МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.

«___» _____2021 р.

ДИПЛОМНА ПРОЕКТ

(ПОЯСНЮВАЛЬНА З*АРІ*СКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Програмний модуль класифікації даних відеохостингу *YouTube*

Виконавець: Павленко В.С.

Керівник: Нечипорук О.П.

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Освітнього ступеня бакалавр

Спеціальність 123 "Комп'ютерна інженерія"
(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Литвиненко О. Є.

«_____»_____2021 р.

ЗАВДАННЯ
на виконання дипломної роботи (проекту)

Павленка Владислава Сергійовича
(прізвище, ім'я, по батькові)

1.Тема роботи: "Програмний модуль для класифікації даних відеохостингу *YouTube*"

затверджена наказом ректора від "04" лютого 2021 року № 135/ст.

2. Термін виконання роботи: з 17.05.2021 до 20.06.2021

3. Вихідні дані до роботи: постановка задачі до виконання роботи;
мова програмування *Python*, *IDE Sublime Tex*.

4.Зміст пояснювальної записки (перелік питань, що підлягають розробці):
1) аналіз принципів побудови штучних нейронних мереж;
2) проектування архітектури та алгоритму штучної нейронної мережі;
3) розробка програмного модуля класифікації даних відеохостингу *YouTube*

5.Перелік обов'язкового графічного матеріалу:
1) діаграма прецедентів
2) діаграма послідовності
3) вікно користувацького інтерфейсу
4) схума алгоритму перенесення нейронного представлення зображення

6. Календарний план

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Провести аналіз літератури за темою дипломної роботи та аналіз існуючих систем	17.05.2021	
2	Зробити вибір компонентів системи	17.05.2021-19.05.2021	
3	Розробити структуру програмних засобів системи цифрового репозитарію	20.05.2021-24.05.2021	
4	Розробити програмні засоби цифрового репозитарію	28.05.2021-31.05.2021	
5	Провести налаштування програмних засобів на сервері	31.05.2021-06.06.2021	
6	Написати пояснювальну записку	07.06.2021-14.06.2021	
7	Підготувати презентацію	9.06.2021-11.06.2021	

7. Дата видачі завдання « 17 » грудня 2021 р.

Керівник дипломного проекту _____ д.т.н., доц. Нечипорук О.П.
(підпис)

Завдання прийняв до виконання _____ Павленко В.С.
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту “Програмний модуль для класифікації даних відеохостингу *YouTube*”: 57 с., 19 рис., 1 таблиця , *11* літературних джерел, 2 додатки.

ШТУЧНА НЕЙРОННА МЕРЕЖА, PYTHON, КЛАСИФІКАЦІЯ, ПРОГРАМНИЙ МОДУЛЬ, ДАТАСЕТ , ДАНІ , ВІДЕОМАТЕРІАЛИ, БІБЛІОТЕКА.

Мета дослідження – розробити програмний модуль класифікації выдеохостингу *YouTube*.

Об’єкт дослідження – проведення загальних класифікаційних операцій з використанням технології нейронних мереж.

Предмет дослідження – програмний модуль класифікації відеоматеріалу за назвою.

Встановлено, що класифікація відеоматеріалів за назвою з використанням технології штучних нейронних мереж дозволяє поліпшити розподілення та пошук потрібного відеоматеріалу за класами всередині системи відеохостингу *YouTube* .

Результати дипломного проекту рекомендується використовувати при розробці системи класифікації выдеофайлів за категоріями.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

НМ – нейронна мережа

ШНМ – штучна нейронна мережа

БНМ – багатошарові нейронні мережі

NLTK – *Natural Language Toolkit* (Інтсрументарій природної мови)

ГП – графічний процесор

ТП – тензорний процесор

ВСТУП

За останні десятиліття відбувся перехід до так званого «інформатизованого суспільства». Його концепція відображає зростання темпів виробництва, розподілу, переробки та споживання інформації. Це спричинило зростання актуальності розробки та модернизації інформаційних технологій. Важливе місце серед них займають алгоритмічні та програмноапаратні системи і комплекси з елементами штучного інтелекту, призначені розв‘язувати інтелектуальні задачі та виконувати функції, які раніше вважалися прерогативою людини.

Також, значного прогресу було досягнуто біологами в області дослідження роботи мозку. Досліджуючи структуру і функції нервової системи людини, вони багато чого дізналися про функціонування мозку.

У процесі досліджень з'ясувалося, що мозок має приголомшуючу складність: мільярди нейронів, кожен з яких з'єднаний з сотнями або тисячами інших, утворюють систему, яка перевершує нинішні концепції суперкомп'ютерів.

Концепцію «штучна нейронна мережа» вперше було вжито в 40-*х* роках минулого століття. На рівні логіки, діяльність нервової системи людини і тварин було перенесено на концепцію штучної нейромережі. У 1943-му році формальна модель нейрона була розроблена, хоча в ті часи вона мала певні недоліки, та могла вирішувати невелику кількість задач. Ці недоліки вдалося виправити, об'єднуючи нейрони в мережі по декілька шарів. Такі системи виявилися більш гнучкими: об'єднані в мережу формальні нейрони можуть вирішувати завдання, які традиційно відносяться до області «людської діяльності» (наприклад прийняття рішень на основі неповної інформації і навіть розпізнавання образів). Найромережі виявились здатними запам‘ятовувати інформацію та навчатися, що нагадує розумові процеси людини. Саме тому в ранніх роботах по дослідженню нейромереж часто згадувався термін «штучний інтелект».

За останній час швидко зростав інтерес до нейронних мереж. Ними

зайнялися фахівці з різних областей — техніки, фізіології, психології. Цей інтерес зрозумілий, так як штучна нейронна мережа, по суті, являє собою модель природної нервової системи, тому створення і вивчення таких мереж дозволяє дізнатися багато про функціонування природних систем.

Сама теорія штучних нейронних мереж з'явилася в 40-х роках завдяки останнім на той момент досягнення біології, так як штучні нейрони складаються з елементів, які моделюють елементарні функції біологічних нейронів. Ці елементи організуються за способом, який може відповідати (або не відповідати) анатомії мозку.

Незважаючи на таку поверхневу подібність, штучні нейронні мережі демонструють дивовижні властивості, подібні до властивостей природного мозку. Наприклад, штучна нейронна мережа здатна змінювати свою поведінку в залежності від зовнішнього середовища. Прочитавши пред'явлені їй вхідні сигнали (можливо, разом з необхідними виходами) вона здатна «навчитися» так, щоб забезпечувати необхідну реакцію.

Варто відзначити, що нейронна мережа робить узагальнення автоматично завдяки своїй структурі, а не за допомогою спеціально написаних програм.

Іншою цікавою властивістю нейромереж є надійність: навіть якщо кілька елементів працюватимуть неправильно або вийдуть з ладу, то мережа все одно буде здатна видавати правильні результати, але з меншою точністю.

Штучні нейронні мережі застосовуються у різноманітних сферах, зв‘язаних з обробкою інформації. Наприклад в таких галузях як: розпізнавання образів, системи пам‘яті, що основані на асоціаціях, класифікація, компресія даних, задачі оптимізації, теорія керування, вирішення інженерних задач проектування, екстраполяція та прогнозування. Таким чином, тренування нейронних мереж для задачі класифікації, яка є однією з основних є актуальною темою .

Метою написання роботи є вивчення системи роботи та побудови нейромережі класифікації.

При написанні роботи були поставлені наступні завдання :

1. Ознайомитись з поняттям сучасних нейронних мереж , їх будови та

принципу роботи;

2. Вивчити способи реалізації нейромережі класифікації;
3. Розробити програмний модуль для класифікації тексту за категоріями.

Мета дослідження – розробити програмний модуль класифікації выдеохостингу *YouTube*.

Об’єкт дослідження – проведення загальних класифікаційних операцій з використанням технології нейронних мереж.

Предмет дослідження – програмний модуль класифікації відеоматеріалу за назвою.

РОЗДІЛ 1
АНАЛІЗ ОСОБЛИВОСТЕЙ ПРЕДСТАВЛЕННЯ ТА ВИКОРИСТАННЯ ВІДЕОКОНТЕНТУ

1.1. Можливості використання відеоконтенту в розповсюдженні інформації

У наше століття прогресивних технологій відео є одним з ефективних способів передачі інформації, залучення клієнтів. Сьогодні ми просто не уявляємо життя без відео. Взяти хоча б новини, фільми, телепередачі, яку дивиться кожен з нас, або рекламні ролики, які миготять між передачами або в процесі перегляду прем'єр. Таким чином, відео міцно увійшов в наше життя. На сьогоднішній день різновидів відео дуже багато. Давайте дізнаємося про них. Також з'ясуємо який відеоконтент потрібен для досягнення ваших цілей в бізнесі.

Незважаючи на різноманіття і розрізнену структуру сучасного ринку відеоматеріалів, експерти виділяють п'ять основних категорій відеоконтенту.

Сучасний ринок відеовиробництва вражає своїми масштабами і різноманітністю доступного користувачам контенту. Незважаючи на це, сьогодні не існує єдиної структури ринку і часто розібратися в чинній класифікації роликів, кліпів, документальних і художніх фільмах буває дуже складно. Пропонуємо розібратися в даному питанні і почати наше обговорення з термінології.

- Відеоролик для продажів. Що розуміється під цим терміном - більшість учасників ринку вважають, що цей відеоконтент в першу чергу покликаний привернути увагу глядачів і розповісти про який-небудь товар або послугу.

Безперечно, що продає відео має привертати увагу потенційних клієнтів, але в першу чергу від нього вимагається продати ідею, донести її до кінцевого споживача. І зовсім неважливо йдеться про брендових годинниках або перевиборній агітації. Споживач повинен почути і спалахнути ідей, тільки в цьому випадку ролик може вважатися продають.

• Відео - контент. По суті, це будь-яке відео від музичних кліпів та телевізійної реклами до документальних фільмів та графічних заставок.

види відеоконтенту

Умовно весь відеоконтент можна розділити на кілька категорій:

презентаційні ролики

В наші дні саме презентаційні відеоролики є найбільш популярним типом відео. Їх завдання полягає в візуальної презентації продукту, послуги, явища і т.д. кінцевого споживача. Саме до цієї категорії відноситься класична телевізійна реклама, яка розповідає про переваги товару.

іміджеві ролики

Імеджевих відеоконтент покликаний не просто продати ідею, а породити в свідомості глядача певний емоційний відгук. За допомогою контенту цієї категорії створюється імідж компанії, товару, людини. Яскравим прикладом імедвежевих відеороликів є відео про виставку самураїв. Ролик не рекламує сам захід, однак при цьому він настільки реалістично занурює глядача в атмосферу виставки, завдяки чому в свідомості потенційних відвідувачів формується до неї позитивне ставлення. Не менш цікавим прикладом є відеоролик художниці Олександри П'ятницькій. Його атмосфера і сюжет створюють у свідомості глядачів образ героїні, що природно провокує людини на пошук інформації про роботи художниці і бажання побачити їх не тільки в інтернеті, але і наживо.

Навчальні ролики

Цілком очевидно, що головним завданням, яке ставить перед собою навчальний відеоконтент - навчити свого глядача виконання певних процедур і дій, поведінки в різних умовах і т.д. Сучасні навчальні фільми можуть бути не тільки постановочними, а й графічними, тобто виконані за допомогою засобів комп'ютерної графіки.

вірусні ролики

Вірусний відеоконтент в першу чергу покликаний привернути увагу користувачів і при цьому все одно, позитивне або негативне настрій вони створюють. Вірусний ролик «чіпляє» глядача, заражає його певної ідей, внаслідок чого у користувача з'являється потреба передати «вірус» далі.

Тобто якщо вірусний ролик вдався, то після його перегляду у глядачів обов'язково має з'явитися бажання поділитися ним з іншими користувачами, наприклад, відправивши посилання одному або виклавши відео на своїй сторінці в соціальних мережах.

Деякі рекламні агенції вважають, що вірусний ролик обов'язково повинен бути епатажним, з елементами насильства або еротики. Експерти, які не беруться заперечувати цю думку, однак в одному можна бути впевненим точно - вміст відео не повинно суперечити концепції продукту або і зовсім дискредитувати його.

Що краще за все використовувати для створення вірусного ролика - професійну камеру чи мобільний телефон повинен вирішувати виключно менеджер з реклами, ніяких строгих правил в цьому відношенні немає.

соціальні ролики

Соціальний відеоконтент покликаний розповісти і візуально продемонструвати глядачам шляхи вирішення тієї чи іншої соціальної проблеми. При цьому творцям роликам необхідно лише показати наявність проблеми і заявити про себе, при цьому уникаючи відкритої реклами власних послуг.

Відео Арт

Даний відеоконтент є найбільш цікавою категорією. Найчастіше подібні відео ролики створюються не з метою заробітку, а для того щоб заявити про себе і свою творчість. У цю категорію відносяться так звані фестивальні фільми, тобто картини, які створювалися не для масового прокату на великих екранах, а як один із напрямів сучасного кіномистецтва та спосіб самовираження для невизнаних геніїв.

Необхідно відзначити, що сьогодні досить рідко можна зустріти відеоконтент, який можна було б віднести лише до однієї категорії. Однак, при

цьому перед кожним роликом варто своя задача, а все інше це тільки способи і інструменти для її вирішення.

9 способів просування відео контенту

Якщо вірити маркетинговим дослідженням, цільові сторінки з відео контентом отримують високий відсоток конверсій. До 80% відвідувачів, які потрапляють на такі сторінки, перетворюються в передплатників, клієнтів або взаємодіють з брендом іншим способом. Відео контент генерує конверсії - для того, щоб це твердження відносилося і до вашого бізнесу, радимо протестувати ці 9 стратегій.

1. Заклик до дії

Знайомий усім *CTA* - це старий, але вірний спосіб підвищити цінність вашого відео. Заклик до дії може бути видно завжди під час показу ролика або розміщений в будь-якій частині - наприклад, на початку або в кінці відео або будь-який стратегічній частині.

Запам'ятайте, що кращі заклики - короткі і мотивують. І дійте за цим принципом.

2. Соціальні мережі

По тому, скільки відео форматів сьогодні використовують соціальні мережі, легко зрозуміти, як важливо використовувати їх для просування роликів. Це і *Instagram Stories*, і *Instagram TV*, і *Live* формати -в різних соціальних мережах. Все це є цінним активом в вашому арсеналі.

3. Оптимізація для *Google*

Оптимізувати відео контент для *Google* теж потрібно, також як це роблять з текстами. Тому, придумуйте привабливий заголовок, включаючи ключові слова, а також пишіть відповідний текст опису.

4. Заставка

Звичайно, судити про якість відео по заставці не дуже правильно, але факт залишається фактом - «зустрічають по одягу». Існують способи створити ескізи, колажі, заставки, які будуть відображати ваш фірмовий стиль. Не забудьте додати свій логотип або ім'я бренду, а також зробити текстові тизери - про що

буде відео.

5. Використання популярних сторінок на сайті, щоб просувати відео контент

Це можна вважати недооціненою стратегією, хоча вона може значно збільшити успіх вашого відео маркетингу. Принцип простий - використовуйте аналітичні сервіси, щоб з'ясувати відвідувані сторінки і розділи, і публікуйте там ролики або додайте анонси та посилання на відео платформи.

6. Кнопка «Поділитися»

Просування відео може бути легше, якщо використовувати елементарні способи - наприклад, додати кнопку «Поділитися». Це дозволить глядачам розміщувати ваш контент на своїх сторінках в соцмережах, збільшить охоплення. І, можливо, дозволить трохи заощадити на просуванні.

7. Навчальний відео

Презентуйте свій бренд як експерта в сфері - це може допомогти охопити більше глядачів.

8. Відео контент в розсилці

Електронні листи, що містять відео, переглядають частіше. І, що більш важливо, це збільшує кількість переходів по посиланнях і зменшує кількість відписок.

9. Оптимізація сторінки з відео

Тема, метатеги, опису - повинні бути зроблені таким чином, щоб залучити людей на вашу сторінку через пошук. Але при цьому бути «людськими», тобто зрозумілими і написаними простою мовою.

Поради, якими ми поділилися в цій статті - прості, але це дієві способи максимізувати ефективність вашої роботи з відео контентом. Але не забувайте про здатність створити класний креатив, адже в підсумку саме в змісті відео полягає основний успіх.

1.2. Можливості масштабної класифікації заображень відеоконтенту

Дана бітова матриця, яка містить закрашене зображення кола, квадрата або треугольніка.Ізображеніе може бути трохи перекручене і може містити помехі.Необходімо написати алгоритм для визначення типу намальованої фігури по матриці.

Ця проста на перший погляд завдання зустрілася мені на вступному іспиті в *DM Labs*.На першому занятті ми обговорили рішення, а викладач (Олександр Шоломів; він керував і подальшою реалізацією) показав, чому для вирішення краще використовувати машинне навчання.

В процесі дискусії ми виявили, що наше рішення проводиться в два етапи. Перший етап - фільтрація перешкод, другий етап - обчислення метрики, по якій буде проходити класифікація. Тут виникає проблема визначення меж: необхідно знати, які значення може приймати метрика для кожної фігури. Можна прокласти ці кордони вручну "на око", але краще доручити цю справу математично обґрунтованого алгорітму.Ета навчальна задача стала для мене введенням в *Machine Learning*, і я хотів би поділитися з вами цим досвідом.

Спочатку ми створюємо безліч випадкових зображень кіл, трикутників і чотирикутників. Далі пропускаємо картинки через фільтри для ліквідації перешкод і обчислюємо набір метрик, які будуть використані для класифікації фігур. Граничні значення для метрик визначимо за допомогою дерева рішень (*Decision tree*), а точніше за допомогою його різновиди *CART* (докладніше можна прочитатитут). Реалізація алгоритму проста і краще інтерпретується в термінах границ.Пропустів невідоме зображення через фільтри і обчисливши метрики, по дереву рішень ми зможемо сказати, до якого класу належить фігура на зображенні.

генерація

Генератор повинен не тільки створювати картинки, а й вміти накладати перешкоди на них. При генерації ми повинні відкидати фігури з сторонами менше якоїсь величини (наприклад, 15 пікселів) і з тупими кутами (більше 150 градусів) - навіть людині складно їх класіфіціровать.Скріпт дбайливо надав Віктор Євстратов.

bitbucket.org/rampeer/image-classifier/src/HEAD/picture_generator.py

фільтри

В процесі дискусії з колегами ми знайшли хороші ідеї для фільтрів. Приймемо пікселі чорного кольору, оточені пікселями протилежного кольору за перешкоди (ця ідея лягла в основу медіанного фільтра), а за шуканий контур приймемо велике скупчення чорних пікселів, тобто використовуємо бікомпонентних фільтр.

медіанний фільтр: Для кожного чорного пікселя визначаємо кількість «чорних» сусідів в околиці радіуса *R*. Якщо це число менше нікого *T*, то відкидаємо цей піксель, тобто зафарбовує його в білий колір. У «оригінальному» медіанному фільтрі ми відкидаємо піксель, якщо у нього менше половини чорних сусідів, а ми визначили свій поріг *T*, так що, насправді - це квантільний фільтр.

Я написав медіанний фільтр чесно і «в лоб», тому він працює заO ($WHR2$), Де W і H - розміри картинки. Викладач розповів, що можна прискорити алгоритм, використовуючи перетворення Фур'є. Дійсно, медіанний фільтр можна виразити через згортку, а згортку можна швидко обчислити за допомогою

швидкого перетворення Фурье.Получается, що обчислити матрицю з кількістю сусідів можна як

result = *FFT*-1 (*FFT* (*data*) *FFT* (*window*))

Цей алгоритм працює за*O* (*WHlog* (*W* + *H*)), Тобто набагато швидше наївною реалізації, і не залежить від розмірів вікна. Через циклічності згортки виникає артефакт на кордонах: при обробці крайніх лівих пікселів сусідами будуть вважатися крайні праві. Це можна виправити, додавши рамку з чистих пікселів по краях зображення, а можна залишити так, що я і зробив - все одно цей ефект не завдає шкоди працездатності.

bitbucket.org/rampeer/image-classifier/src/HEAD/filter_median.py

Однак, я виявив у цього фільтра нехорошу властивість: він округляють гострі кути трикутників. Через це доводиться брати радіус вікна *R* досить маленьким і проходитися по зображенню фільтром тільки кілька (*N*) разів. Хоча спочатку здавалося, що можна застосовувати медіанний фільтр до тих пір, поки він видаляє якісь пікселі.

бікомпонентних фільтр: Беремо довільний чорний піксель, призначаємо йому якесь число *Q*. Призначаємо це ж число «чорним» сусідам цього пікселя в околиці з радіусом *R* і повторимо для сусідів сусідів. Будемо повторювати до тих пір, поки не дійдемо до кордону зображення (це нагадує дію інструменту «Заливка» в *Paint*, а фарбуємо в колір *Q*). Потім збільшимо число *Q* на одиницю, виберемо черговий «незабарвлений» піксель і повторимо процесс.После виконання цього алгоритму вийде набір несопрікасающіхся островів. Ми можемо з високою вірогідністю сказати, що найбільший острів - це і є фігура, а решта острова - перешкоди.

<*Filter_bicomp.py*> На відміну від попереднього, цей фільтр не псує зображення. Він може завдати шкоди, тільки якщо фігуру перетне лінія з

перешкод шириною більше *T*, що маловероятно.У медіанного фільтра я виявив ще одну властивість, на цей раз позитивне. Він розбиває простір, заповнений перешкодами, на "острівці". Значить, застосувавши потім бікомпонентних фільтр, ми отримаємо контур з приліплених перешкодами. Після обробки бікомпонентних фільтром варто ще раз пройтися медіанного, щоб прибрати залишилися нерівності. Потім потрібно побудувати навколо решти точок опуклий контур, заповнити його і обчислювати метрики вже для нового зображення.

bitbucket.org/rampeer/image-classifier/src/HEAD/process.py

Робота фільтрів:

початкове зображення	медіанний фільтр	Медіанний, бікомпонентних	бікомзалив
			

Параметри фільтрів підбираються виходячи з розмірів зображень в вибірці і їх зашумленности. Для хитрого чотирикутника, зображеного вище:

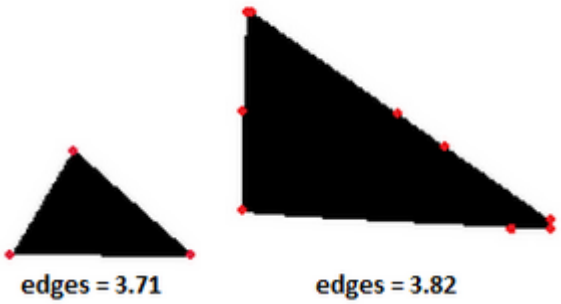
median.filter (*img*, 2, 6, 1)
bicomp.filter (*img*, 2)
median.filter (*img*, 2, 5, 2)

У нашій вибірці зображення будуть менш зашумлені, і налаштування будуть більш щадні.

метрики

В ході все тієї ж дискусії з колегами ми придумали ще багато різних цікавих метрик.

підрахунок кутів. Це найперше, що спадає на думку після прочитання завдання. Але через перешкоди можуть з'явитися додаткові кути, близькі до 0 градусів. Я безуспішно намагався боротися з цим, «склеюючи» майже Колінеарні вектора і встановлюючи пороги. Такі методи важко налаштувати, і вони все одно можуть дати некоректний результат, так як при фільтрації фігурка згладжується. Краще підсумувати квадрати синусів кутів, а якщо кути більше нікого порога *T* - округляти квадрат вгору до одиниці. Це дає досить хороший результат: гострі кути додають до лічильника одиницю, а кути, близькі до 0, майже нічого не додають. До речі, мені здалося забавним, що в такому випадку кількість кутів у трикутника може варіюватися від 2,5 до 4.

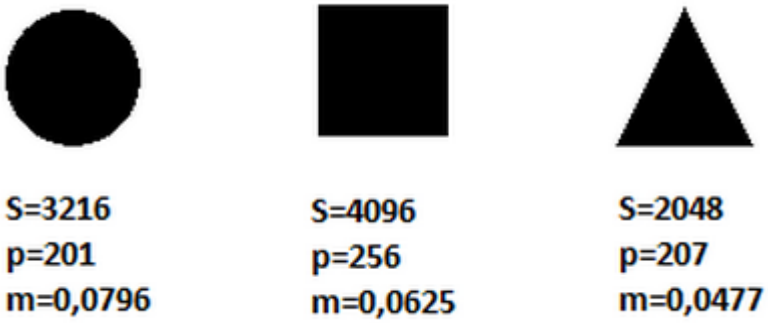


bitbucket.org/rampeer/image-classifier/src/HEAD/feature_edges.py

віддзеркалення. Сенс цієї метрики - порахувати, наскільки фігура буде збігатися з перевернутої копією себе при відображенні по горизонталі / вертикалі / уздовж обох осей. Тобто ця метрика - своєрідна міра симетрії. Коло, як не крути, завжди буде повністю збігатися сам з собою. Також, я інтуїтивно припустив, що квадрат буде більше збігатися сам з собою, ніж трикутник.

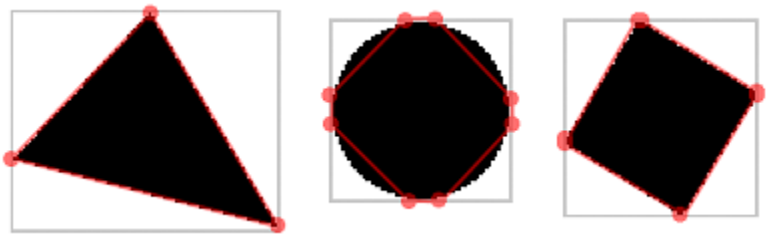
bitbucket.org/rampeer/image-classifier/src/HEAD/feature_mirror.py

Відношення площі до квадрату периметра. Периметр необхідно зводити в квадрат, щоб метрика не мала розмірності і не залежала від розміру зображення. Периметр і площа будемо брати від опуклого контуру. Коло має найбільше значення метрики серед фігур:$s / p \wedge 2 = (\pi r \wedge 2) / (4\pi \wedge 2\ r \wedge 2) = 1 / 4\pi$. Для рівностороннього трикутника (у нього це співвідношення найбільше серед трикутників):$s / p \wedge 2 = (4\sqrt{3}\ a \wedge 2) / (3 * 9a \wedge 2) = (4\sqrt{3}) / 27$. Для квадрата:$s / p \wedge 2 = (a \wedge 2) / (16 * a \wedge 2) = 1/16$.



bitbucket.org/rampeer/image-classifier/src/HEAD/feature_perimeter.py

Метрики на описує прямокутнику. Попередні метрики не дуже добре поділяли чотирьох- і трикутник, і я вирішив придумати нову метрику. Побудуємо обмежує прямокутник навколо фігури. Для кожної сторони прямокутника знайдемо перше ("мінімальне") і останнє ("максимальне") перетин з фігурою. У нас вийде "восьмикутник", для якого можна обчислювати різні метрики.



Наприклад, відношення площі фігури до площі описує квадрата (*sbound*).

bitbucket.org/rampeer/image-classifier/src/HEAD/feature_sbound.py

Також багатообіцяючою метрикою здається відношення площі фігури до площі цього восьмикутника (*sbound*2):

*bitbucket.org/rampeer/image-classifier/src/HEAD/feature_sbound*2*.py*

Збір даних

Застосувавши отримані знання, я згенерував зображення і зібрав статистику. У цей набір зображень я додав фігури, що представляють собою потенційні "погані випадки" для метрик:

рівносторонній трикутник

прямокутний трикутник з двома сторонами, паралельними осями

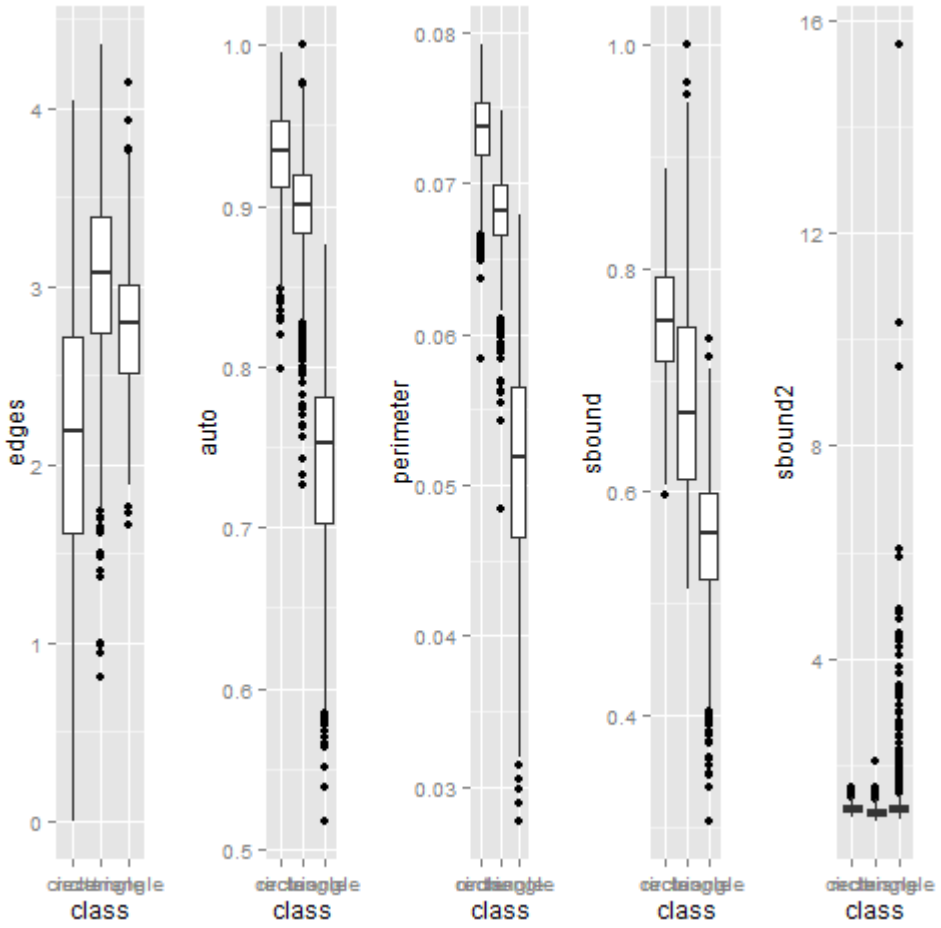
квадрат зі сторонами, паралельними осям.

Цим випадкам я дав велику вагу при побудові дерева. Справа в тому, що ймовірність випадкової генерації таких картинок мала, а ці випадки є граничними для деяких метрик. Отже, для коректної класифікації треба їх додати до вибірки з великим весом.Процедуру фільтрації зображення і збору метрик я виніс в окремий файл, вона знадобиться для аналізу. До речі, в дереві рішень наші метрики називаються "вхідними параметрами" або "фічами" (*feature*).

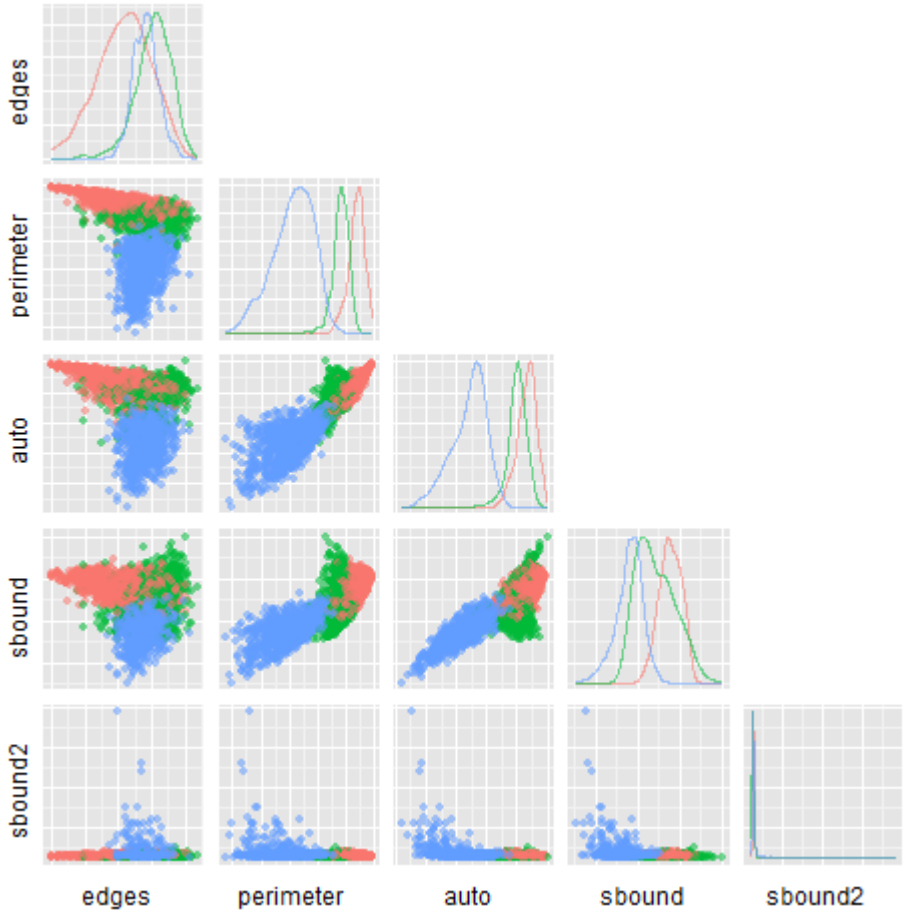
bitbucket.org/rampeer/image-classifier/src/HEAD/process.py

bitbucket.org/rampeer/image-classifier/src/HEAD/stats.py

Потім я побудував графік, щоб переконатися, що метрики досить добре розрізняють фігурки. Для цього підійде графік типу "коробки з вусиками":



"Вусики" перетинаються, а це означає, що можливі неточності при аналізі. Як було написано вище, саме для точності нам потрібні кілька метрик, а не одна. Далі я спробував переконатися, що «погані випадки» метрик не перетинаються. Для цього я побудував залежність однієї метрики від іншої. Якщо вийде, що вони монотонно залежать один від одного, то їх "погані випадки" також співпадуть.



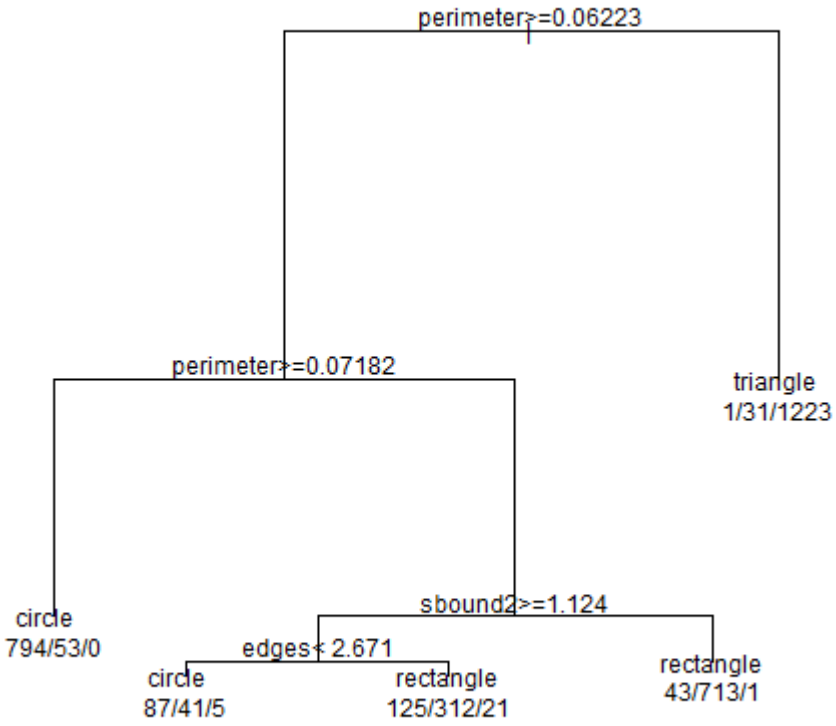
* Як ми бачимо за графіками, "хмари" точок сильно перетинаються. Отже, при класифікації можлива велика помилка. До того ж, метрики незалежні монотонно один від одного.

аналіз

За зібраними даними можна побудувати дерево рішень:

bitbucket.org/rampeer/image-classifier/src/HEAD/analyze.py

Я спробував візуалізувати дерево. Вийшло ось така схема:



З неї випливає, що деякі метрики залишилися невикористаними. Ми не могли з самого початку передбачити, які метрики виявляться "краще" других.Точность передбачення складає приблизно 91 відсоток, що непогано, враховуючи спотворення квадратів і перешкоди у вибірці:

Спробуємо намалювати зображення самостійно і проаналізувати їх:

Circle	*Rectangle*	*Triangle*

Спробуємо підвищити напряженіе.Будем спотворювати фігури доти, поки вони не стануть неправильно визначатися.

 *Rectangle*	 *Rectangle*	 *Triangle*

Ось і всё.В останньому зображенні кути трикутника сильно округлені, *edges* не може вірно працювати, а *perimeter* дає занадто велику похибку. Трикутник невдало повернуть: при побудові прямокутника тільки дві вершини будуть його стосуватися, і *sbound* і *sbound*2 не видадуть нічого розумного. Тільки *mirror* міг би видати коректний результат, але він не включений в дерево. Та й якщо з 5 метрик тільки одна вкаже на трикутник, то можна трактувати цей висновок як помилковий.

1.6. Висновки до розділу

Методи машинного навчання дозволили побудувати систему, яка добре справляється з поставленим завданням - вона досить добре розпізнає фігурки на зображенні.

РОЗДІЛ 2
ПРИНЦИПИ ГЛИБИННОГО НАВЧАННЯ СИСТЕМОЮ *TENSORFLOW*

2.1. Особливості використання системи *TensorFlow*

TensorFlow - це система машинного навчання, яка може бути використана, якщо є набір даних, за якими ця система можу бути навчена, або задача вписується в одну з відкритих для використання баз.

Відкрита за допомогою *TensorFlow* планета *Kepler*-90*i* робить систему *Kepler*-90 єдиною відомою системою з вісьмома планетами навколо однієї зірки. Поки не знайшли іншу систему восьми планетами, тому я припускаю, що це означає, що ми поки прив'язані до *Kepler*-90. Докладнішетут.

2 Химерний підхід є необов'язковим

Я сама не своя від*TensorFlow Eager*.

Якщо ви пробували *TensorFlow* в колишні часи і з криками втекли від нього, тому що вам доводилося кодіть як вченому або як інопланетянинові, а не як розробнику, возвращаааайтееесь!

Миттєве виконання (*eager execution*) в *TensorFlow* дозволяє вам взаємодіяти з ним як справжній програміст на *Python*: вам дається вся безпосередність написання коду та налагодження його рядок за рядком, а не напружене побудова графіків. Я сама раніше була вченим (і, цілком можливо, інопланетянином), але я була в захваті від миттєвого виконання *TF* коли він вийшов. Догоджає миттєво!

Training loop - eager execution

```
optimizer = tf.train.MomentumOptimizer(...)

for (x, y) in dataset.make_one_shot_iterator():
  with tf.GradientTape() as g:
    y_ = model(x)
    loss = loss_fn(y, y_)
  grads = g.gradient(y_, model.variables)
  optimizer.apply_gradients(zip(grads, model.variables))
```

Keras + *TensorFlow* = спрощена конструкція нейронної мережі!

Keras - це саме зручність і простота прототипирования, *TensorFlow* вони були давно потрібні. Якщо вам подобається об'єктно-орієнтоване мислення, і вам подобається будувати нейронні мережі по одному шару за раз, вам сподобається *tf.keras*. У деяких рядках коду нижче ми створили послідовну нейронну мережу зі стандартними свистілки і те й пісенька у вигляді списку (як небудь я впаду в ліричний настрій і видам вам свою метафору про випадних списках, в ній буду степлери і грип).

Define models

`tf.keras` provides a simple, expressive API to construct models
(or use low level operations like tf.nn.conv2d directly)

```
L = tf.keras.layers
model = tf.keras.Sequential([
    L.Reshape((28, 28, 1)),
    L.Conv2D(32, 5, activation=tf.nn.relu),
    L.MaxPooling2D((2, 2), (2, 2)),
    L.Conv2D(64, 5, activation=tf.nn.relu),
    L.MaxPooling2D((2, 2), (2, 2)),
    L.Flatten(),
    L.Dense(1024, activation=tf.nn.relu),
    L.Dropout(0.4),
    L.Dense(10),
])
```

О, вам подобаються головоломки, чи не так? Терпіння. Не думайте занадто багато про степлерах.

4 Це не тільки *Python*

Добре, ви скаржилися на одержимість *TensorFlow* Пітоном вже деякий час.

Гарні новини! *TensorFlow* тепер не тільки для пітонщіков. Тепер він працює на багатьох мовах, від *R* до *Swift* до *JavaScript*.

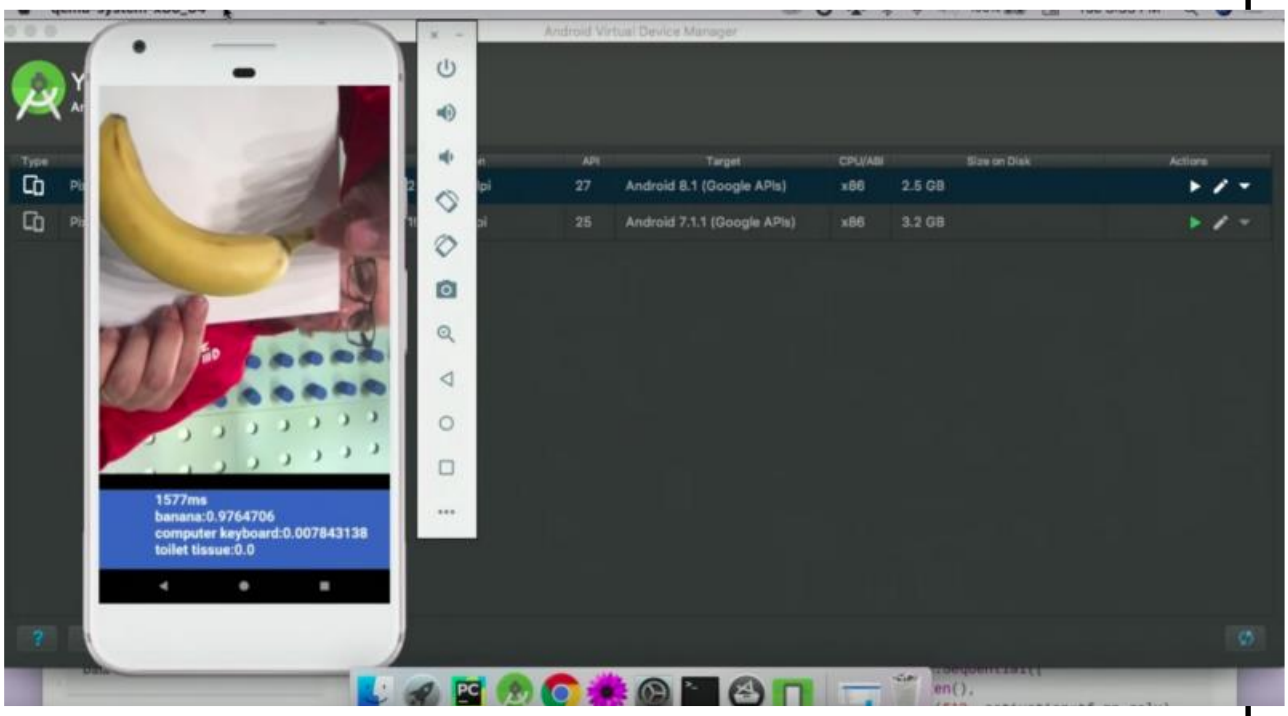
5 Ви можете робити все в браузері

Говорячи про *JavaScript,* ви можете навчати і виконувати моделі в браузері за допомогою*TensorFlow.js*.Поботайте зкласними демки я почекаю тут вашого повернення.

Оцінка людського пози в реальному часів браузері за допомогою *TensorFlow.js*. Увімкніть камеру для демонстрації тут. Або не вставайте зі стільця. ¯_(ツ)_/¯ Вам вирішувати.

6 Є *Lite* версія для маленьких пристроїв

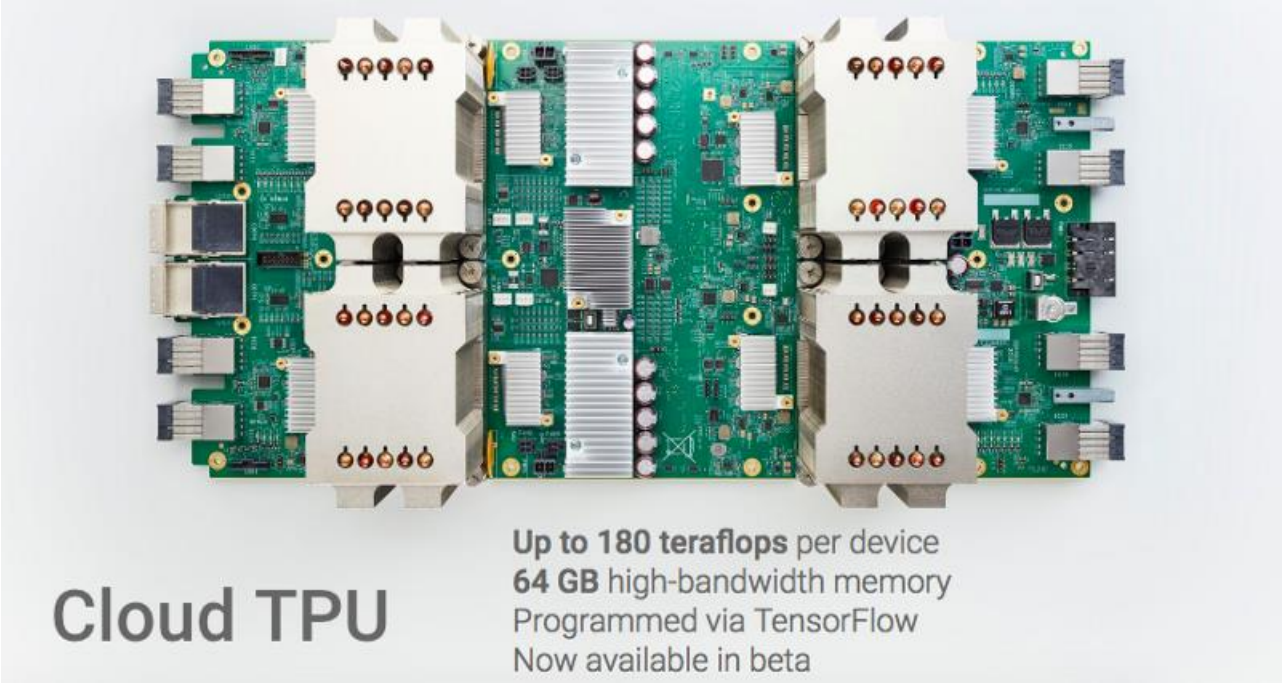
Так і не віднесли свій старовинний комп в музей? Є тостер? (Що-небудь таке ж марне?)*TensorFlow Lite*дозволяє виконання моделі на різних пристроях, в тому числі мобільниках і інтернету речей, що дає вам більш ніж 3-кратне прискорення виведення у порівнянні з оригінальним *TensorFlow*. Так, тепер ви можете отримати машинне навчання на своєму *Raspberry Pi* або на телефоні. В своїйпрезентації Лоуренс сміливо показує класифікацію зображень на емуляторі *Android* перед тисячами людей ... і це працює.



1,6 секунди для обчислення? Так! Банан з імовірністю більше 97%? Так! Туалетний папір? Ну, я була в декількох країнах, де аркуш паперу, типу того, який Лоуренс тримає в руках, вважаються за туалетний папір.

7 Спеціалізоване обладнання покращився

Якщо ви втомилися чекати, поки ваш процесор завершить роботу з вашими даними, щоб навчити вашу нейронну мережу, ви можете тепер використовувати свої апаратні засоби, спеціально розроблені для роботи зхмарними *TPU*. Т - значить тензор. *Як* і *TensorFlow* ... збіг? Не думаю! Кілька тижнів тому *Google* анонсувала версії *TPU* в версії 3.



8 Нові потоки даних значно поліпшені

Що ви робите з*numpy*? *Я*кщо ви хочете зробити це в *TensorFlow,* але зліться і кидаєте,простір імен *tf.data*тепер робить вашу обробку введення в *TensorFlow* більш виразною і ефективної. *tf.data* дає вамшвидкі, гнучкі і прості у використанніконвеєри даних, синхронізовані з навчанням.

9 Вам не потрібно починати з нуля

Ви знаєте, що не дуже цікаво почати машинне навчання? Порожня нова сторінка в вашому редакторі і ніякого зразкового код прикладу на кілометри навколо. З*TensorFlow Hub*ви можете використовувати більш ефективну версію перевіреної часом традиції позичати чужий код і називати його своїм власним (інакше відомої як професійна розробка програмного забезпечення).

2.2. Глибоке навчання з *Tensorflow*

2.2.1. Робота мережі глибинного навчання

Глибинне навчання - це методика реалізації машинного навчання. Він використовує нейронні мережі для навчання, іноді використання дерев рішень може також згадуватися як глибоке навчання, але здебільшого глибоке навчання передбачає використання нейронних мереж.

Отже, що таке нейронна мережа? Ось аналогія: уявіть собі нейронну мережу у вигляді ряду дверей один за одним і уявіть себе «входом» в нейронну мережу. Кожен раз, коли ви відкриваєте двері, ви стаєте іншою людиною. На той час, коли ви відкриваєте останню двері, ви стаєте зовсім іншою людиною. Коли ви виходите через останні двері, ви стаєте «виходом» нейронної мережі. У цьому випадку кожна двері являють собою шар. Отже, нейронна мережа - це набір шарів, які якимось чином перетворять вхідні дані для отримання вихідних даних.

Глибоке навчання з *Tensorflow*: Частина 2 - Класифікація зображень

класифікація зображень

Привіт всім, ласкаво просимо назад в мою серію *Tenserflow*, це частина 2.Я вже описав логіку і функціональність нейронних мереж і *Tenserflow* в першій частині, а також показав, як налаштувати середу кодування. Якщо ви ще не перевірили його, знайдіть йогоось,

Розпізнавання і класифікація зображень є темами цієї частини. Використовуючи модель *Inception-v*3, ми почнемо класифікувати зображення з використанням попередньо навченого набору даних *Google ImageNet*, а потім перейдемо до створення власного класифікатора. Давайте почнемо.

Що таке модель *inception-v*3?

початок *v*3Модельглибока сверточное нейронна мережаякий був попередньо підготовлений для*ImageNet*Велика проблема візуального розпізнавання з використанням даних 2012 року, і вона може розрізняти 1000 різних класів, таких як «кішка», «посудомийна машина» або «літак».

Команда *TensorFlow* вже підготувалакерівництвопро те, як виконати класифікацію зображень на вашому комп'ютері. Проте, я покажу і вам.

Класифікація зображень на попередньо навчений наборі даних *ImageNet*

Ну, оскільки ми не починаємо з нуля, почнемо з клонуванняРепо моделей *Tensorflow*Ситора від *GitHub*. Запустіть наступні команди:

git clone https://github.com/tensorflow/models.gitcd models / tutorials / image / imagenet

python classify_image.py

Якщо ви ще не встановили *Git*, завантажте йогоось,

*classify_image.py*завантажує навчену модель з бекенда *Google*, коли програма запускається в перший раз. Вам знадобиться близько 200 МБ вільного місця на жорсткому диску.

Наведені вище команди будуть класифікувати надане зображення ведмедя

панди.

Якщо модель працює правильно, скрипт видасть наступний висновок:

giant panda, panda, panda bear, coon bear, Ailuropoda melanoleuca (*score* = 0.88493)

indri, indris, Indri indri, Indri brevicaudatus (*score* = 0.00878)

lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens (*score* = 0.00317)

custard apple (*score* = 0.00149)

earthstar (*score* = 0.00127)

Якщо ви хочете надати інші зображення, ви можете зробити це, відредагувавши--*image_file*аргумент:

python classify_image.py --iAmAdifferentImage.jpg

Це було не так складно, правда? Але не хвилюйтеся, це буде більш складним. Наведений вище скрипт дозволяє нам класифікувати зображення на попередньо навчених класах командою *Google*. Але що, якщо ви хочете перевчити *Inception* для наших власних занять?

Останній шар *Retrain Inception* для певних категорій

Початкова модель шару

налаштувати

Ми додамо чотири нових класу в модель «Початок», які називаються «коло», «квадрат», «плюс» і «трикутник».

Почніть з клонування мого сховища з *GitHub*, введіть:

git clonehttps://github.com/koflerm/tensorflow-image-classifier.git

Після того, як ви клонували вказаний репозиторій, ми повинні якось повідомити *Inception*, яка правильна мітка для кожного зображення.

Оскільки ми класифікуємо, чи є об'єкт трикутником, квадратом, плюсом або гуртком, нам потрібно додати каталог «*training_dataset*» і заповнити його чотирма подпапками, названими на честь мітки класу. Ці папки міститимуть набори даних (зображення) для суб'єктів, для яких повинна бути виконана класифікація.

/ --- / *training_dataset* | || --- / *circle*

| | *circle*1.*jpg* | | *circle_small_red.png* | | ... | || --- / *square* | *square.jpg* | *square*3.*jpg* | ...

Потім ми повинні отримати набір даних, який використовується для процесу навчання. Оскільки завантаження кожного окремого зображення вручну займає дуже багато часу, ми використовуємо розширення *Chrome* з ім'ям *namedFatkun Batch* Завантажити зображення». Це дозволяє нам автоматично завантажувати перші 50 зображень з пошуку картинок *Google*. Просто скопіюйте їх в папки (формат зображення не важливий!)

зразкові тренувальні образи для класу «плюс»

проведення

Після налаштування всіх каталогів і наборів даних почніть навчання! виконати*train.sh*Сценарій, двічі клацнувши по ньому. Сценарій встановлює початкову модель (якщо вона ще не встановлена) і ініціює процес повторного навчання для зазначених наборів даних зображень.

```
100 bottleneck files created.
200 bottleneck files created.
300 bottleneck files created.
400 bottleneck files created.
500 bottleneck files created.
600 bottleneck files created.
2017-08-01 09:59:17.258806: Step 0: Train accuracy = 35.0%
2017-08-01 09:59:17.258806: Step 0: Cross entropy = 1.292076
2017-08-01 09:59:17.383817: Step 0: Validation accuracy = 19.0% (N=100)
2017-08-01 09:59:29.591315: Step 100: Train accuracy = 96.0%
2017-08-01 09:59:29.591315: Step 100: Cross entropy = 0.335396
2017-08-01 09:59:29.716326: Step 100: Validation accuracy = 91.0% (N=100)
2017-08-01 09:59:41.861543: Step 200: Train accuracy = 98.0%
2017-08-01 09:59:41.861543: Step 200: Cross entropy = 0.189951
2017-08-01 09:59:41.986550: Step 200: Validation accuracy = 93.0% (N=100)
2017-08-01 09:59:54.096292: Step 300: Train accuracy = 98.0%
2017-08-01 09:59:54.096292: Step 300: Cross entropy = 0.182122
2017-08-01 09:59:54.221307: Step 300: Validation accuracy = 91.0% (N=100)
```

процес перенавчання

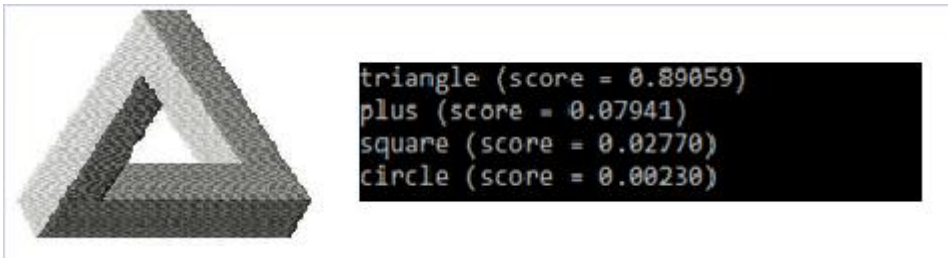
Як тільки процес завершиться, він поверне точність навчання десь близько 90 відсотків.

Перенавчання мітки, графи і підсумки навчання будуть збережені в папці з ім'ям*tf_files*на випадок, якщо ви захочете поглянути на це.

Після перенавчання моделі прийшов час протестувати модель на наших

власних зображеннях. Скачайте або намалюйте інше зображення і скопіюйте його в кореневий каталог. Я завантажив інше зображення «трикутника». Виконайте тест, набравши:

python classify.py downloadedImage.jpg



зліва: *loadedImage.jpg* | правильно: оцінка за кожний клас

Відмінно, мій трикутник фактично класифікований як трикутник!

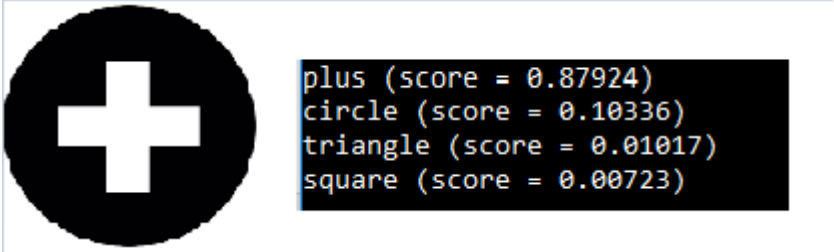
(За потребою): якщо ви додали кілька зображень або підпапок для нових класів, але не хочете викликати*train.sh*а потім*classify.sh,* Ви можете об'єднати введення, набравши:

python retrain.py - bottleneck_dir = tf_files / bottlenecks - how_many_training_steps = 500 *- model_dir = inception - summaries_dir = tf_files / training_summaries / basic - output_graph = tf_files / retrained_graph.pb - output_labels = tf_files / retrained_labels.txt - image_dir = training_dataset - eval_step_interval* = 100 & *python classify.py image.jpg*

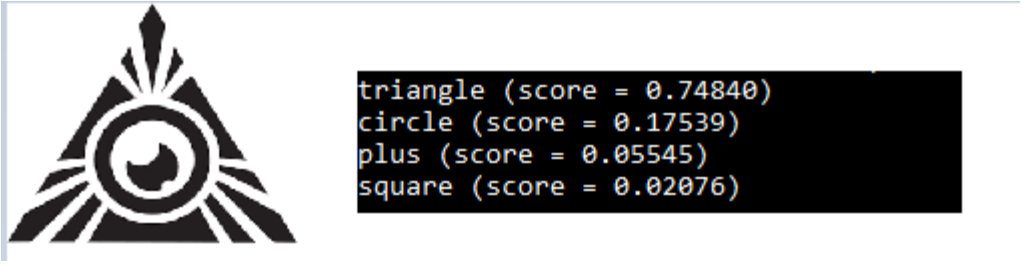
Примітка: *image.jpg* це зображення, яке ви хочете перевірити

питання класифікації

Хоча класифікація працює більшу частину часу, є деякі проблеми:



Отримує визнання як плюс



Знову не визнаний кругом

Початковий етап підготовлений для класифікації зображень по одній мітці, що означає, що класифікація за кількома матюками неможлива. Якщо я все ще хотів, щоб вищезгадані зображення були класифіковані як гуртки, я просто повинен був використовувати більш точний тренувальний набір для класу. Це насправді трохи більше часу, саме по собі розширення *Chrome* просто не може дати вам все кращі результати.

Добре, ми бачили, що програма робить те, що повинна (принаймні, більшу частину часу) Але як пройшов процес перенавчання?

По суті, скрипт завантажує попередньо навчену модель *Inception v*3, видаляє старий верхній шар і навчає новий класами геометричних фігур, які ви хочете додати. Це називається трансферним навчанням. Перепідготовка проходить в два етапи - Вузьке місце і Навчання:

На першому етапі аналізуються всі зображення на диску і розраховуються значення вузьких місць для кожного з них. «Вузьке місце» - це неформальний термін, використовуваний для шару безпосередньо перед фінальним вихідним шаром, який фактично виконує класифікацію. Це обробляє зображення до змістовного і компактному резюме зображень.

Другий етап - це фактичне навчання верхнього рівня мережі. Ви можете побачити серію покрокових виходів, кожен з яких показує точність навчання, точність перевірки і перехресну ентропію. Точність навчання показує, який відсоток зображень, використаних в навчанні, був позначений правильним класом. Точність перевірки - це точність випадково обраної групи зображень з іншого набору. Перехресна ентропія - це функція втрат, яка дає уявлення про те, наскільки добре йде процес навчання.

Якщо хочеш копнути глибше, йдисюди

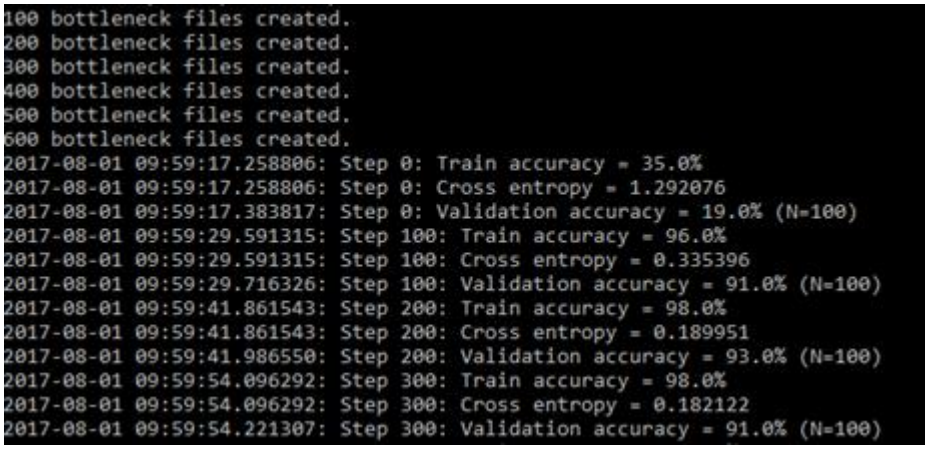
висновок

Вітаємо, тепер ви можете класифікувати зображення за допомогою *Tensorflow*. У третій частині я покажу вам інші варіанти використання *Tensorflow*. Так що стежте за оновленнями до наступного поста!

Глибоке навчання з *Tensorflow*: частина 4 - класифікація осіб і відео входи

Оскільки класифікація зображень є найбільш цікавою частиною, коли мова йде про нейронних мережах (по крайней мере, на мій погляд), ми зробимо це знову в цій частині. Але замість геометричних форм ми будемо класифікувати зображення людей, перш ніж ми перейдемо до розбудови нашої класифікатора, щоб приймати відеовходи. Давай зробимо це.

Озираючись на наш класифікатор, побудований у другій частині



як працював наш класифікатор?

Ви, ймовірно, не пам'ятаєте, про що була друга частина, тому ось коротке резюме:

Ми створили програму, яка завантажила попередньо навчену модель *Inception v3* з *Google*, видалила старий верхній шар і навчила новий класами геометричних фігур, які ми хотіли додати. Перенавчання працювало в два етапи - Вузьке місце і Навчання. «Вузьке місце» опрацювало наші зображення для отримання змістовного і піддається узагальненню. Навчання фактично навчило верхній шар нашої нейронної мережі. Якщо це все ще неясно, будь ласка, прочитайтечастина 2;)

Класифікація людей

Минулого разу ми додали чотири класи геометричних фігур в нашу початкову модель. На цей раз ми класифікуємо трьох різних людей: «Стів

Джобс», «Білл Гейтс» і «Марк Цукерберг». Давайте подивимося, чи достатньо сильне початок, щоб розрізняти обличчя цих людей:

Почніть з клонування мого сховища з *GitHub,* введіть:

git clonehttps://github.com/koflerm/tensorflow-image-classifier.git

Далі нам потрібно додати наші тренувальні образи і позначити їх. Створіть структуру як це:

/ --- / *training_dataset* | || --- / *billgates* | | *billy.jpg* | | *bill_gates.png* | | ... | || --- / *stevejobs* | *stevejobs.jpg* | *jobs.jpg* | ... | || --- / *markzuckerberg* | *zuckerberg*46*.jpg* | *zuckerberg*2020*.jpg* | ...

Для завантаження зображень я використовую розширення *Chrome* з ім'ям 'Масова завантаження зображень». Це розширення дозволяє автоматично завантажувати *Google Images* на ваш комп'ютер. Формат зображення не важливий!

як можуть виглядати зображення для класу Стіва-Джобса

Час тренувати нашу модель

Після настройки даних почніть навчання! виконати*train.sh*Сценарій, двічі клацнувши по ньому. Цей сценарій встановлює початкову модель і ініціює процес повторного навчання для зазначених наборів даних зображень.

```
100 bottleneck files created.
200 bottleneck files created.
300 bottleneck files created.
400 bottleneck files created.
500 bottleneck files created.
600 bottleneck files created.
2017-08-01 09:59:17.258806: Step 0: Train accuracy = 35.0%
2017-08-01 09:59:17.258806: Step 0: Cross entropy = 1.292076
2017-08-01 09:59:17.383817: Step 0: Validation accuracy = 19.0% (N=100)
2017-08-01 09:59:29.591315: Step 100: Train accuracy = 96.0%
2017-08-01 09:59:29.591315: Step 100: Cross entropy = 0.335396
2017-08-01 09:59:29.716326: Step 100: Validation accuracy = 91.0% (N=100)
2017-08-01 09:59:41.861543: Step 200: Train accuracy = 98.0%
2017-08-01 09:59:41.861543: Step 200: Cross entropy = 0.189951
2017-08-01 09:59:41.986550: Step 200: Validation accuracy = 93.0% (N=100)
2017-08-01 09:59:54.096292: Step 300: Train accuracy = 98.0%
2017-08-01 09:59:54.096292: Step 300: Cross entropy = 0.182122
2017-08-01 09:59:54.221307: Step 300: Validation accuracy = 91.0% (N=100)
```

процес перенавчання

Як тільки процес завершений, наша точність навчання повинна бути десь близько 90 відсотків.

Після перенавчання моделі прийшов час протестувати модель з іншими зображеннями. Завантажте та скопіюйте їх в кореневий каталог і введіть:

python classify.py downloadedBillGates.jpg

В порядку, що це добре. Але як працює наша модель?

```
billgates (score = 0.75619)
stevejobs (score = 0.13700)
markzuckerberg (score = 0.10680)
```

модель працює!

Білл Гейтс дивиться прямо в камеру, світло не поганий, так що немає проблем.


```
markzuckerberg (score = 0.96866)
billgates (score = 0.01945)
stevejobs (score = 0.01189)
```

Успіх!

Дивно, але потрібну людину знову дізнаються. Дозвіл зображення було поганим (300x300), а Цукерберг навіть не в центрі зображення. Молодець,

Початок!

Обидва визнані, але обидва не можуть отримати високий бал

Це складно! Початковий етап навчений класифікації зображень по одній мітці, що означає, що він становить в цілому один бал. Класифікація за кількома матюками неможлива, і оскільки на малюнку показані два наших класу, обидва не можуть отримати високий бал. Якби я хотів, щоб зображення було класифіковано як «Стів Джобс», я повинен був використовувати більш точний і більший тренувальний набір для свого класу.

Знову Гейтс над Джобсом

Знову ж таки, зображення класифікується як *Gates over Jobs*, на цей раз з величезним відривом. Я припускаю, що більш точний тренувальний набір і відповідно більше тренувальних даних в таких точних умовах освітлення можуть виграти Стіву Джобсу в цій дуелі.

Все про Стіва Джобса на цей раз

Нарешті, чоловік пізнається! Але це було не складно для нашої моделі, правда (відмінне освітлення, він дивиться прямо в камеру, ...)


```
billgates (score = 0.47070)
markzuckerberg (score = 0.41991)
stevejobs (score = 0.10940)
```

Тестування на зображеннях з іншими, невідомими людьми

Гейтса дізнаються, але Марка Цукерберга теж (хоча його немає на фотографії). Я припускаю, що *Inception* думав про Сандро Розелле (зліва) як про Марка Цукерберга, але так, безумовно невідповідність!

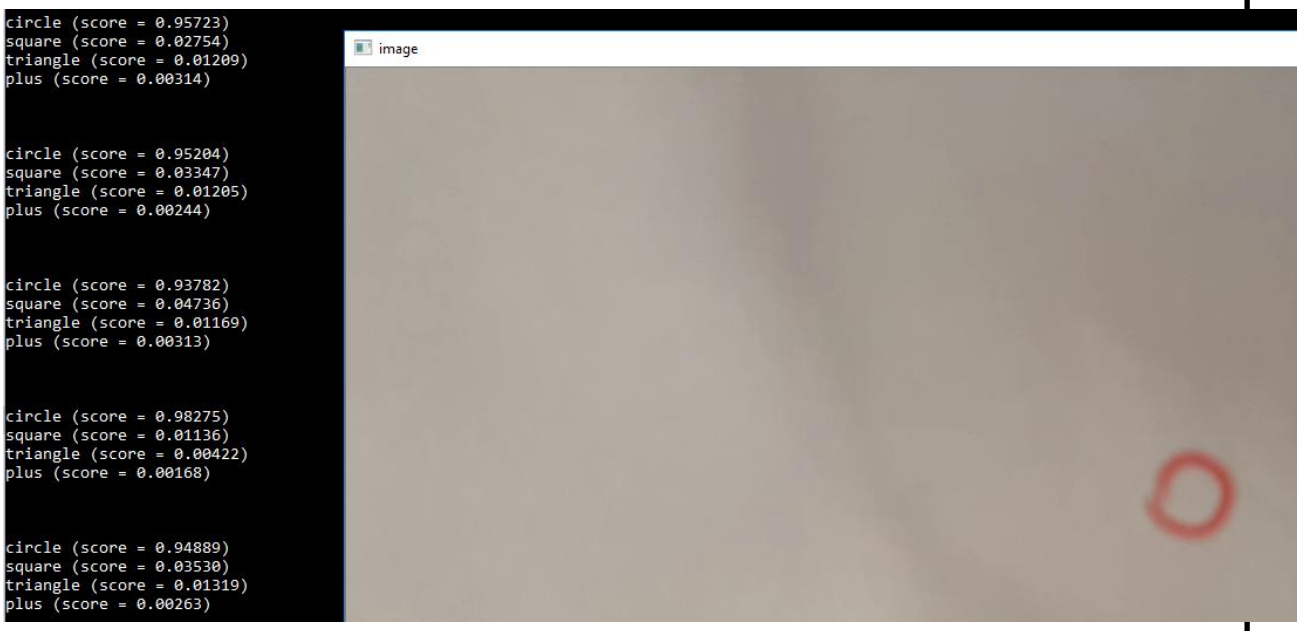
(За потребою): якщо ви хочете додати нові класи, ви можете об'єднати навчання і тестування, набравши:

python retrain.py - bottleneck_dir = tf_files / bottlenecks - how_many_training_steps = 500 *- model_dir = inception - summaries_dir = tf_files / training_summaries / basic - output_graph = tf_files / retrained_graph.pb - output_labels = tf_files / retrained_labels.txt - image_dir = training_dataset - eval_step_interval* = 100 & *python classify.py image.jpg*

Примітка: *image.jpg* - це зображення, з яким ви хочете протестувати свою модель згодом

Там у вас це є, ми можемо класифікувати особи з початковим. Але зараз давайте перейдемо до розбудови нашої класифікатора для відеовходів.

Відновлення нашої класифікатора для введення відео



класифікація зображень з використанням відео в якості вхідного потоку і покадровий аналіз

До сих пір ми завантажували наші тестові дані зображення за зображенням, але хіба це не копітко? Набагато простіше протестувати нашу модель з вхідним відео. Нам потрібно було вставити відео тільки один раз, але класифікацію зробили покадрово. Набагато простіше Тому давайте побудуємо таку систему:

Отже, це важлива частина коду. Якщо ви хочете заглибитися в це, будь ласка, прочитайте коментарі, я думаю, вони говорять самі за себе. Принцип простий: коли ми отримуємо видеокадр, ми зберігаємо його і відразу ж знову читаємо його для цілей класифікації, перш ніж виводити результати і сам кадр в окремому вікні.

Тепер давайте перевіримо нашу модель з відео Білла Гейтса і Стіва Джобса, що розмовляють один з одним.

Але по-перше, я думаю, що ви повинні тим часом ... Виконати*train.sh*і дочекатися завершення процесу :)

Потім починається найцікавіше. Я протестують свою модель на наступному відео:

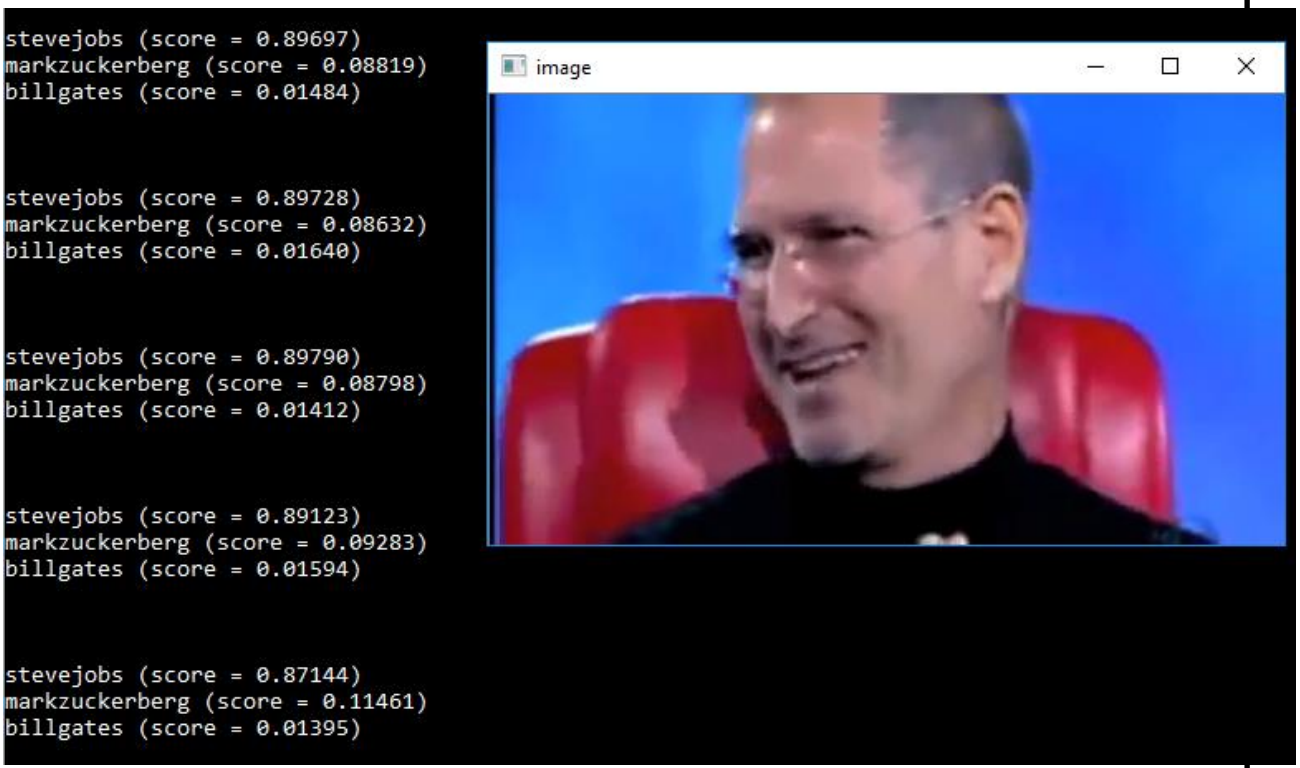
Відео не містить Марк Цукерберг

стратити*python classify.py myVideo.mp*4Потім нам доведеться довго чекати,

поки Білл Гейтс врешті не буде вперше знято. Яка реакція класифікатора?

Гейтс дізнаються більш-менш

Як ми бачимо, модель насправді класифікує його правильно, хоча голова Білла Гейтса трохи відрубана. Ми точно знаємо, що наша модель тепер працює з відео, але розпізнає вона і Стіва Джобса?



Так. Насправді з Джобсом навіть краще, рахунок близько 0,9. Але одне

питання все ще залишається відкритим: що станеться, якщо обидва чоловіки перебувають в одному кадрі?

Ну, кадр класифікується як «Робота». Як пояснювалося раніше, бали складаються в один, обидва чоловіки не можуть отримати високий бал.

Настав час прийняти новий виклик: ручні малюнки. Чи буде наша модель розпізнавати намальовані від руки об'єкти?

Ми знову додамо наші чотири геометричних класу «коло», «трикутник», «квадрат» і «плюс», щоб зробити це:

Давайте перевіримо нашу модель з відео-малюнками в якості вхідних даних

Отже, спочатку ми повинні знову додати наші класи в модель, на випадок, якщо ви видалили їх починаючи з другої частини. В кінці повинна бути така структура в папці навчання.

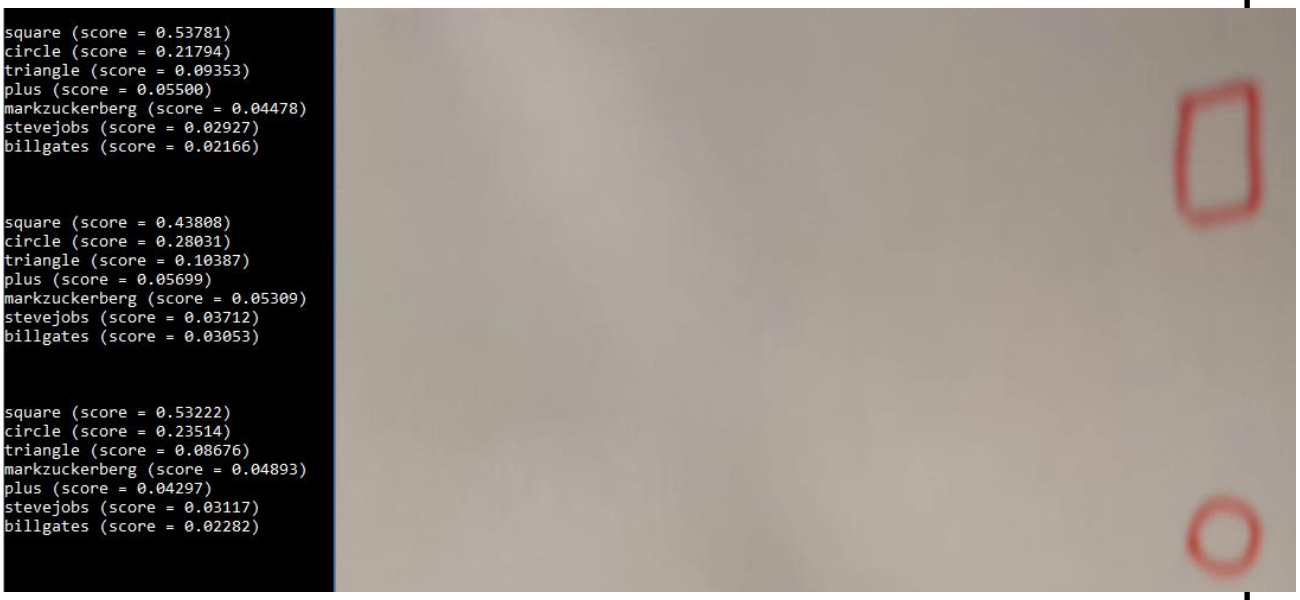
/ --- / *training_dataset* | || --- / *billgates* | --- / *stevejobs* | --- / *markzuckerberg* | --- / *circle* | --- / *square* | --- / *plus* | --- / *triangle*

Заповніть всі ці папки навчальними зображеннями. Я буду заповнювати їх зображеннями з пошуку *Google*, хоча ці зображення не будуть ручними малюнками! Буде цікаво подивитися, як це піде.

Далі тренування. запустити*train.sh*скрипт для додавання нових класів в модель, перш ніж знімати власне відео з геометричними фігурами. Ви також можете використовувати мій, знайти його як*math_own_old.mp*4в моєму репо на

GitHub.

Отже, давайте класифікувати деякі кадри. Тип*python classify.py math_own_old.mp*4в вашому *CLI* і подивіться, чи може ваша модель розпізнавати правильні форми.



Квадрат класифікується правильно

Яке дуже добрий початок! Квадрати - це успіх, а як щодо інших об'єктів?

Наша модель помилки трикутника

Ну от і все. Наша модель не розпізнає намальований від руки трикутник, вона вважає його квадратним. Може бути, наступний ...

Знову трикутник не розпізнано

Це коштувало спробувати, але він знову думає про це як про квадраті. Як не дивно, плюс, схоже, невидимий і для моделі. Коло - наша остання надія!

Успіх!

Так, коло класифікований правильно! Але, як видно, моделі, навченої комп'ютерними зображеннями, дуже важко розпізнати погано дозволені намальовані від руки картинки.

2.5. Висновки до розділу

Представлено методи класифікації зображень системою , у вас це є, можна класифікувати людей за допомогою *Inception* і *Tensorflow*. Крім того, ви дізналися, як переписати деякий код, щоб використовувати відео в якості джерела введення. Це була остання частина серії, сподіваюся, вам сподобалася подорож.

РОЗДІЛ 3

ОПИСАННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МОДУЛЬ КЛАСИФІКАЦІЇ ДАНИХ ВІДЕОХОСТИНГУ *YOUTUBE*

3.1. *TensorFlowJS*: використання навчених моделей без їх модифікацій в браузері

При роботі з *TensorFlowJS* можна виділити три напрямки його використання (рисунок 1):

Використання навчених моделей без модифікацій топологій і без їх перенавчання

Використання попередньо навчену модель з можливою модифікацією її топології або перенавчанні на базі нової тренувальної вибірки даних, яка повністю відповідає розв'язуваної вашого завдання

Створення і навчання моделей з нуля

У даній статті, ми детально розглянемо як ви можете використовувати вже навчені моделі без модифікації топології і з перенавчанні, тобто перший напрямок використання бібліотеки.

1. Використання моделей через абстрактний *API*

Цей той випадок, коли вам не будуть потрібні якісь глибокі знання по машинному навчання в принципі. Вам навіть не доведеться робити ніяких маніпуляцій з вхідними даними для моделі. Наприклад, якщо шукана модель приймає на вхід зображення розмірністю 240*x*240 пікселів, а ви хотіли б використовувати зображення 20*x*30, то вам би довелося робити маніпуляції над зображенням, щоб вона була сумісна з моделлю.

Також досить часто, моделі вимагають нормалізацію вхідних даних (значення в одному кольором каналі для пікселів змінюються від 0 до 256, однак для кращої збіжності моделі можуть іноді вимагати, щоб величина кожного пікселя була в інтервалі [0, 1] або [-1, 1]).

Однак цей тип моделей абстраговані таким чином, що єдиним слідом того, що всередині моделі використовується *TensorFlowJS* - це буде тільки імпорт цієї бібліотеки разом з моделлю.

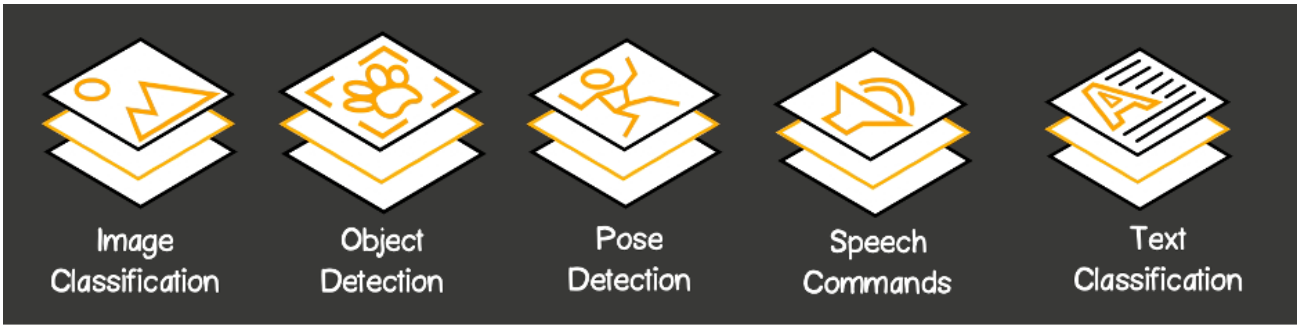


Рис. 3.1. Класифікація моделей за типом розв'язуваних задач

На момент написання статті (жовтень 2020), *Google* представив 13 офіційних моделей для використання у відкритому доступі. Список моделей може бути знайденийтут. У списку ви можете знайти моделі, які вирішують такі типи завдань (малюнок 2):

класифікація зображень: *MobileNet* - класифікація зображень між 1000 категорій, при цьому на зображенні повинен знаходиться об'єкт одного класу з нейтральним фоном (посилання)

виявлення об'єктів на зображенні з зазначенням просторових координат: *Coco-SSD* - визначення об'єктів на зображенні з зазначенням вікна, в якому знаходиться об'єкт; модель може розпізнавати 80 різних категорій об'єктів (посилання)

визначення положення людського тіла:*BodyPix* - просторове визначення частин тіла (руки, ноги, плечі, очі, вуха), а також за допомогою моделі можна побудувати маску положення тіла. На зображенні одночасно може перебувати кілька людей (посилання)

розпізнавання голосових команд: *SpeechCommands* - розпізнавання звукових команд; в ісходнос стані модель може розпізнавати команди зі

словника з 20 слів англійською мовою, наприклад: '*go*', '*stop*', '*yes*', '*no*' (посилання)

текстова класифікація: *Toxicity* - визначає чи містить повідомлення не прийнятний контент, що містить образи, не повага, непристойні вирази з сексуальним змістом (посилання)

текстова класифікація: *Toxicity* - визначає чи містить повідомлення не прийнятний контент, що містить образи, не повага, непристойні вирази з сексуальним змістом (посилання) -.

В першу чергу, необхідно зробити імпорт моделі і бібліотеки *@ tensorflow / tfjs*, цей той єдиний слід, що говорить, що під капотом модель використовує *TensorFlowJS*. При першому рендерінге *React* компонента необхідно завантажити модель: *cocoSsd.load* (рядки 22-27). Як тільки модель буде завантажена (це може зайняти деякий час), можна запускати процес обробки кадрів, що отримуються з відео потоку викликом *detectFrame* функції (рядок 25). Для отримання метаінформації про становище об'єктів на зображенні - досить викликати метод *model.detect*, першим аргументом якого є посилання на *DOM* елемент *video* (рядок 14). Цей метод повертає *Promise*, результатом якого є масив з метаінформацією про кожен об'єкт, який був визначений моделлю в наступному форматі:

```
[{
 bbox: [x, y, width, height],
 class: "Person",
 score: 0.8380282521247864
}, {
 bbox: [x, y, width, height],
 class: "Kite",
 score: 0.74644153267145157
}]
```

Функція *buildObjectReactngle* - отрісовиває області на *canvas* навколо об'єктів, розпізнаних моделлю. *Canvas* накладено на відео потік зверху.

тутпосилання на *git-repository* з повним кодом.

2. Використання серіалізірованних навчених моделей *TensorFlow*

У зв'язку з тим, що деякі моделі можуть навчатися днями, тижнями або місяцями, то очевидним стає факт, що необхідний механізм збереження моделей на проміжних етапах навчання.

У *TensorFlow.js* в *API* є спеціальний метод *tf.GraphModel.save*. Модель зберігається в *TensorFlowJS JSON* форматі. Це набір файлів *model.json* і один або більше бінарних файлів (рисунок 3). Файл *model.json* містить інформацію про топології мережі і має вичерпну інформації про класи, з яких складається модель, а також конфігурації шарів моделі. Бінарні файли містять значення ваг всіх верств моделі і якщо модель має велике число параметрів, то вони можуть розбиватися на Шардена, по умовчання кожен бінарний файл не більше 4 Мб.

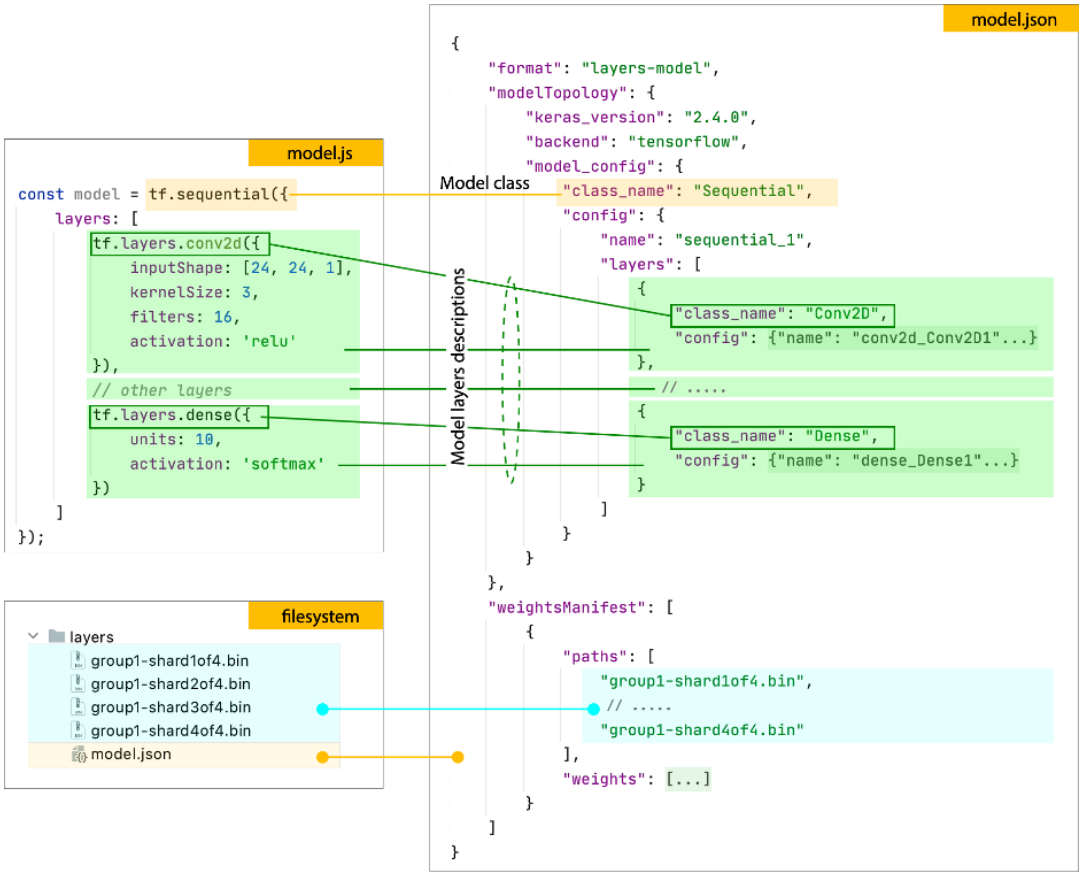


Рис. 3.2. Структура збереженої моделі в форматі *TensorFlow.js JSON*.

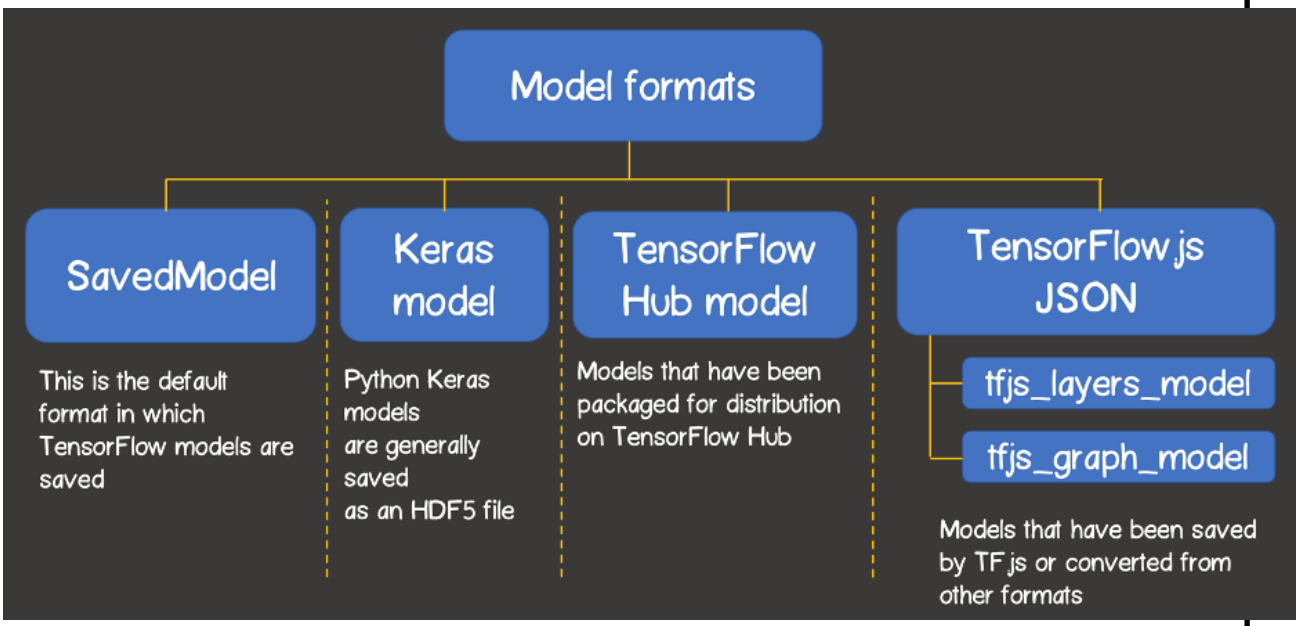
Насправді *TensorFlowJS* може працювати не тільки з моделями, які були збережені ним же, але так само є можливість працювати з моделями, які були навчені за допомогою фреймворка *Keras* на мові *Python*, але збережених за допомогою *TensorFlow* в форматі *SavedModel*. Всього є 3 типи формату

сериализации моделей, які ви зможете використовувати з *TensorFlowJS* (малюнок 4):

TensorFlow SavedModel - формат моделі за замовчуванням, з якими моделі зберігаються з допомогу *TensorFlow*;

Keras Model - моделі, які зберігаються фреймворком *Keras* в форматі *HDF*5;

TensorFlow Hub Model - моделі, які поширюються через спеціальну платформу *TensorFlow Hub*.



Однак перед використання *SavedModel, Keras Model, Tensorflow Hub model* в *TensorFlowJS* необхідно конвертувати ці моделі за допомогою*tfjs_converter*.

Тут слід звернути увагу, що моделі в форматах *SavedModel* і *TensorFlow Hub* можуть бути конвертовані тільки в формат *tfjs_graph_model*. Моделі ж, збережені у форматі *Keras*, можуть конвертовані як в формат *tfjs_graph_model*, так і в формат *tfjs_layers_model*.

Чим же відрізняються *tfjs_layers_model* від *tfjs_graph_model*?

Моделі в форматі *tf_js_graph_model* перетворюються в екземпляр класу *tf.FrozenModel*, що означає що всі параметри моделі зафіксовані і не підлягають зміні. Таким чином, якщо ви хотіли б перенавчити модель з нової вибіркою вхідних даних, яка більш релевантна розв'язуваної вами завдання, або ж змінити

топологію моделі на базі існуючої - то вам такий формат моделі не підійде. Однак є перевага для цієї моделі - це що обчислення (*inference time*) буде значно менше, в порівнянні з тією ж моделлю, але перетвореної в формат *tfjs_layers_model*.

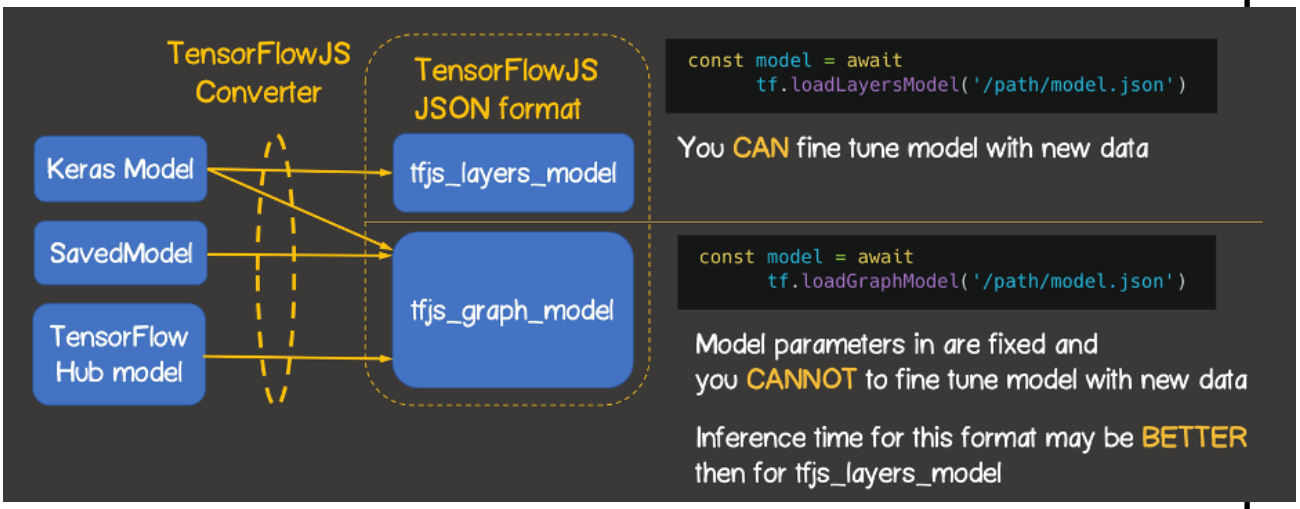
Модель в цьому форматі завантажується за допомогою наступного *API*:

const model = tf.loadGraphModel ('/Path/to/model.json')

Якщо є потреба в перенавчанні моделі або зміні її топології на стороні браузера, то модель повинна бути завантажена в форматі *tfjs_layers_model*. Модель в цьому форматі завантажується за допомогою наступного*API*:

const model = tf.loadLayersModel ('/Path/to/model.json')

На малюнку 5 представлена зведена схема з вищесказаного.



Малюнок 5 - Зведена таблиця по конвертації серіалізовать моделей

Конвертор можна використовувати прямо з командного рядка або ж його можна викликати безпосередньо в *Python*-скрипті.

Після роботи конвертора, в папці *path / to / tfjs_target_dir* ви побачите знайомі вже вам файли *model.json* і та бінарні файли містять значення ваг моделі.

Деякі помилки, з якими ви можете зіткнутися

Після того, як ви конвертували модель в *tfjs_layers_model* і ви намагаєтеся завантажити модель на клієнті за допомогою *tf.loadLayersModel*, то можете отримати такого роду помилку:

Якщо знову погляньте на малюнок 3, то побачите, що файл містить опис класів, на базі яких будується модель. При цьому ми бачимо що модель створена з допомогу класу *Functional*. Вам треба звернути увагу на версію *TF.js*, яку ви

використовуєте і його *API*. В даному випадку використовувався *TF.js* версії 2.0, і якщо ви подивіться на *API*, то побачите, що в *API* цієї версії немає класу *tf.Functional*. Саме тут криється і проблема. Як його можна дозволити: - оновити версію *TF.js* до версії, де в *API* представлений клас *tf.Functional*- в зв'язку з тим, що *tf.Functional extends tf.LayersModel*, то в *model.json* файлі замість *Functional* в поле *class_name* потрібно використовувати *Model*

Також можете подивитися про помилку на *stackoverflow*тут.

Як використовувати завантажену модель?

На відміну від упакованих моделей в абстрактний *API*, тут нам не уникнути знайомства з *TensorFlow*, так як крім завантаження моделі, нам також необхідно потурбуватись про передачі вхідних даних в модель.

Отже, давайте просто використовуємо попередньо конвертовану модель *MobileNetV*2 з формату *Keras* в формат *tfjs_graph_model* для класифікації зображень.

Після завантаження зображення, для того щоб зрозуміти в якому форматі нам треба передавати дані в модель, подивіться файл *model.json* "*inputs*" поле:

"*Inputs*": {
"*Input*_1: 0": {
"*Name*": "*Input*_1: 0",
"*Dtype*": "*DT_FLOAT*",
"*TensorShape*": {
"*Dim*": [
{ "*Size*": "-1"},
{ "*Size*": "224"},
{ "*Size*": "224"},
{ "*Size*": "3"}
]
}
}
},

Таким чином на вхід модель необхідно передати 4*D-tensor* розмірністю [*null*, 224, 224, 3], що відповідає [*EXAMPLE_BATCH*, *WIDTH*, *HEIGHT*, *COLOR_CHANNELS*]. Уздовж першої осі може розташовуватися одночасно декілька зображень, але так як ми будемо розробляти додаток, що дозволяє користувачам долучати тільки одне зображення, то в нашому випадку на вхід моделі завжди будемо передавати тензор розмірністю [1, 224, 224, 3].

Також поглянемо, що модель нам видасть в файлі *model.json* в поле "*outputs*":

```
"Outputs": {
 "Identity: 0": {
 "Name": "Identity: 0",
 "Dtype": "DT_FLOAT",
 "TensorShape": {
 "Dim": [
 { "Size": "-1" },
 { "Size": "1000" }
]
}
}
}
```

Це 2*D*-тензор розмірністю [*null* 1000], де 1000 - це кількість класів, на які наша мережа може класифікувати зображення. Це так званий *one-hot* вектор, в якому все значення рівні нулю, за винятком одного, наприклад [0, 0, 1, 0, 0] - це значить що нейронна мережа вважає з ймовірністю 1, що на зображенні клас з індексом 2 (індексація як зазвичай починається з нуля). Коли ми буде використовувати модель, то цей вектор буде являти собою розподіл ймовірності для кожного з класів, наприклад, можна отримати такі значення: [0.07, 0.1, 0.03, 0.75, 0.05]. Зверніть увагу, що сума всіх значень буде дорівнює 1, а модель вважає з максимальною ймовірністю 0.75, що це об'єкт класу *c* індексом 3.

При рендеринге реактив-компонента, ми завантажуємо модель, яка була конвертована з *Keras* (рядки 10-14). Як тільки модель буде завантажена -

користувачеві відкриється можливість завантажувати зображення для класифікації.

Як тільки користувач вибере зображення - ми для початку повинні перетворити зображення в тензор розмірністю [224, 224, 3], а так само необхідно нормалізувати дані, щоб значення пікселів було в проміжку [-1, 1]. Передамо цей тензор моделі і отримаємо вихідний тензор (рядок 30). Тепер нам треба всього лише зіставити текстові мітки з індексами, і вивести п'ять найбільш ймовірних класів для завантаженого зображення.

Зверніть тут увагу, що функція обгорнута в *tf.tidy* - це важлива деталь. Для збільшення продуктивності, *TensorFlow.js* в браузері зазвичай обчислення виробляє за допомогою *WebGL*, і на жаль тут немає механізму збирача сміття, який автоматично якось міг би зрозуміти, що деякі тензори вже не потрібні і можна вивільнити пам'ять з *WebGL*. Щоб цей процес не був ручним (бо кожен тензор в своєму *API* має метод *dispose*, який вивільняє пам'ять), ми обертаємо операції над тензором в *tf.tidy* і після виконання функції виживуть тільки ті тензори, які повертаються цією функцією, а все решта тензори будуть знищені. Так як ми нічого не повертаємо з функції - то все тензори будуть знищені після її виконання.

3.2. Класифікатор відео на *Python* з допомогою бібліотеки *TensorFlow*

Що повинен виконати алгоритм:

проаналізувати зображення і визначити, чи є на ньому квітка;

якщо є - визначити, до якої із запропонованих категорій він відноситься;

вивести відсоток упевненості в своїй відповіді.

Категорії кольорів, включені у вхідній набір даних: ромашки, кульбаби, троянди, соняшники і тюльпани.

Ми будемо використовувати *Inception v*3 - готову модель машинного навчання для тренування на нашому власному наборі даних. Ця нейронна мережа використовується в «Кампанії за широкомасштабного розпізнаванню образів *ImageNet*» (*ILSVRC - ImageNet Large Scale Visual Recognition Challenge*) і

раніше на ній проводилося навчання описуваного алгоритму на визначення категорій об'єктів на зображеннях.

Налаштування інструментів

Скористаємося пакетом контейнерів *Docker*, що надаються *TensorFlow*. Одним з переваг контейнера є те, що всі залежності, необхідні для запуску *TensorFlow*, вже присутні в контейнері, тому немає необхідності встановлювати що-небудь самостійно (проте, необхідно переконатися, що в контейнері встановлена остання версія бібліотеки *TensorFlow*.

Для цього в консолі *Docker* необхідно ввести*pip install --upgrade tensorflow*). дотримуйтесьцим вказівкам для установки панелі інструментів *Docker* на ПК.

Docker-контейнер працює за принципом невеликого комп'ютера, відокремленого від вашого основного комп'ютера. Він має свою власну файлову систему, а також йому буде призначений власний *IP*-адреса після успішного завершення установки. Щоб перевірити, чи правильно встановлено *Docker*, запустіть *Docker Daemon*, ввівши таку команду в основному терміналі:

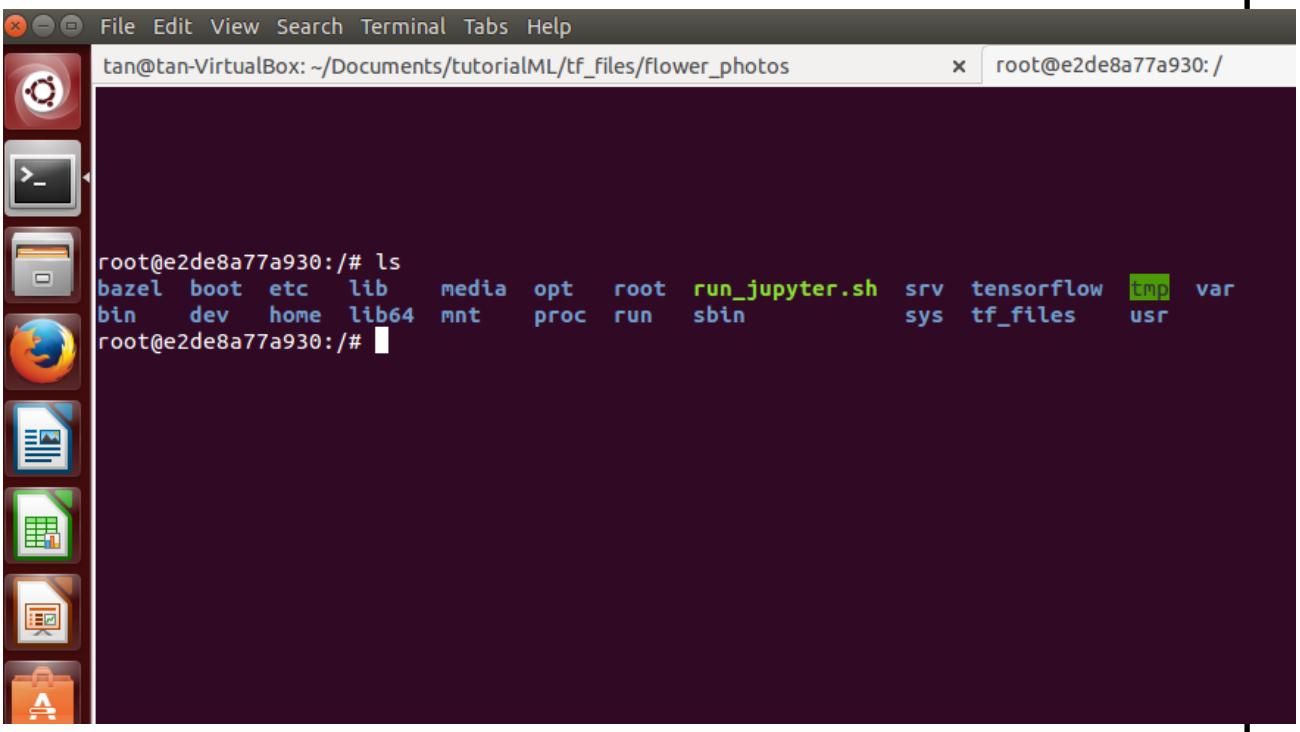
sudo service docker start

Запуск способу *Docker TensorFlow*

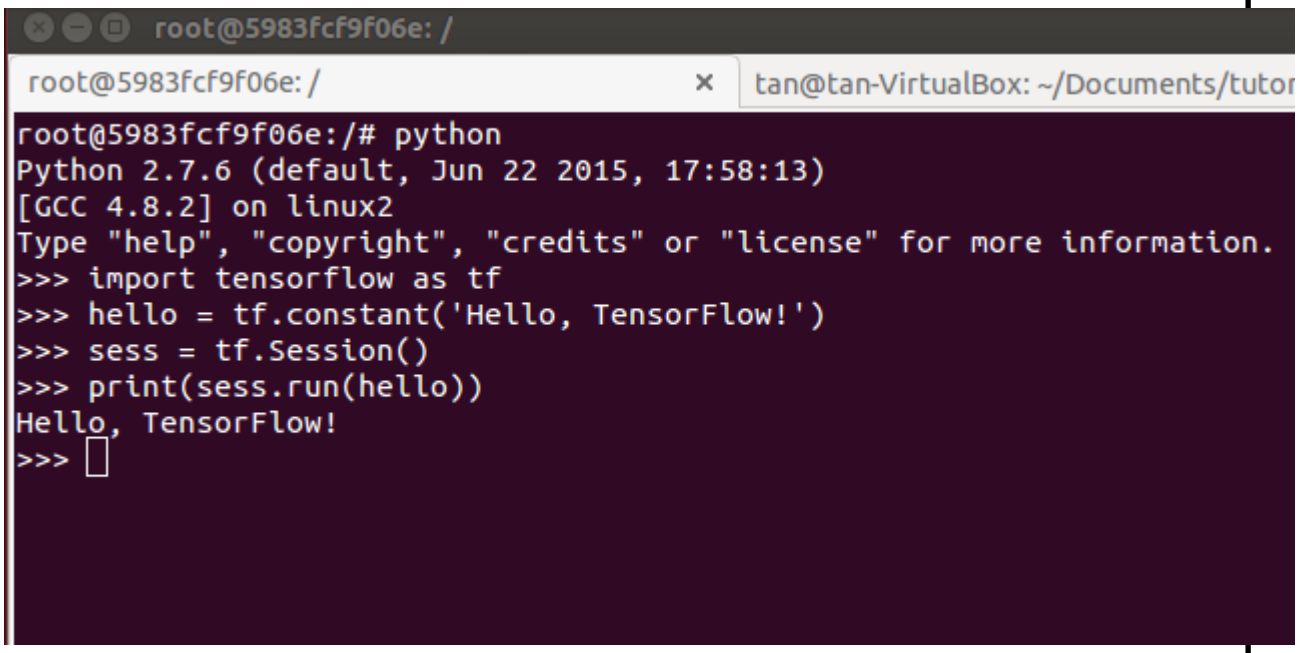
Встановіть образ *Docker TensorFlow*:

sudo docker run -it gcr.io/tensorflow/tensorflow:latest-devel

Після того, як ви введете цю команду, з'явиться новий користувач з рут-правами і довгим номером (*root @ xxx*), Як це показано на скріншоті нижче. Зверніть увагу, що в *Docker* буде знаходитися папка*tensorflow*.



Щоб переконатися в коректній роботі *TensorFlow*, ви можете набрати наступний код на *Python*, що складається з трьох рядків, для відображення вітання *TensorFlow* в терміналі *Docker*:



Підготовка вхідних даних для навчання

Щоб модель *Inception v*3 могла визначати квіти на зображеннях, необхідно підготувати вхідні дані, на яких алгоритм буде навчатися. Для цього спочатку слід створити каталог на вашому основному комп'ютері / віртуальній машині з

назвою*tf_files*, Шлях може виглядати приблизно так: ~ / *Documents* / *tutorial_ML* / *tf_files*.

Якщо ви все ще перебуваєте в терміналі *Docker*, можете натиснути *Ctrl* + *D*, щоб вийти з *Docker* і повернутися в термінал вашого основного комп'ютера / віртуальної машини, або створити нове вікно терміналу. Для завантаження набору зображень з квітами спочатку перейдіть в каталог*tf_files*, А потім введіть наступну команду в основному терміналі:

curl -O http://download.tensorflow.org/example_images/flower_photos.tgz

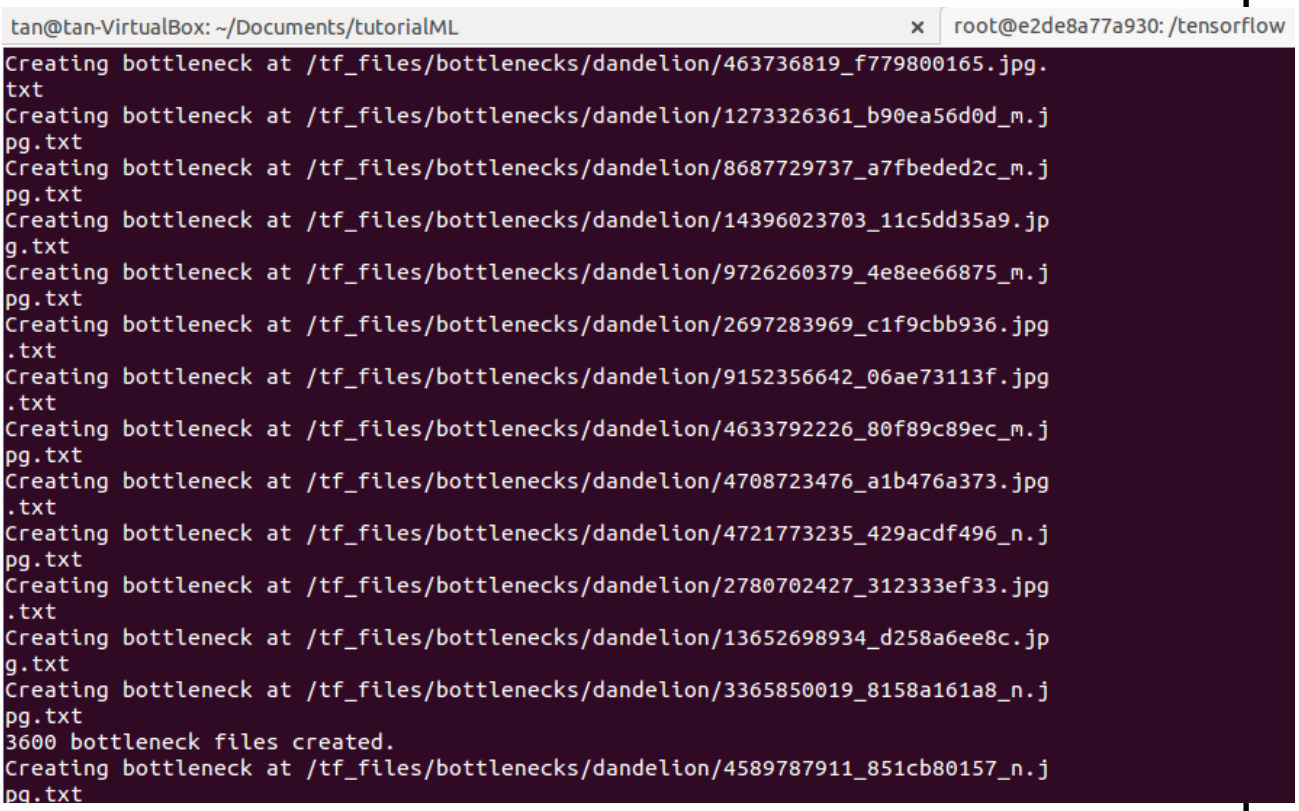
Разархівіруйте отриманий файл, ввівши цю команду:

tar xzf flower_photos.tgz

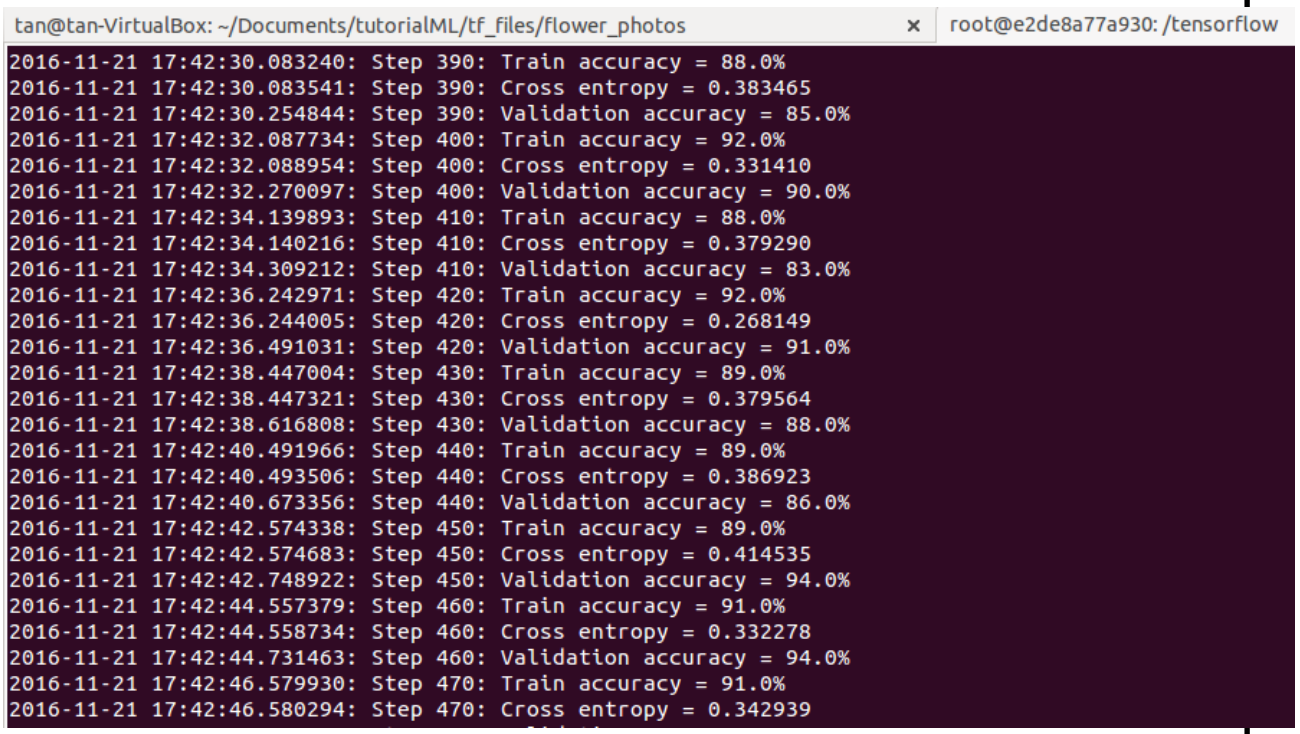
У каталозі *flower_photos*ви побачите піддиректорії з іменами:*daisy*(Ромашки),*dandelion*(Кульбаби),*roses*(Трояндиі),*sunflowers*(Соняшники) і*tulips*(Тюльпани).

Прив'язка набору даних до образу *Docker TensorFlow*

Поки модель *Inception* обробляється (а це займе тривалий час), ви побачите, що спочатку створюються критичні параметри (*bottlenecks*), як це показано на скріншоті нижче. Критичні параметри - це останній шар перед основним шаром початковій моделі, який буде класифікувати зображення. У них полягає змістовне, але при цьому компактне резюме зображення, яке допомагає класифікатором зробити найбільш точний вибір:

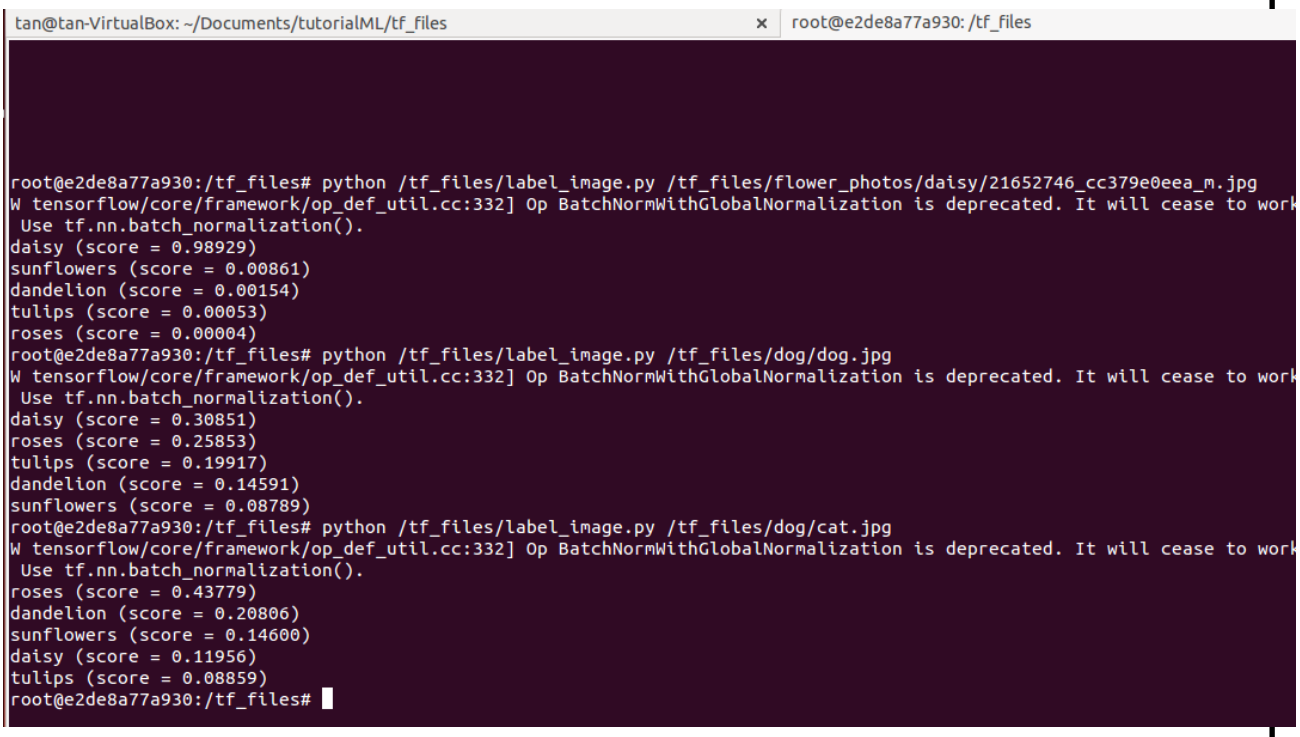


Після того, як навчання критичних параметрів завершиться, алгоритм перейде на фінальний етап навчання. Під час навчання в терміналі *Docker* будуть відображені наступні параметри: точність навчання (*train accuracy*) і точність перевірки (*validation accuracy*). Ви побачите аналогічну скриншоту нижче картину. В кінці буде відображено значення точності від 85% до 99%:



Тестування класифікатора зображень

Як видно на скріншоті нижче, при тестуванні обраного зображення було отримано число 0,98929. Це вказує на те, що класифікатор на 98% впевнений, що зображення містить квітка ромашки. Якщо аналогічним чином протестувати зображення собаки і кота, буде отримано низький відсоток упевненості для всіх категорій квітів. Це говорить про те, що алгоритм визначив, що наявність будь-якого квітки малоймовірно на тестованому зображенні:



3.5. Розробка модуля класифікації

Перш за все потрібно відкрити збережену раніше натреновану модель

Наступним кроком буде створення функції класифікації . В мові *Python* для цього потріно використати команду *def predict*(*str_query*,*numwords*): {}, де *def* -

Ініціалізація функції , *predict* – назва функції , *str_query* та *numwords* – атрибути функції .

Наступним кроком є використання функції *Tokenizer*() в яку надається змінна *numwords* котру було передано до функції для визначення розмірності масиву токенів та отримання даних з датасету для можливості створення двійкового масиву з вхідних даних ідентичного за структурою до масиву , що використовувався при навчанні, для уникнення можливих помилок та

максимізації точності результату командою

x_test = tokenizer.texts_to_matrix(X_raw_test, mode='binary') , де *tokenizer* – змінна що містить масив токенів , *X_raw_test* – змінна що містить назву відео котра класифікується , *mode='binary'* – визначення що масив буде саме двійковим.

Далі потрібно ініціалізувати функцію

prediction = model.predict(np.array(x_test)) , де *model.predict* - функція яка і буде класифікувати вхідні дані за відсотковим співвідношенням до кожної категорії , *np.array* – функція для перетворення двійкового масиву у двійковий вектор , *x_test* – двійковий масив що відповідає перетвореній вхідній назві відеоматеріалу.

Потім з отриманого масиву потрібно використати перший елемент (у даному випадку нумерація починається з 0), який є порібним нам списком коефіцієнтів принадлежності вхідної назви до кожної з категорій за допомогою команди *prediction_in = prediction*[0], де *prediction*[0] – це перший елемент матриці. Єдиним що залишається - це виведення отриманих результатів у користувацький інтерфейс за допомогою циклу , що почергово спвввідносить назву категорії та її значення

```
for name, index in classes.items():
    info = name + " " + str(round(prediction_in[index]*100,1)) + "%" + "\n"
    text_out.insert(float(index) , info) , де name – назва класу , str(round(prediction_in[index]*100,1)) – перетворення у символьні значення розрахунок відсотків принадлежності до категорії , text_out.insert(float(index) , info) – функція додавання результату до текстового поля користувацького інтерфейсу.
```

Також потрібно створити функції для оброблення дій після натискання кнопки “Старт” –

```
def enter(a):
    query = text_in.get(1.0 , END)
```

predict(*query,*1000),

де *query = text_in.get*(1.0 , *END*) – функція отримання введеної у спеціальне поле назви , *predict*(*query,*1000) – виклик функції описаної раніше .

Функцію очищення текстового поля виводу :

def clean(*a*):

text_out.delete(1.0 , *END*)

3.6. Користувацький інтерфейс

Користувацький інтерфейс створюється за допомогою спеціальзованої бібліотеки під назвою *Tkinter* , котра має всі необхідні набори функцій та подій для цього.

Бібліотека *Tkinter* містить компоненти графічного інтерфейсу користувача.

Існує безліч бібліотек *GUI,* серед яких *Tkinter* не самий популярний інструмент, хоча з його допомогою написано чимало проектів. Він був обраний для *Python* за замовчуванням.

До складу користувацького інтерфейсу входять 2 текстових поля (вводу та виводу) , 1 елемент типу “*Label*” , що являє собою назву поля для вводу , 2 кнопки (Старт та очищення поля виводу)

Створення користувацького інтерфейсу починається з функції *Tk*()

root = Tk()

Для того щоб назвати вікно користувацького інтерфейсу використовується метод *title*().

root.title("Класифікатор відео за назвою")

Визначення розмірів вікна відбувається за допомогою методу *geometry*().

root.geometry("650*x*500")

Робоча область на якій будуть знаходитися елементи визначається за

допомогою функції *Frame*():

fr = Frame(*root*)

fr.grid()

nal_gen = Frame(*fr*)

nal_gen.grid(*row* = 0 , *column* = 0 , *columnspan* = 1)

Для того щоб ця робоча область працювала потрібно її “скомпілювати ” методом *grid*().

Назва Назва текстового поля для вводу , котру не можна змінити під час роботи програми визначається за допомогою методу *Label*():

lname = Label(*nal_gen*)

При використанні методу *grid*() можна визначити розташування елементу використовуючи систему таблиць(рядків та стовпчиків):

lname.grid(*row* = 0 , *column* = 0)

текстового поля для вводу , котру не можна змінити під час роботи програми визначається за допомогою методу *Label*():

lname = Label(*nal_gen*)

При використанні методу *grid*() можна визначити розташування елементу використовуючи систему таблиць(рядків та стовпчиків):

lname.grid(*row* = 0 , *column* = 0)

Методом *configure*() можна задати зміст даного елемента :

lname.configure(*text* = "Введіть назву")

Текстові поля створюються методом *Text*() , який має налаштування в ширину, довжину та типажу поля:

text_in = Text(*nal_gen , width* = 50 , *height* = 1 , *wrap = WORD*)

text_in.grid(*row* = 0 , *column* = 1)

Аналогічним чином створюється поле для виводу інформації :

text_out = Text(*nal_gen , width* = 60 , *height* = 20 , *wrap = WORD*)

text_out.grid(*row* = 7 , *column* = 0 ,*columnspan* = 3 , *sticky = N*)

Функціональні кнопки створюються за допомогою методу *Button*() та мають методи *configure*() та *bind*() .

У методі *configure*() аналогічно до назви поля вказується текст що буде розміщений на кнопці . У методі *bind*() визначається тип кнопки(за замовчуванням '<*Button*-1>') та функція котрі буде викликана при натисканні.

Кнопка “Старт” :

bstart = *Button*(*nal_gen*)

bstart.*grid*(*row* = 0 , *column* = 4 , *columnspan* = 3)

bstart.*configure*(*text* = " Старт ")

bstart.*bind*('<*Button*-1>',*enter*)

Кнопка “Очистити”

bclean = *Button*(*nal_gen*)

bclean.*grid*(*row* = 2 , *column* = 4 , *columnspan* = 3)

bclean.*configure*(*text* = " Очистити ")

bclean.*bind*('<*Button*-1>',*clean*)

Для завершення та збереження елементів на інтерфейсі потрібно скористатися обов’язковою функцією , котра повинна бути використана в самому кінці – *mainloop*():

root.*mainloop*()

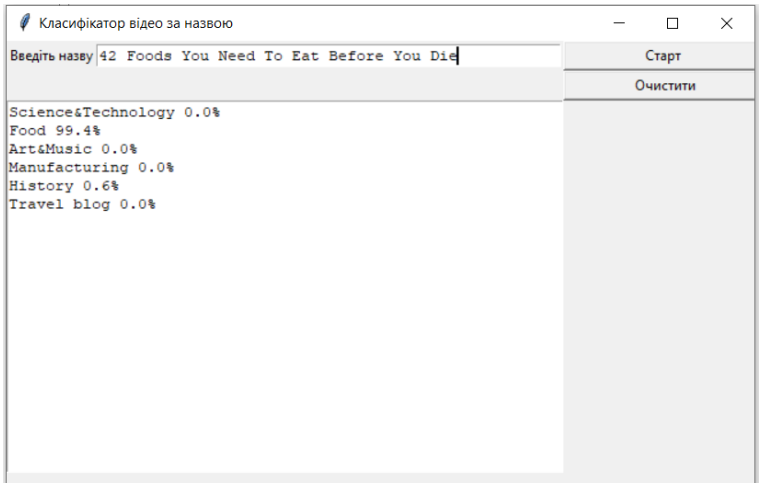
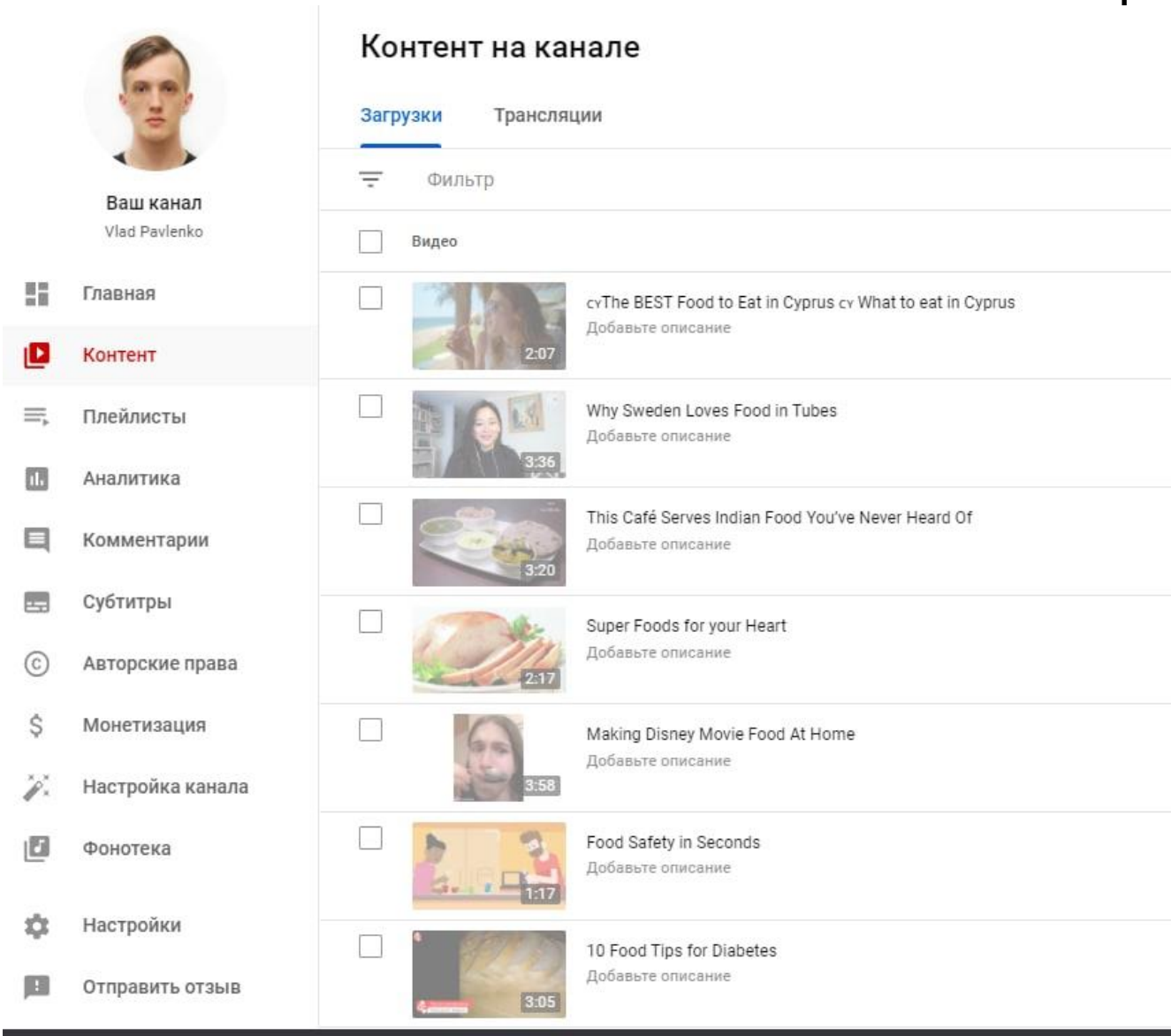
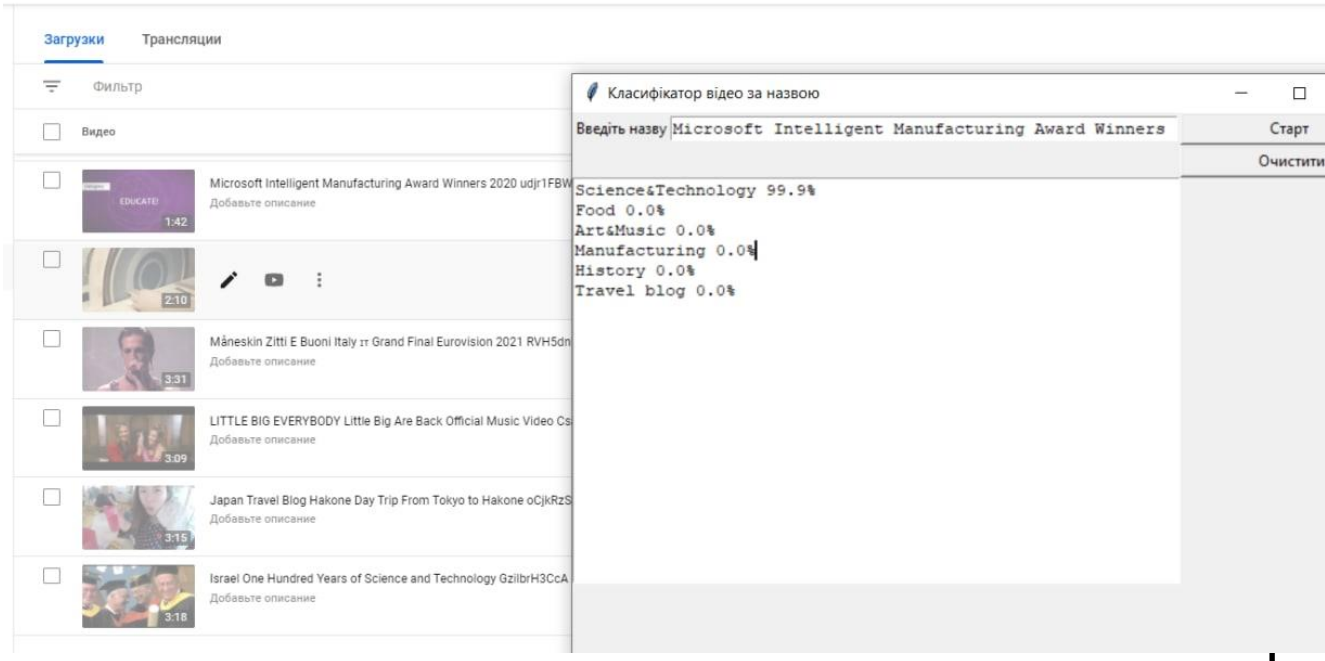
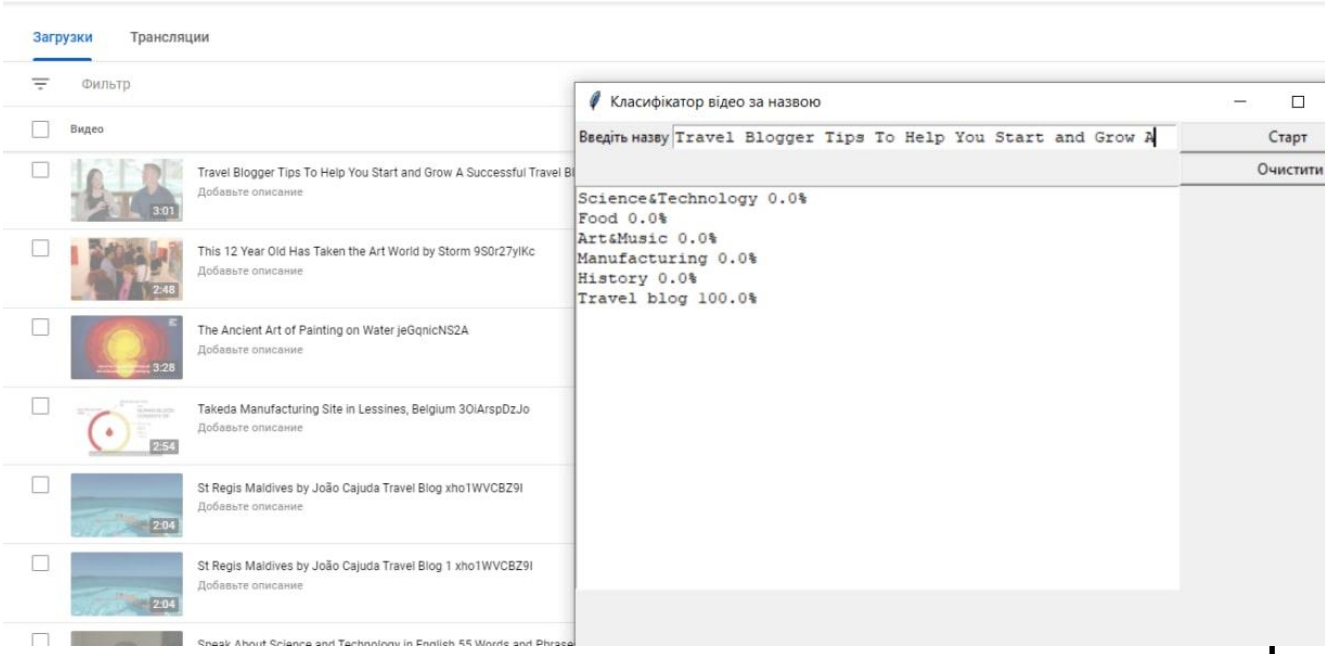
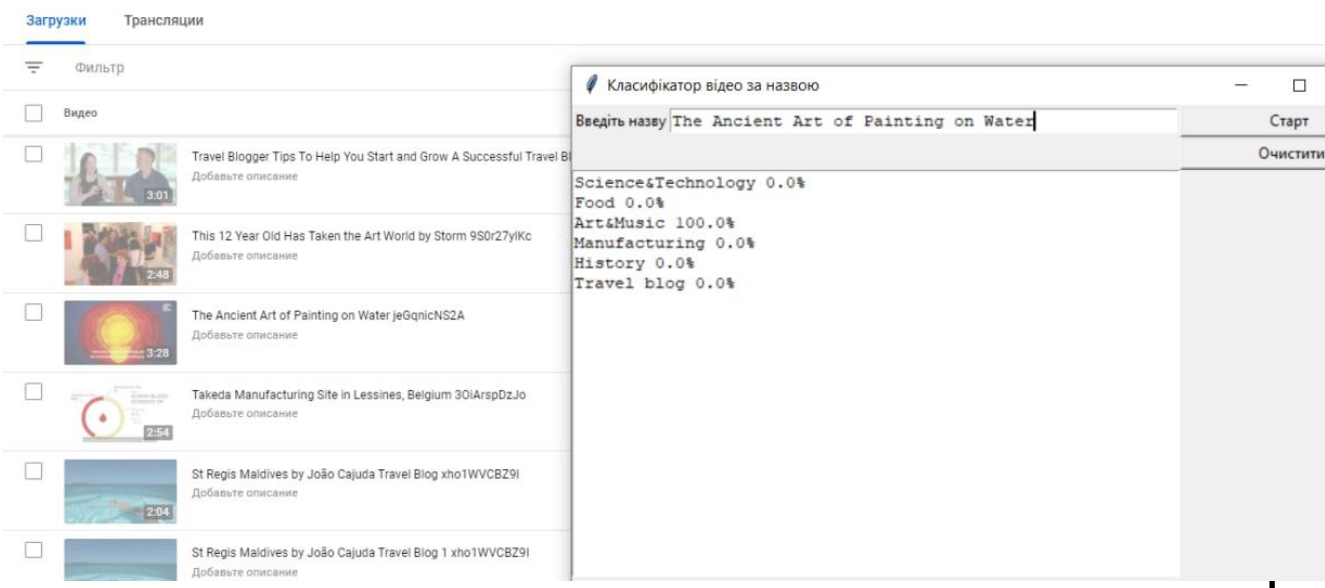


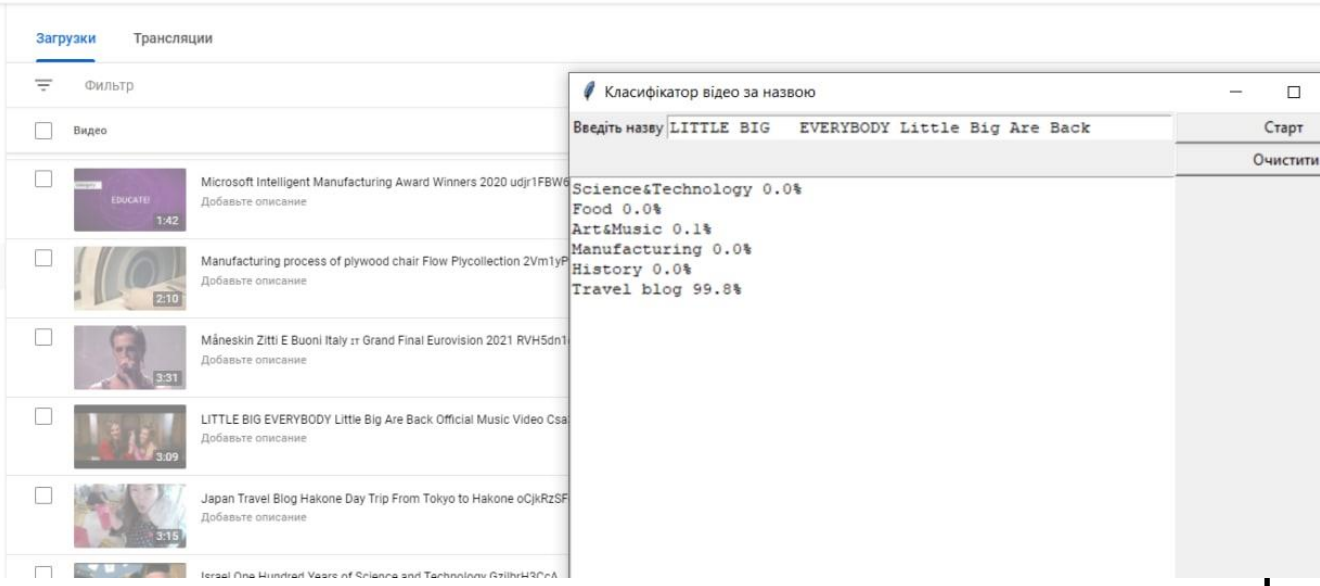
Рис.3.9. Вікно користувацького інтерфейсу

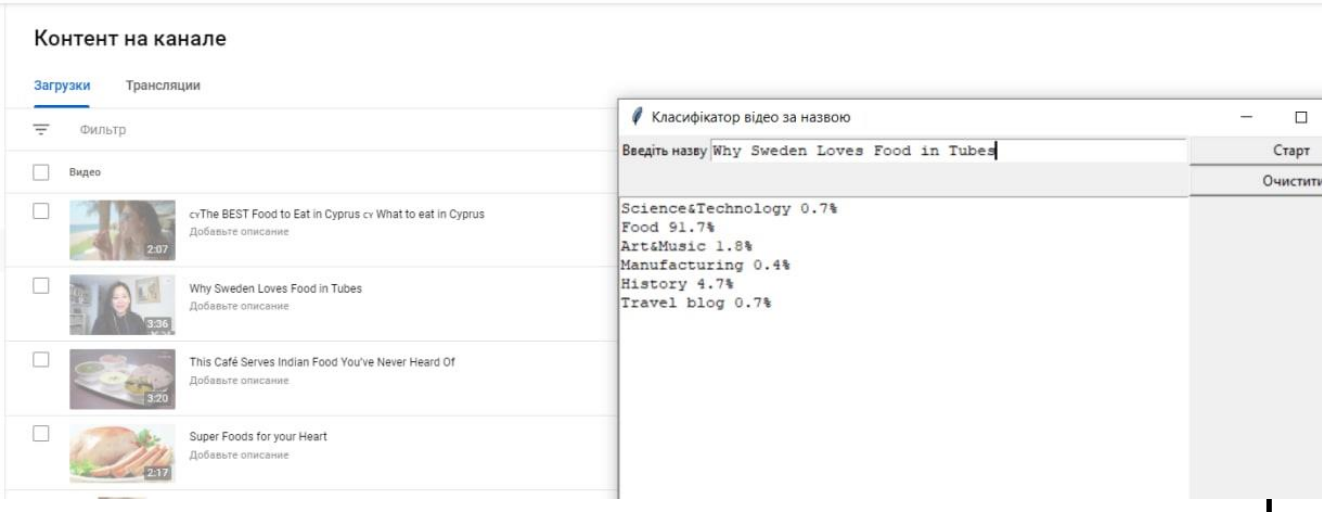


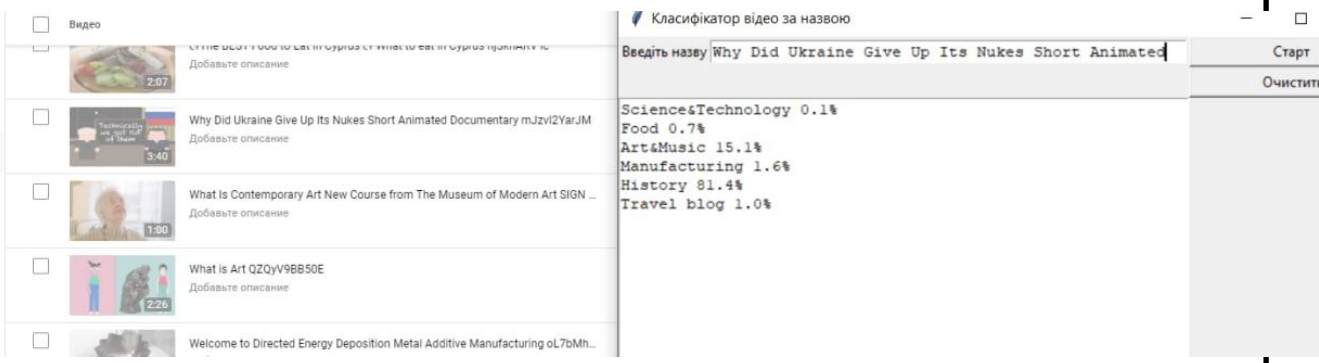

Загрузки
Трансляции
Фильтр
Видео
Microsoft Intelligent Manufacturing Award Winners 2020 udjr1FBW
Добавьте описание
1:42
2:10
Måneskin Zitti E Buoni Italy 🇮🇹 Grand Final Eurovision 2021 RVH5dn
Добавьте описание
3:31
LITTLE BIG EVERYBODY Little Big Are Back Official Music Video Cs
Добавьте описание
3:09
Japan Travel Blog Hakone Day Trip From Tokyo to Hakone oCjkRzS
Добавьте описание
3:15
Israel One Hundred Years of Science and Technology GzilbrH3CcA
Добавьте описание
3:18
Класифікатор відео за назвою
Введіть назву
Microsoft Intelligent Manufacturing Award Winners
Старт
Очистити
Science&Technology 99.9%
Food 0.0%
Art&Music 0.0%
Manufacturing 0.0%
History 0.0%
Travel blog 0.0%

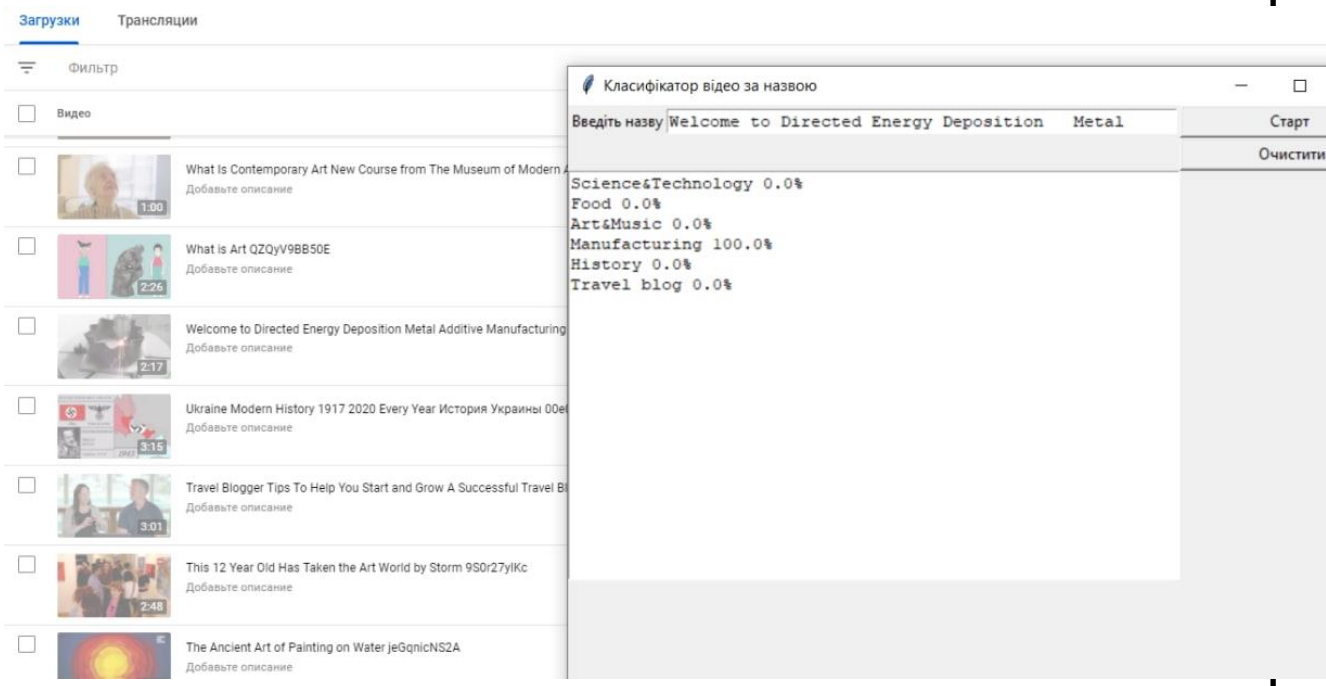

Загрузки
Трансляции
Фильтр
Видео
Travel Blogger Tips To Help You Start and Grow A Successful Travel Bl
Добавьте описание
3:01
This 12 Year Old Has Taken the Art World by Storm 9S0r27ylKc
Добавьте описание
2:48
The Ancient Art of Painting on Water jeGqnicNS2A
Добавьте описание
3:28
Takeda Manufacturing Site in Lessines, Belgium 3OiArspDzJo
Добавьте описание
2:54
St Regis Maldives by João Cajuda Travel Blog xho1WVCBZ9I
Добавьте описание
2:04
St Regis Maldives by João Cajuda Travel Blog 1 xho1WVCBZ9I
Добавьте описание
2:04
Класифікатор відео за назвою
Введіть назву
Travel Blogger Tips To Help You Start and Grow A
Старт
Очистити
Science&Technology 0.0%
Food 0.0%
Art&Music 0.0%
Manufacturing 0.0%
History 0.0%
Travel blog 100.0%


Загрузки
Трансляции
Фильтр
Видео
Travel Blogger Tips To Help You Start and Grow A Successful Travel Bl
Добавьте описание
3:01
This 12 Year Old Has Taken the Art World by Storm 9S0r27ylKc
Добавьте описание
2:48
The Ancient Art of Painting on Water jeGqnicNS2A
Добавьте описание
3:28
Takeda Manufacturing Site in Lessines, Belgium 3OiArspDzJo
Добавьте описание
2:54
St Regis Maldives by João Cajuda Travel Blog xho1WVCBZ9I
Добавьте описание
2:04
St Regis Maldives by João Cajuda Travel Blog 1 xho1WVCBZ9I
Добавьте описание
Класифікатор відео за назвою
Введіть назву The Ancient Art of Painting on Water
Старт
Очистити
Science&Technology 0.0%
Food 0.0%
Art&Music 100.0%
Manufacturing 0.0%
History 0.0%
Travel blog 0.0%


Загрузки
Трансляции
Фильтр
Видео
Microsoft Intelligent Manufacturing Award Winners 2020 udjr1FBW6
Добавьте описание
1:42
Manufacturing process of plywood chair Flow Plycollection 2Vm1yP
Добавьте описание
2:10
Måneskin Zitti E Buoni Italy 1т Grand Final Eurovision 2021 RVH5dn1
Добавьте описание
3:31
LITTLE BIG EVERYBODY Little Big Are Back Official Music Video Csa
Добавьте описание
3:09
Japan Travel Blog Hakone Day Trip From Tokyo to Hakone oCjkRzSF
Добавьте описание
3:15
Класифікатор відео за назвою
Введіть назву LITTLE BIG EVERYBODY Little Big Are Back
Старт
Очистити
Science&Technology 0.0%
Food 0.0%
Art&Music 0.1%
Manufacturing 0.0%
History 0.0%
Travel blog 99.8%


Контент на канале
Загрузки
Трансляции
Фильтр
Видео
cvThe BEST Food to Eat in Cyprus cv What to eat in Cyprus
Добавьте описание
2:07
Why Sweden Loves Food in Tubes
Добавьте описание
3:36
This Café Serves Indian Food You've Never Heard Of
Добавьте описание
3:20
Super Foods for your Heart
Добавьте описание
2:17
Класифікатор відео за назвою
Введіть назву Why Sweden Loves Food in Tubes
Старт
Очистити
Science&Technology 0.7%
Food 91.7%
Art&Music 1.8%
Manufacturing 0.4%
History 4.7%
Travel blog 0.7%





3.7. Тестування розробленої мережі

Було розроблено програмний модуль на мові програмування *Python*. Модуль включає реалізацію двох виконавчих файлів – *main.py* та *classify.py* . У процесі розробки було виконано тестування навченої мережі за допомогою відокремлення частини датасету та використання його у вигляді тестової вибірки. Для побудови графічного зображення результатів тестування була використана бібліотека *Mathplotlib* , однією з основних переваг якої є побудова графіків різноманітних типів та форм .

На рис.3.10. зображено результати тестування у такій формі , що зліва вертикально зображені дійсні категорії елементів , а в нижній частині горизонтально зображені категорії які визначила нейронна мережа.

На клітинках де перетинаються однакові категорії мережа отримала вірну відповіть , на всіх інших – не вірну . Число зображене на кожній клітинкі відповідаю кількості випадків що відповідають значенню кожної клітинки.

Наприклад клітинка на перетині *Travel blog* та *Food*(на першому місці знаходиться категорія з лівого боку) зображено число 10 , тобто до 10 тестувальних елементів з дійсною категорією *Travel blog* нейромережа підібрала хибну категорію *Food*.

3.6. Висновки до розділу

Даний розділ присвячено програмній реалізації проекту.

Окремо виділені питання створення моделі нейронної мережі, її подальше навчання та збереження.

Також було розібрано створення модуля класифікації вхідних текстових даний у вигляді текстової назви відеоматеріалу англійською мовою .

Розібрано процес створення графічного користувацького інтерфейсу.

Протестовано програму на тестовій вибірці з датасету із графічним відображенням результату у вигляді співвідношення вірних та хибних вихідних даних з числовими характеристиками.

ВИСНОВКИ

У дипломному проекті було створено програмний модуль для навчання нейронної мережі та подальшого її використання для класифікації тексту з використанням графічного інтерфейсу користувача. Для вирішення даної задачі було виконано наступне:

1) проаналізовано технології штучних нейронних мереж;

2) розкрито питання щодо переваг та недоліків штучних нейронних мереж;

3) розкрито питання класифікації , топології та розмірності НМ

4) розроблено на основі програмний модуль перетворення вибірки та навчання НМ;

5) реалізовано мовою *Python*3 програмний модуль класифікації тексту;

6) протестовано створену нейронну мережу.

Варто відзначити, що нейронна мережа робить узагальнення автоматично завдяки своїй структурі, а не за допомогою спеціально написаних програм.

Штучні нейронні мережі застосовуються у різноманітних сферах, зв‘язаних з обробкою інформації.

З огляду на те, що проект носить інформативну, а не комерційну направленість, відсутня необхсть в збільшенні розмірів датасету та ускладнення топології мережі.

Так для реалізації мережі було використано проект з вікритим вихідним кодом *TensorFlow*, який дозволяє пришвидшити роботу програми за наявності на пристрої апаратного забеспечення *NVIDIA*.

Практична цінність результатів дослідження полягає у наданні можливості користувачам світового відеохостингу *YouTube* класифікувати власні відеоматеріали , а також пришвидшити пошук потрібних матеріалів всередині екосистеми *YouTube*.

Дана робота потребує подальшої доробки в збільшенні функціональних

можливостей для збільшення цільової користувацької аудиторії.

При незначній зміні в програмному інтерфейсі і коригуванні методик навчання мережі дану програму можна використовувати при вирішенні задач будь-якої класифікації тексту на будь-якій мові котра підтримується *TensorFlow*.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи(проекти) випускників Національного авіаційного університету. Київ : НАУ , 2017. 63 с.

2. ДСТУ 3008-95. Документація. Звітиу сфері науки і техніки. Структура і правила оформлення: видання офіційне , Київ : Держстандарт України , 1995. 38 с.

3. Субботін С. О. НЕЙРОННІ МЕРЕЖІ: ТЕОРІЯ ТА ПРАКТИКА: Навчальний посібник, 2020. -184 с.

4. Заенцев И.В. Нейронные сети: основные модели: учебное пособие для студентов / Заенцев И.В.— Воронеж:ВГУ, 1999. — 76с .

5. *Turing A.M. Computing machinery and intelligence* / *Turing A.M.* // *Mind*. — 1950.— *vol*. 59, №236. — *P*. 433—460.

6. Уоссермен Ф. Нейрокомпьютерная техника: теория и практика / Уоссермен Ф.; пер. с англ. Ю. А. Зуев, В. А. Точенов. — М.:Мир, 1999. — 184с. — *ISBN* 5060040941.

7. Розенблатт Ф. Принципы нейродинамики. Перцептроны и теория механизмов мозга / Ф. Розенблатт. — М.: Мир, 1965. — 175с.

8. *Estebon M.D. Perceptrons*: *An Associative Learning Network* / *M.D. Estebon* // *Virginia Tech*. — 1997.

9. Круглов В. В. Искусственные нейронные сети. Теория и практика / В. В. Круглов, В. В. Борисов. — М.: Горячая линия — Телком, 2002. — 382с. — *ISBN* 5-93517-031-0.

10. *https://ru.wikipedia.org*

11. Тимощук П.В. Штучні нейронні мережі Навчальний посібник/-Львів: Видавництво Львівської Політехніки, 2011. – 444 стор. *ISBN*: 978-617-607-063-4

Додаток А

Лістинг основного програмного модуля

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras import layers
import pandas as pd
from tensorflow.keras import utils
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.layers import LSTM, Dense,
Activation,Dropout,Embedding
from sklearn.preprocessing import LabelEncoder
import numpy as np
from tensorflow.keras.datasets import imdb
#объявляем константы
max_words = 1000
batch_size = 32
epochs = 3
#считываем из CSV
df = pd.read_csv('clean_dataset.csv')
num_classes = 6
X_raw = df['Title'].values
Y_raw = df['Category'].values

#трансформируем текст запросов в матрицы
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_raw)
```

```
x_train = tokenizer.texts_to_matrix(X_raw)

#трансформируем классы
encoder = LabelEncoder()
encoder.fit(Y_raw)

encoded_Y = encoder.transform(Y_raw)
y_train = keras.utils.to_categorical(encoded_Y, num_classes)

#строим модель
model = Sequential()
model.add(Dense(512, input_shape=(max_words,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

model.fit(x_train, y_train,
            batch_size=batch_size,
            epochs=epochs,
            verbose=1)

model.save('classifier.h5')



import tensorflow as tf
```

```
from tensorflow import keras
from tensorflow.keras.models import load_model
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
import pandas as pd
import nltk
from nltk.tokenize import wordpunct_tokenize
from tkinter.filedialog import askopenfilename
from tkinter import *
classes = {"travel
blog":0,"Science&Technology":1,"Food":2,"Art&Music":3,"manufacturing":4,"History
":5}

model = load_model('classifier.h5')

def predict(str_query,numwords):

    tokenizer = Tokenizer(num_words=numwords)
    X_raw_test = [str_query]
    df = pd.read_csv('clean_dataset.csv')
    X_raw = df['Title'].values
    tokenizer.fit_on_texts(X_raw)
      x_test = tokenizer.texts_to_matrix(X_raw_test, mode='binary')


      prediction = model.predict(np.array(x_test))
      prediction_in = prediction[0]
      for name, index in classes.items():
          info = name + " " + str(round(prediction_in[index]*100,1)) + "%" + "\n"
          text_out.insert(float(index) , info)
```

```
def enter(a):
    query = text_in.get(1.0 , END)
    predict(query,1000)

def clean(a):
    text_out.delete(1.0 , END)

root = Tk()
root.title("Класифікатор відео за назвою")
root.geometry("650x500")

fr = Frame(root)
fr.grid()
nal_gen = Frame(fr)
nal_gen.grid(row = 0 , column = 0 , columnspan = 1)

lname = Label(nal_gen)
lname.grid(row = 0 , column = 0)
lname.configure(text = "Введіть назву")

text_in = Text(nal_gen , width = 50 , height = 1 , wrap = WORD)
text_in.grid(row = 0 , column = 1)



bstart = Button(nal_gen)
bstart.grid(row = 0 , column = 4 , columnspan = 3)
bstart.configure(text = "                Старт                ")
bstart.bind('<Button-1>',enter)
```

```
bclean = Button(nal_gen)
bclean.grid(row = 2 , column = 4 , columnspan = 3)
bclean.configure(text = "                Очистити                ")
bclean.bind('<Button-1>',clean)

text_out = Text(nal_gen , width = 60 , height = 20 , wrap = WORD)
text_out.grid(row = 7 , column = 0 ,columnspan = 3 ,  sticky = N)

root.mainloop()
```