Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Семинарска работа по предметот
Дигитално процесирање на слика на тема:

Детекција на лажни потписи со помош на надгледувано учење и наоѓање на сличноста на потписите

Филип Самарџиски, Тамара Стојанова

Содржина

Содржина 2
Вовед 3
Структура на апликацијата 4
Тек на апликацијата 4
Надгледувано учење 4
Препроцесирање на податоците 5
Модели 6
Невронски мрежи 7
Наивен Баесов класификатор 8
Дрво на одлучување 8
Шума од дрва на одлучување 9
Точности на различните модели 9
Заклучок 10
Споредба на сличност меѓу потписи 11
Начини за наоѓање на сличноста на потписите 11
Наоѓање на сличност врз основа на контури 12
Наоѓање на сличност со помош на SSIM 13
Резултати од наоѓање на сличност меѓу два потписи 13
Резултати од наоѓање на сличност меѓу еден и повеќе потписи 14
Проблеми 14
Референци 15

Вовед

Дигиталното процесирање на слика е технологија која се користи за анализирање и манипулација на слики со помош на компјутерски алгоритми. Се применува во различни области како медицина, безбедност, медиуми, итн. Една од областите на интерес на дигиталното процесирање на слика е и детекцијата на лажни потписи.

Потписите се важен начин за идентификација на личности и проверка на автентичност на документи. Со сè поголемата бирократија која вклучува безброј документи во секоја сфера на животот на човекот, како и напредокот на технологијата и појавата на дигитални потписи, сè почест е предизвикот за откривање на лажни потписи кои може да се користат во најразлични видови на криминални активности.

Фалсификацијата на потписи е една од најраспространетите форми на злоупотреба на потписи. Оваа практика се состои во создавање и употреба на лажни потписи со цел да се изигра или злоупотреби идентитетот на некоја личност. Фалсифицирањето на потписи може да се случи и рачно, но и со помош на компјутерски програми и напредни техники за обработка на слика.

При рачно фалсифицирање на потписи, злоупотребувачот се обидува да го имитира потписот на легитимната личност, што е понекогаш тежок процес, особено ако потписот е комплексен или детален. За да го изведе ова, злоупотребувачот го анализира и простудира потписот на жртвата, за потоа да го применува неговиот стил, притисок на пишување, облик на буквите и други карактеристики на потписот.

Со помош на компјутерски програми и техники за обработка на слика, фалсифицирањето на потписи станува постапка која може да се изведе со поголема прецизност и леснотија. Ова вклучува скенирање на вистински потписи и нивно дигитализирање во компјутерски формат. Потоа, се користат напредни алгоритми за реконструкција на потписот или генерирање на нови потписи базирани на вистинскиот потпис. Овие алгоритми може да ги применуваат методи на вештачка интелигенција, како генеративни адверсарни мрежи (GAN), за да се добие што попрецизна и реалистична реплика на потписот.

Фалсификацијата на потписи може да биде извршена со цел претставување во име на друга личност и да се остварат најразни злоупотреби, како на пример, отворање на банкарски сметки, префрлање на средства на сметки, воведување на изменети договори или потпишување на важни документи без дозвола на лицето кое се претставува.

Во овој проект, ние ќе разгледаме методи за детекција на лажни потписи преку користење на Supervised Learning (Надгледувано учење), како и споредба на сличноста на два и повеќе потписи. Проектот ќе вклучува и демо апликација во програмскиот јазик Python, која врз основа на избор на акции од страна на корисникот преку стандарден влез, ќе ги извршува потребните пресметки и методи.

Структура на апликацијата

Апликацијата се состои од 4 .py датотеки, коишто имаат различни намени, како и еден директориум за податоците.
Овие датотеки се следните:

→ *main.py* - оваа датотека е главната извршна датотека на фајлот и корисникот треба да започне со користење на апликацијата преку неа.

→ *preprocessor.py* - оваа датотека се користи за препроцесирање, односно чистење на сликите од шум, како и нивна трансформација за користење на податоците во надгледуваното учење.

→ *supervised_learning.py* - датотеката којашто се користи за самото надгледувано учење и проценка дали потписот е веродостоен или не.

→ *similarities.py* - датотека којашто има неколку методи за наоѓање на сличност помеѓу два потписи, како и на еден потпис со останатите.

Директориумот со податоците се состои од слики кои ние ги добивме од „ICDAR 2009 Signature Verification Competition (SigComp2009)“, каде што можеме да најдеме потписи од различни автори, и тоа од два типа: вистински и лажни. Нив ги одделивме во два директориуми, еден за тренирачкото множество и еден за тестирачкото множество за алгоритмите од надгледуваното учење. Во овие директориуми податоците понатаму ги поделивме во други директориуми според авторите на кои припаѓаат, односно Avtor1, Avtor2 и Avtor3. Секоја слика во овие директориуми е именувана на два начини, односно со две клучни зборчиња-genuine и forged. Ова е важно за алгоритмите за надгледувано учење и ќе биде подетално објаснето подолу во документов.

Тек на апликацијата

На почетокот корисникот треба да ја изврши main.py датотеката и таму тој е запрашан со потписите од кој автор сака да работи програмата, односно како што е претходно споменато, има три опции: Avtor1, Avtor2, Avtor3. По избирање на авторот со внесување на неговото „име“, корисникот треба да избере што сака да прави програмата преку внесување на редниот број на акцијата од понудените акции на екранот. Тие акции се:

1. *Проверка дали еден потпис е веродостоен врз основа на Supervised Learning*
2. *Наоѓање на сличност помеѓу два потписи*
3. *Наоѓање на сличност на потпис со останатите потписи*

Во понатамошниот дел од текстот ќе ги опишеме овие акции подетално.

Надгледувано учење

Supervised Learning, односно надгледувано учење, е техника на машинско учење каде што моделот се тренира со податоци кои веќе се означени со соодветни класи.

Првиот чекор во овој процес на учење е собирање на податоци за тренирање и тестирање, којшто го опишавме во делот Структура на апликацијата. Тоа што преостана да биде дообјаснето е делот за именувањето на сликите. Како што беше кажано, тие во своето име вклучуваат „genuine" и „forged", односно „genuine" значи дека потписот е вистински, додека „forged" значи дека потписот е фалсификуван, односно лажен. Ова е важно заради самата природа на учењето на алгоритмите од надгледувано учење кои ги користиме. Тие се состојат од класификатори коишто според податоците што им се дадени, имаат задача да ги класифицираат во одредени класи, односно во нашиот случај класата 1 значи дека потписот е вистински, додека класата 0 значи дека потписот е лажен.

Препроцесирање на податоците

Во овој процес на собирање на податоците, се итерира низ фолдерот (тренирачки или тест) и се вчитува секоја слика со помош на методот imread() од модулот за дигитално процесирање на слики cv2 кој припаѓа на библиотеката OpenCV. OpenCV (Open Source Computer Vision Library) е отворена библиотека за компјутерска видео-обработка и компјутерска видео-визија. Таа е моќна алатка која обезбедува широк спектар на алгоритми и функции за обработка на слика и видео, и истата се користи во разни области како роботика, вградени системи, интерактивни системи, надзор, медицина, сигурност и многу други. Откако сликите се вчитани, следи можеби најважниот дел од овој процес, а тоа е препроцесирањето на податоците (сликите). Ние тоа го правиме со повикување на методот preprocess од датотеката preprocessor.py.

Во овој метод најпрво се повикува методот fastNlMeansDenoising() за дадената слика.
Овој метод е од cv2 и се користи за намалување и елиминација на шумот во сликата. Методот применува нелинеарен филтер за редукција на шумот, наречен Non-local Means (NLM) филтер. Примарната цел на методот е да го подобри квалитетот на сликата со отстранување на шумот, како што се гаусиев (Gaussian) шум, сол и бибер (Salt and Pepper) шум или други видови шум. Вметнатата слика input_image е влезот на методот, врз кој ќе се изврши намалувањето на шумот.

Потоа на излезната слика denoised со методот cv2.threshold(denoised, 127, 1, cv2.THRESH_BINARY_INV), сите пиксели во сликата чија вредност е поголема од прагот (127) се сетираат на 1 (бела боја), додека сите пиксели со вредност помала или еднаква на прагот се сетираат на 0 (црна боја). Знаменцето cv2.THRESH_BINARY_INV означува дека сакаме инверзна бинаризација, каде што белата боја станува црна, а црната боја станува бела.

Во методот crop(thresh) се применува cv2.findNonZero() за пронаоѓање на ненулевите точки (пиксели) во сликата. Оваа функција враќа листа од координати на ненулевите пиксели во сликата.

Користејќи ги координатите на ненулевите пиксели, методот користи cv2.boundingRect(points) за да го опфати најмалото правоаголно преградување (bounding rectangle) околу тие пиксели. Ова правоаголно преградување ги дефинира координатите (x, y) на горниот лев агол на преградувањето, како и ширината (w) и висината (h) на преградувањето.
На крајот, методот враќа дел од сликата кој се наоѓа внатре во правоаголното преградување, преку употреба на img[y: y + h, x: x + w]. Ова го враќа делот од сликата

што е во границите, што на практично ниво значи дека сликата е „исечена“ за да е во граници каде што нема непотребно голем број на нулеви вредности.

Со flatten_img = cv2.resize(cropped, (40, 10), interpolation=cv2.INTER_AREA).flatten() правиме промена на големината и претворање на сликата во еднодимензионална низа со цел да има конзистентност меѓу влезните податоци во алгоритмите за машинско учење, бидејќи тие примаат на влез податоци од сосема идентичен тип.

Потоа се менува големината на исечената слика со помош на методот cv2.resize(cropped, (400, 100), interpolation=cv2.INTER_AREA). Со columns = np.sum(resized, axis=0) се наоѓа збирот на сите колони во сликите, додека пак со помош на lines = np.sum(resized, axis=1) се наоѓа збир на сите редици во сликите. Со „h, w = cropped.shape“ и „aspect = w / h“ се наоѓа соодносот на сликите. На крај како резултат од методот кој досега го опишувавме - preprocess(input_image), се враќа [*flatten_img, *columns, *lines, aspect]. Овој излез се зема како np (NumPy) низа, а потоа со data = np.reshape(data, (901, 1)) и data = data.flatten() повторно ја менуваме големината и го претвораме излезот во еднодимензионална низа за да се осигураме за конзистентноста на податоците.

Последните 3 практики, односно пресметувањето на збирот на колони, редици и сооднос, значително ја подобруваат точноста на нашите модели за надгледувано учење, давајќи ни до знаење дека само „исчистени“ слики не се доволни за прецизно учење на класификаторите.

Модели

Ние користиме 4 различни видови на надгледувано учење, и тоа:

1. *Невронски мрежи*
2. *Наивен Баесов класификатор*
3. *Дрво на одлучување*
4. *Шума од дрва на одлучување*

Со внесување на редниот број во стандарден влез, корисникот бира каков модел ќе се користи. Сите модели кои ги користиме се од библиотеката scikit-learn во Python и овие модели се целосни, односно не правевме никакво изменување во самите класи, туку користевме готови класи и методи понудени од самата библиотека. Предност е што овие класификатори што ни ги нуди оваа библиотека имаат исти методи за секој тип на класификатор.

Тоа значи дека со само еден метод, кој не е комплициран и нема дупликат код, односно во нашиот случај process_data(classifier, train_data, test_data), можеме да го истренираме дадениот модел со помош на методот fit(train_x, train_y), како и да ги добиеме предвидувањата за тестирачкото множество со методот predict(test_x), којшто ќе ни врати листа од листи од предвидени класи. Потоа, лесно можеме да ја пресметаме точноста на дадениот класификатор со аритметичка средина од точно предвидените податоци. Доколку побараме од моделот да ги предвиди податоците од тренирачкото множество и

ги споредиме точностите што се добиваат при двете предвидувања, можеме да видиме дали кај одреден модел со податоци од одреден автор има overfitting на податоците.

Overfitting се случува кога моделот претерано се прилагодува на тренирачкото множество и губи способност за правилна генерализација на нови податоци. Ова може да биде предизвикано од прекомплексни модели, недоволно тренирање или неконтролирани влијанија на податоците. За да се намали ризикот од overfitting може да се користат различни стратегии, но нема да навлегуваме во тоа.

Откако ќе се испечатат на екран овие информации за моделот, од корисникот се бара да внесе слика, односно потпис по своја желба за моделот да предвиди дали тој е вистински или лажен.

Во прилог, ќе ги опишеме подетално користените техники, односно модели за надгледувано учење.

Невронски мрежи

Невронските мрежи, познати како и deep learning модели, се модели за машинско учење инспирирани од работата на човечкиот мозок. Тие се користат за решавање сложени задачи на препознавање облици, класификација, регресија, генерирање на содржина и многу други.

Главната карактеристика на невронските мрежи е нивната структура составена од вештачки неврони, наречени јазли или неврони, кои се поврзани во мрежа. Овие неврони работат заедно како колектив за обработка на информации и донесување на одлуки. Глобалната структура на мрежата обично се состои од повеќе слоеви, како влезен слој, скриен слој и излезен слој.

Кога се тренираат со податоци, невронските мрежи користат алгоритми како Backpropagation и Feed Forward, за да ги сменат тежините на конекциите помеѓу невроните врз основа на нивните грешки и во насока на поправање, за да можат да научат од закономерностите и зависностите присутни во податоците. Ова им овозможува да изведуваат комплексни пресметки и да прават предвидувања или класификации врз основа на непознати податоци.

Ние како модел за невронска мрежа ја користиме класата MLPClassifier од претходно споменатата библиотека scikit-learn. При креирање на објект од овој модел, треба да се наведе бројот на неврони во скриените слоеви на мрежата и колку поголем е овој број, толку е подлабока мрежата и толку е поверојатно да се научат скриените карактеристики и зависности меѓу податоците, меѓутоа преголем број би значело поголема шанса за overfitting на податоците. Ист е случајот и со максималниот број на епохи кој треба да се наведе. Една епоха претставува едно изминување низ мрежата. Треба да се наведат и видот на активациска функција, ратата на случајното генерирање на почетните вредности на тежините (random_state), ратата на учење на мрежата во секоја итерација која не треба да биде ниту премала, ниту преголема за да има ефективно учење на податоците. Можат да се постават и ред други параметри, меѓутоа ние ќе ги оставиме нив да бидат со default вредности.

По повеќе обиди и тестирања на точноста на класификаторот со различни вредности за параметрите, ние избравме мрежата да има 20 неврони по скриен слој, 150 епохи, активациска функција „ReLu“, рата на учење од 0.001 и рата за случајни тежини 0. Со други вредности за параметрите, во одредени случаи класификаторот не даваше доволно голема точност, а во други случаи настануваше преголем overfitting на податоците.

Наивен Баесов класификатор

Наивниот Баесов класификатор е едноставен, но ефикасен статистички алгоритам за класификација на податоци. Се базира на Баесовата теорема и претпоставката за независност меѓу атрибутите на податоците. Клучната карактеристика на наивниот Баесов класификатор е дека смета дека секој атрибут е независен од сите други атрибути, што треба да знаеме дека не е точно во секој случај.

При класификација, алгоритмот го пресметува веројатностниот модел и го применува принципот на најголемата веројатност за да одреди која класа е највероајтна за дадените влезни атрибути.

Наивниот Баесов класификатор е познат по својата брзина и добри перформанси при просторни и вкупни претпоставки. Меѓутоа, неговата претпоставка за независност може да биде проблематична во некои ситуации, особено кога атрибутите се корелирани.

Класата за овој модел од scikit-learn е GaussianNB, којшто работи со непрекинати податоци, односно во нашиот случај бројчаните вредности што ги опишуваат сликите и се поставени како некои фактори односно критериуми врз основа на кои самиот класификатор ја носи одлуката за класата.

Дрво на одлучување

Дрвото на одлучување е структуриран модел на машинско учење составен од јазли и гранки (ребра), каде што секој јазол претставува одлука или тест врз одреден атрибут, а секоја гранка претставува резултат од тестот.

Главната цел на дрвото на одлучување е да се разделат податоците во различни групи (класи) во зависност од нивните атрибути. За да се постигне тоа, моделот го користи тренирачкото множество за да го изгради дрвото со избирање на најрелевантните атрибути и критериуми за поделба, а за да одреди кои се моментално најдобрите критериуми за поделба прави одредени пресметки, како што е на пример пресметката на информациска придобивка. При класификација, нови податоци се проследуваат низ дрвото на одлучување од коренот до листовите, каде што се донесува категоризирана одлука. Во случај на регресија, дрвото предвидува непрекинати вредности.

Дрвото на одлучување може да биде склоно кон overfitting, особено кога има многу комплексни дрва со голем број на нивоа. Ова може да се надмине со примена на различни техники за подесување на параметрите или со употреба на алгоритми за кратење – поткастрување на дрвото.

Се користи класата DecisionTreeClassifier од scikit-learn и има неколку параметри кои можеме да ги подесуваме. Параметарот criterion во дрвото на одлучување го одредува критериумот кој се користи за мерење на квалитетот на поделбата при изградба на

дрвото. Критериумот го одбира најдобриот атрибут за поделба на податоците во јазлите на дрвото, односно оној од кој добиваме најголема информациска добивка. Двата најчесто користени критериуми се "gini" и "entropy" (ентропија), а ние го избравме „gini“. Друг параметар кој ние го подесивме беше повторно „random_state“, кој во контекстот на дрвото на одлучување е параметар кој го контролира случајното генерирање на почетните вредности при изградбата на дрвото и повторно го поставивме на 0. Останати параметри кои ги оставивме да бидат со default вредности се максимална длабочина на дрвото, максимален број на јазли во него и други.

Шума од дрва на одлучување

Шумата од дрва на одлучување се однесува на група од дрва на одлучување, позната и како случајна шума. Во оваа шума секое дрво се гради со различни случајни подмножества од податоци и атрибути.

Одбирањето на случајни подмножества од податоци и атрибути го намалува ризикот од overfitting и помага во подобрувањето на генерализацискатa способност на моделот. Кога се прави предвидување со шумата од дрва на одлучување, се зема во предвид која класа е најчесто предвидена од сите дрва во шумата и таа класа се зема како крајниот резултат.

RandomForestClassifier е класата која се користи од scikit-learn за овој модел. Има повеќе параметри кои можат да се подесат и тие се-број на дрва во шумата кој ние избравме да биде 100, повторно random_state кој има иста улога како кај единечното дрво на одлучување и кој повторно го ставивме на 0, критериумот за одлука(истиот од дрво на одлучување) кој одлучивме да биде повторно „gini“ и други параметри во кои нема да навлегуваме.

Точности на различните модели

Добивме интересни и разновидни резултати од нашите модели за надгледувано учење Во прилог ќе ги наведеме точностите за различните модели подредени од најголема кон најмала точност, пресметана преку аритметичка средина во однос на различните автори. Ќе се споредува точноста од тестирачкото множество, затоа што врз тренирачкото множество сите модели имаат точност од 100%.

1. ***Шума од дрва на одлучување*** со точност над тестирачкото множество од **0,972222222222467**.

2. ***Дрво на одлучување*** со точност над тестирачкото множество од **0,902777777778**.

3. ***Наивен Баесов класификатор*** со точност над тестирачкото множество од **0,80555555555513**.

4. ***Невронска мрежа*** со точност над тестирачкото множество од **0,77777777778**.

Во прилог дадена е и табела со сите точности над тестирачкото множество од сите модели, поделени по автори.

Точност над тестирачкото множество според модели и автори			
Модел	**Avtor1**	**Avtor2**	**Avtor3**
Невронска мрежа	0.8333333333333334	0.875	0.625
Наивен Баесов класификатор	0.9166666666666666	0.5833333333333334	0.9166666666666666
Дрво на одлучување	0.875	1.0	0.8333333333333334
Шума од дрва на одлучување	0.9583333333333334	1.0	0.9583333333333334

Заклучок

Како што може да се забележи, резултатите се разновидни и за нас се можеби и изненадувачки. Очекуваниот исход според нас беше дека невронската мрежа ќе биде со најголема точност заради нејзиниот начин на работа, односно бидејќи таа расудува податоци слично на човечкиот мозок и е најширокот распространет модел за надгледувано учење и во дигиталното процесирање на сликите, но и пошироко.

Една причина за ваквиот резултат е можеби недоволното препроцесирање на сликите(потписите) пред да се стават во тренирачкото и тестирачкото множество, меѓутоа тоа не го објаснува фактот дека останатите класификатори имаат поголеми точности врз истите слики. Друга причина би можела да биде заради overfitting на тренирачкото множество, којшто може да се разреши со различни вредности на параметрите на класификаторите иако, како што спомнавме погоре, ние пробавме повеќе различни комбинации на вредностите на параметрите и таа што ја ставивме беше најдобрата од нашите испробувања. Овој overfitting може да настане и заради неразновидност на податоците во тренирачкото множество, меѓутоа, повторно ќе кажеме, тоа не објаснува како останатите класификатори имаат поголеми точности. Според нашите споредби на точностите над тренирачкото и над тестирачкото множество, во два од три случаи невронската мрежа имаше overfitting, сметајќи дека разлика од 15% или повеќе во пресметаните точности добиени од тестирачкото и тренирачкото множество е доволен фактор за да се заклучи дека има overfitting.

Наивниот Баесов класификатор има прилично неконзистентни точности помеѓу различните автори и тоа може да се објасни со неговата слаба моќ за расудување, со оглед на тоа што работи врз принцип на независност на карактеристиките на податоците. И покрај тоа, тој има прилично солидни точности за два автори. Точноста за останатиот автор е најлоша од сите модели во испитувањето и без сомневање има overfitting, така што во случајов не може да се препорача употреба на овој модел.

Дрвото на одлучување даде прилично конзистентни и добри резултати во споредба со претходните два модела. Исклучок е точноста за третиот автор која е помала и би можеле да кажеме дека за неговите потписи веројатно настанува overfitting на тренирачкото множество.

Најдобриот и најконзистентен модел во однос на точноста е шумата од дрва на одлучување. Таа има најголеми точности и не покажа ниту една инстанца на overfitting. Тоа најверојатно се должи на случајната распределба на податоци и нивни атрибути при процесот на тренирање како што споменавме погоре, и би можеле да ја препорачаме како одличен избор на модел за донесување одлука во границите на ова поле од нашиот проект.
Сè на сè, можеме да кажеме дека со различни вредности на параметрите, би можеле да добиеме поголеми и поконзистентни точности за моделите, меѓутоа тоа ќе оставиме да се разреши со нашето понатамошно учење и искуство, кое се надеваме дека ќе ни помогне уште повеќе да ја надоградиме и подобриме нашата апликација и нејзината моќ.

Споредба на сличност меѓу потписи

Наоѓањето на сличноста меѓу два или повеќе потписи е важно поради неколку причини:

- **Детекција на лажни потписи** - способноста да се процени сличноста меѓу потписите е од суштинско значење за детекција на лажни потписи. Ако потпишувачот обиде да направи лажен потпис, ќе постојат одредени разлики во неговата структура, стил или облик. Преку споредување на сличноста меѓу потписите, може да се идентификуваат потенцијални лажни потписи и да се превземат соодветни мерки.

- **Идентификација на потпишувачи** - анализата на сличноста помага во идентификацијата на потпишувачите. Секој потпишувач има свои уникатни карактеристики, стил и форма на потписот кои може да се препознаат и издвојат. Преку споредување на сличноста меѓу два или повеќе потписи, може да се утврди дали потписите припаѓаат на ист потпишувач или не.

- **Автоматско процесирање на потписи** - Ако имаме голема колекција на потписи, автоматското процесирање на потписите може значително да ни помогне. Споредувањето на сличноста помага во групирањето на слични потписи или во откривањето на групи потписи кои припаѓаат на ист потпишувач. Ова може да биде корисно во форензички истраги, банкарство, административни процеси и слично.

Во нашиот проект, имаме две акции кои може да ги преземе корисникот доколку сака да ја најде сличноста на некој потпис, едната е за споредба на два потписи меѓу себе, а другата е за споредба на еден потпис со останатите. Сите потписи кои може корисникот да ги избере се од тестирачкото множество на авторот кој ќе го избере на почеток на апликацијата. Важат истите автори коишто се наведени во делот „Тек на апликацијата“.

Начини за наоѓање на сличноста на потписите

Нашата апликација се состои од два начина за наоѓање на сличноста на потписите и тие два начина ќе се користат без разлика дали ќе бараме сличност на два потписи меѓу себе или на еден потпис со останатите. Едниот начин е преку наоѓање на контурите на потписот, а другата е со користење на SSIM-Structural Similarity Index, односно, на македонски - индекс за структурална сличност. Корисникот на апликацијата бира сам кој начин на наоѓање на сличност сака да го искористи.

Наоѓање на сличност врз основа на контури

Наоѓањето на сличност на потписи врз основа на контури може да се направи со постапките кои ќе ги опишеме во прилог, кои се навистина често користени во подрачјето на дигитално процесирање на слики. За секоја слика којашто ќе сакаме да ги најдеме нејзините контури, го повикуваме методот get_countours_of_img(img_name, test_folder).

Овој метод за аргументи ги прима името на сликата и апсолутната патека до тестирачкиот фолдер според даденият автор. Во него, со методот cv2.cvtColor(cv2.imread(os.path.join(test_folder, img_name)), најпрво се спојува патеката од тестирачкиот фолдер со името на сликата, па се вчитува самата слика и најпосле таа се конвертира во RGB формат, од BGR форматот, затоа што тоа е форматот кој го во кој се враќа сликата со методот cv2.imread().

Потоа, со img = cv2.resize(cv2.cvtColor(img.copy(), cv2.COLOR_BGR2GRAY), (100, 100)), сликата img се конвертира во сив простор на бои и се менува нејзината големина во димензии 100x100 пиксели.

Со помош на blur_img = cv2.GaussianBlur(img, (5, 5), 0), на сликата img се применува Гаусово заматување со јадро од 5x5 и стандардна девијација 0. Ова помага да се намали шумот и се избегнуваат прекумерните контури. Без применување на заматување, може да се добијат многу мали и непотребни контури поради шумот или мали детали на сликата. Гаусовото заматување ги отстранува или намалува овие непотребни контури, што резултира во издвојување на почистите и позначајните контури, кои ќе бидат генерално попотребни во овој процес.

Потоа доаѓа на ред следниот чекор: ret1, threshold_img = cv2.threshold(blur_img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU). Овде, преку методот на Otsu се одредува оптималната вредност за прагот (threshold) за бинаризација на сликата blur_img. Сликата се бинаризира, каде пикселите со вредност помала од прагот добиваат вредност 0 (црно), а пикселите со вредност поголема или еднаква на прагот добиваат вредност 255 (бело).

Линијата со код morph_img = cv2.morphologyEx(threshold_img, cv2.MORPH_OPEN, kernel=np.ones((5, 5)), iterations=1) извршува операција на морфолошко отварање (morphological opening) со јадро 5x5 и една итерација, на бинарната слика threshold_img.

Морфолошкото отварање е процес на морфолошка обработка на слика кој се состои од две операции - ерозија и дилатација. При ерозијата, се отстрануваат мали области или тенки линии од објектите на сликата, што резултира во намалување на шумот и отстранување на мали прекини. При дилатацијата, се прошируваат и пополнуваат областите на објектите, што ги поврзува и ги затвора прекините кои постојат.

Со invert_img = 255 - morph_img, сликата morph_img се инвертира, односно белите пиксели се претвораат во црни, а црните во бели.

Потоа, конечно, contours, ret2 = cv2.findContours(invert_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) ги враќа контурите на инвертираната слика invert_img. Контурите се претставуваат како листа од точки кои ги опишуваат самите контури.

Со final_img = cv2.drawContours(np.zeros(invert_img.shape, np.uint8), contours, -1, 255, 1) се исцртуваат контурите. Сликата final_img се користи за визуелизација на контурите, меѓутоа ние не ја користиме во нашата крајна апликација, туку стои само доколку притреба да се прикаже што е она што се добива после сите овие методи.
На крај се враќа contours[0] од методот get_countours_of_img(), односно бидејќи contours е листа од листи, со contours[0] се враќаат контурите за сликата.

Доколку станува збор за наоѓање сличност меѓу две слики, се повикува гореопишаниот метод за соодветните две слики и потоа се применува готовата функцијата cv2.matchShapes(img1_contours, img2_contours, 1, 0) која ги споредува контурите на две слики, img1_contours и img2_contours, и дава мерка на сличноста помеѓу нив. Доколку се користат повеќе од две слики, тогаш мора да се наведе која слика сакаме да ја споредиме со останатите и во циклус таа слика ќе се спореди со останатите.

Параметарот img1_contours се контурите на првата слика, додека img2_contours се контурите на втората слика кои се споредуваат. Вториот параметар со вредност 1 претставува типот на метрика за споредување, каде што вредноста 1 означува користење на метриката "Hu moments" која ние ја користиме во овој случај. Последниот параметар, поставен на 0, означува дека нема влијание на ориентацијата на контурите.

Резултатот на оваа функција, е нумеричка вредност која ја оценува сличноста меѓу контурите на двете слики. Попрецизно кажано, поголеми вредности укажуваат на поголема разлика помеѓу контурите, додека помали вредности укажуваат на поголема сличност помеѓу контурите.

Наоѓање на сличност со помош на SSIM

SSIM е метрика, односно функција за споредба на слики од библиотеката scikit-image, која го оценува структурниот и текстурниот контекст на сликата, вклучувајќи ги вредностите за осветленост и контраст. Тоа е посебно корисно за споредување на слики кои содржат комплексни објекти или текстури.

SSIM прима два аргументи, односно две слики и вредноста што ја враќа SSIM индексот покажува колку потписите се слични помеѓу себе. Што поголема е вредноста на SSIM индексот (максимална вредност е 1), толку поголема е сличноста помеѓу потписите. Низок SSIM индекс (блиску до 0) покажува на различност помеѓу приложените слики.

Овие слики пред да ги предадеме на оваа функција, исто како претходно ние ги вчитуваме во сив простор на бои и ги скалираме во фиксна големина од 100x100, затоа што функцијата прима само исти големини на двете слики кои ги споредува. Ова се прави со истите функции кои ги споменавме претходно, а тоа се: cv2.cvtColor(cv2.imread(os.path.join(test_folder, img1)), cv2.COLOR_BGR2RGB) и cv2.resize(cv2.cvtColor(img1.copy(), cv2.COLOR_RGB2GRAY), (100, 100)).

Резултати од наоѓање на сличност меѓу два потписи

Генерално, можеме да кажеме дека се добиваат прилично неконзинстентни резултати за секоја споредба што може да се направи меѓу два потписи, па затоа не препорачуваме потпирање на оваа акција. Поточно, тоа е затоа што не можеме да знаеме која вредност е добра, односно која вредност симболизира голема сличност помеѓу два потписи, без да

видиме колкави се вредностите на останатите потписи. Единствена прилика во која би препорачале користење на оваа акција е кога немаме доволно податоци, односно доволно потписи, туку само два, па приморани сме да го испробаме овој метод и да донесеме заклучок базиран на несигурност.

Резултати од наоѓање на сличност меѓу еден и повеќе потписи

Наоѓањето на сличностите на еден потпис со останатите е техника која иако повторно не ни дава висока сигурност во наоѓање на вистинската сличност, сепак ни овозможува да направиме споредба на еден потпис со останатите и да видиме каде се рангирани.

Во овој случај, методот со споредба на контурите дава прилично полоши резултати отколку методот со споредба со SSIM. За добри резултати се сметаат оние каде што има јасна граница помеѓу фалсификуваните и вистинските потписи во однос на нивната сличност. Со споредбата на контурите се добиваат многу разновидни подредености на сличности во рангираната листа. Кога споредуваме вистински потпис со останатите (вистински и невистински) прилично многу потписи кои се лажни можат да бидат во близина на врвот според сличност, додека пак потписите кои се вистински може да се случи да бидат при дното. Целосен исклучок е ситуацијата кога се споредуваат потписи од Avtor3 со помош на контури. При овој пристап, добиваме неколку потписи (и вистински и лажни) со сличност 0 (највисока кај споредбата преку контури), додека останатите се со сличност 1.7976931348623157e+308, а тоа е највисоката можна вредност, па со оглед на ова, овој пристап е целосно неконзистентен и непрепорачлив.

SSIM исто така не е целосно точен и стабилен, меѓутоа сепак дава одредена одделеност на вистинските од лажните потписи со одредени исклучоци. Вредностите кои ги добиваме со овој метод се скалираат во проценти за полесно разбирање од страна на корисникот.

Пред скалирањето кај SSIM, вредностите на сличности на потписот genuine-01.png со останатите, над кои се појавуваат генерално вистински потписи се отприлика 0.6 за Avtor1 и Avtor2, додека пак за Avtor3 тие вредности изнесуваат отприлика 0.35. По скалирањето пак, овие вредности се движат од околу 86% па нагоре, иако за Avtor3 има повеќе неконзистентност отколку останатите два автори.

Важно е да се каже дека пробавме различни техники за препроцесирање и при споредбата со контури и при споредбата со SSIM и со претходно наведените практики за препроцесирање се најдобрите и најконзистентни резултати што ги добивавме. При споредба со SSIM може да се каже дека не е користено препроцесирање затоа што единствени операции со сликите во тој пристап е нивно претворање во сив простор на бои и промена на нивната големина со цел да биде иста за сите. Чудно е што каква било примена на друга техника за препроцесирање, вклучувајќи го и отстранувањето на шум, ги правеше резултатите полоши и понеконзистентни при користење на SSIM.

Проблеми

Проблемите кои би ги навеле ние се оние што се веќе споменати во текот на документов, односно најзначајно нешто што може да се заклучи е дека повеќето употреби на сликите

од Avtor3, предизвикаа навистина големи неконзистентности и резултати, особено за точноста на невронската мрежа и за наоѓање на сличност на потписи преку споредба на контури. Тоа може да се должи на фактот дека во фолдерот за Avtor3 има неколку лажни потписи кои се значително различни од останатите (и вистински и лажни), па нивното вклучување да биде показател дека невронската мрежа не е доволно спремна за најразновидни податоци и дека не можеме да се потпреме на наоѓање сличност на потписи преку споредба на контури. Се надеваме дека со понатамошното искуство и знаење ќе можеме да ги решиме и образложиме настанатите проблеми.

Референци

- ICDAR 2009 Signature Verification Competition (SigComp2009)
- SSIM: Structural Similarity Index
- sklearn.neural_network.MLPClassifier
- sklearn.naive_bayes.GaussianNB
- sklearn.tree.DecisionTreeClassifier
- sklearn.ensemble.RandomForestClassifier
- OpenCV-Contours : Getting Started
- https://github.com/topics/signature-detection