -- tensor of shape (None,), containing the class probability score for selected boxes
    boxes -- tensor of shape (None, 4), containing (b_x, b_y, b_h, b_w) coordinates of selected boxes
    classes -- tensor of shape (None,), containing the index of the class detected by the selected boxes
```

```
    Note: "None" is here because you don't know the exact number of selected boxes, as it depends on the threshold.
    For example, the actual output size of scores would be (10,) if there are 10 boxes.
    """

    box_class_scores = K.max(box_scores, axis=-1)
    box_scores = box_class_probs * box_confidence;
    box_classes = K.argmax(box_scores, axis=-1)
    filtering_mask = box_class_scores >= threshold
    classes = tf.boolean_mask(box_classes, filtering_mask)
    boxes = tf.boolean_mask(boxes, filtering_mask)
    scores = tf.boolean_mask(box_class_scores, filtering_mask)

    return classes, boxes, scores,
```

Комірка 3:

```
with tf.Session() as test_a:
    box_class_probs = tf.random_normal([19, 19, 5, 80], mean=1, stddev=4, seed = 1)
    box_confidence = tf.random_normal([19, 19, 5, 1], mean=1, stddev=4, seed = 1)
    boxes = tf.random_normal([19, 19, 5, 4], mean=1, stddev=4, seed = 1)
    classes, scores, boxes = yolo_boxes_filter(box_class_probs, box_confidence, boxes, threshold = 0.5)
    print("classes[2] = " + str(classes[2].eval()))
    print("boxes[2] = " + str(boxes[2].eval()))
    print("scores[2] = " + str(scores[2].eval()))
    print("classes.shape = " + str(classes.shape))
    print("boxes.shape = " + str(boxes.shape))
    print("scores.shape = " + str(scores.shape))
```

Комірка 4:

```
def iou(box1, box2):
    """This function implements the intersection over union (IoU) between box1 and box2

    Here are arguments:
    box1 -- first box - a list object with coordinates (x1, y1, x2, y2)
    box2 -- second box - a list object with coordinates (x1, y1, x2, y2)
    """

    xj1 = max(box1[0], box2[0])
    yj1 = max(box1[1], box2[1])
    xj2 = min(box1[2], box2[2])
```

```
    yj2 = min(box1[3], box2[3])
    area_intersect = max(xj2-xj1, 0) * max(yj2-yj1, 0)
    area_box1 = (box1[2]-box1[0])*(box1[3]-box1[1])
    area_box2 = (box2[2]-box2[0])*(box2[3]-box2[1])
    area_union = area_box1 + area_box2 - area_intersect
    iou =area_intersect / area_union

    return iou
```

Комірка 5:

```
box1 = (2, 1, 4, 3)
box2 = (1, 2, 3, 4)
print("iou = " + str(iou(box1, box2)))
```

Комірка 6:

```
def yolo_nmax_suppression(classes,  boxes, scores, boxes_max = 10, iou_threshold = 0.5):
    """
    This function applies Non-max suppression (NMS) to set of boxes

    Here are arguments:
    classes -- tensor of shape (None,), output of yolo_boxes_filter()
     boxes -- tensor of shape (None, 4), output of yolo_boxes_filter() that have been scaled to the image size (see later)
    scores -- tensor of shape (None,), output of yolo_boxes_filter()
    boxes_max -- integer, the maximum number of your predicted boxes you'd like
    iou_threshold -- real value, the "intersection over union" (IoU) threshold used for NMS filtering

    Returns:
    classes -- tensor of shape (, None), predicted class for each box
    boxes -- tensor of shape (4, None), predicted box coordinates
    scores -- tensor of shape (, None), predicted score for each box

     Note: Here the "None" dimension of your output tensors has obviously to be less than max_boxes. Also note that this function will transpose the shapes of classes, scores, boxes. This is made for convenience.
    """

    tensor_max_boxes = K.variable(max_boxes, dtype='int32')
    K.get_session().run(tf.variables_initializer([tensor_max_boxes]))
nms_indices = tf.image.non_max_suppression(scores=scores, boxes=boxes, max_output_size=boxes_max, iou_threshold=iou_threshold)
```

```
    classes = K.gather(classes, nms_indices)
    boxes = K.gather(boxes, nms_indices)
    scores = K.gather(scores, nms_indices)

    return classes, boxes, scores
```

Комірка 7:

```
with tf.Session() as test_b:
    classes = tf.random_normal([54,], mean=1, stddev=4, seed = 1)
    boxes = tf.random_normal([54, 4], mean=1, stddev=4, seed = 1)
    scores = tf.random_normal([54,], mean=1, stddev=4, seed = 1)
    classes, scores, boxes= yolo_nmax_suppression(classes, scores, boxes)
    print("classes[2] = " + str(classes[2].eval()))
    print("boxes[2] = " + str(boxes[2].eval()))
    print("scores[2] = " + str(scores[2].eval()))
    print("classes.shape = " + str(classes.eval().shape))
    print("boxes.shape = " + str(boxes.eval().shape))
    print("scores.shape = " + str(scores.eval().shape))
```

Комірка 8:

```
def yolo_evaluation(yolo_outputs, image_shape = (720., 1280.), boxes_max=10, score_threshold=.6, iou_threshold=.5):
    """
    This function converts the output of your YOLO encoding (a lot of boxes) to your predicted boxes along with their classes, scores and box coordinates.

    Arguments:
    yolo_outputs -- output of the encoding model (for image_shape of (608, 608, 3)), contains 4 tensors:
                    box_confidence: tensor of shape (None, 19, 19, 5, 1)
                    box_xy: tensor of shape (None, 19, 19, 5, 2)
                    box_wh: tensor of shape (None, 19, 19, 5, 2)
                    box_class_probs: tensor of shape (None, 19, 19, 5, 80)
    image_shape -- tensor of shape (2,) containing the input shape, in this notebook we use (608., 608.) (has to be float32 dtype)
    boxes_max -- integer, the maximum number of your predicted boxes you'd like
    score_threshold -- real value, if the condition [ highest class probability score < threshold] is true, then get rid of the corresponding box
    iou_threshold -- real value, the "intersection over union" (IoU) threshold used for NMS filtering

    Returns:
```

```
    classes -- tensor of shape (None,), predicted class for each box
    boxes -- tensor of shape (None, 4), predicted box coordinates
    scores -- tensor of shape (None, ), predicted score for each box
    """

    box_confidence, box_xy, box_wh, box_class_probs = yolo_outputs
    boxes = yolo_boxes_to_corners(box_xy, box_wh)
    classes, boxes, scores = yolo_boxes_filter(box_confidence, boxes, box_class_probs, score_threshold)
    boxes = scale_boxes(boxes, image_shape)
    classes, boxes, scores = yolo_nmax_suppression(classes, boxes, scores, boxes_max, iou_threshold)

    return classes, boxes, scores
```

Комірка 9:

```
with tf.Session() as test_b:
    yolo_outputs = (tf.random_normal([19, 19, 5, 1], mean=1, stddev=4, seed = 1),
                    tf.random_normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1),
                    tf.random_normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1),
                    tf.random_normal([19, 19, 5, 80], mean=1, stddev=4, seed = 1))
    classes, boxes, scores = yolo_evaluation(yolo_outputs)
    print("classes[2] = " + str(classes[2].eval()))
    print("boxes[2] = " + str(boxes[2].eval()))
    print("scores[2] = " + str(scores[2].eval()))
    print("classes.shape = " + str(classes.eval().shape))
    print("boxes.shape = " + str(boxes.eval().shape))
    print("scores.shape = " + str(scores.eval().shape))
```

Комірка 10:

```
session = K.get_session()
yolo_model = load_model("model_data/yolo.h5")
image_shape = (720., 1280.)
anchors = read_anchors("model_data/yolo_anchors.txt")
class_names = read_classes("model_data/coco_classes.txt")
```

Комірка 11:

```
yolo_model.summary()
```

Комірка 12:

```
yolo_out = yolo_head(yolo_model.output, anchors, len(class_names))
classes, boxes, scores = yolo_evaluation(yolo_out, image_shape)
```

Комірка 13:

```
def prediction(session, image_file):
    """
    This function runs the graph stored in "session" to predict boxes for an "image_file". Prints and plots the preditions.

    Here are arguments:
    session -- your tensorflow/Keras session that contains the YOLO graph
    image_file -- name of an image that is stored in the  folder "images".

    Returns:
    out_scores -- tensor of shape (None, ), scores of the predicted boxes
    out_boxes -- tensor of shape (None, 4), coordinates of the predicted boxes
    out_classes -- tensor of shape (None, ), class index of the predicted boxes

    Note: Here "None" actually represents the number of predicted boxes -  it varies between 0 and boxes_max.
    """

    image, image_data = preprocess_image("images/" + image_file, model_image_size = (608, 608))
    colors = generate_colors(class_names)
    out_classes, out_boxes, out_scores = sess.run([classes, boxes, scores], feed_dict={yolo_model.input: image_data, K.learning_phase(): 0})
    print('Found {} boxes for {}'.format(len(out_boxes), image_file))
    draw_boxes(image, out_classes, out_boxes, out_scores, class_names, colors)
    output_image = scipy.misc.imread(os.path.join("out", image_file))
    image.save(os.path.join("out", image_file), quality=90)
    imshow(output_image)

    return out_classes, out_boxes, out_scores
```

Комірка 14:

```
out_classes, out_boxes, out_scores = predict(session, "test.jpg")
```

ДОДАТОК Б

Результати виконання коду

Комірка 3:

classes[2] = 7

boxes[2] = [8.42653275 3.27136683 -0.5313437 -4.94137383]

scores[2] = 10.7506

classes.shape = (?,)

boxes.shape = (?, 4)

scores.shape = (?,)

Комірка 5:

iou = 0.14285714285714285

Комірка 7:

classes[2] = -2.24527

boxes[2] = [-5.299932   3.13798141 4.45036697 0.95942086]

scores[2] = 6.9384

classes.shape = (10,)

boxes.shape = (10, 4)

scores.shape = (10,)

Комірка 9:

classes[2] = 54

boxes[2] = [1292.32971191 -278.52166748 3876.98925781 -835.56494141]

scores[2] = 138.791

classes.shape = (10,)

boxes.shape = (10, 4)

scores.shape = (10,)

Комірка 11:

Total params: 50,983,561

Non-trainable params: 20,672

Trainable params: 50,962,889

Комірка 14:

Found 7 boxes for test.jpg

bus 0.67 (5, 266) (220, 407)

car 0.80 (761, 282) (942, 412)

car 0.74 (159, 303) (346, 440)

car 0.70 (947, 324) (1280, 705)

car 0.60 (925, 285) (1045, 374)

car 0.89 (367, 300) (745, 648)

car 0.66 (706, 279) (786, 350)