Факултет инжењерских наука Универзитета у Крагујевцу

Александар З. Кондић

Аутоматско препознавање текста из игре асоцијација на основу видео записа емисије ТВ квиза "Слагалица"

дипломски рад

Крагујевац, 2019.

Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Рачунарска техника и софтверско инжењерство

Ниво студија: Основне академске студије

Предмет: Заштита података (Технике криптовања)

Број индекса: 575/2015

Александар З. Кондић

Аутоматско препознавање текста из игре асоцијација на основу видео записа емисије ТВ квиза "Слагалица"

дипломски рад

Комисија за преглед и одбрану:

1. **доц. др Владимир М. Миловановић - ментор**
2. _____ Датум одбране:_____
3. _____ Оцена:_____

Универзитет у Крагујевцу
Факултет инжењерских наука

Основне студије: **Рачунарска техника и софтверско инжењерство**

Студијско подручје (усмерење): /

Име и презиме: **Александар Кондић**

Број индекса: 575/2015

ПРИЈАВА ДИПЛОМСКОГ РАДА

Тема рада: Аутоматско препознавање текста из игре асоцијација на основу видео записа емисије ТВ квиза „Слагалица"

Задатак:

Теоријско разматрање процеса оптичког препознавања знакова. Креирање решења које применом оптичког препознавања знакова препознаје текст из игре асоцијација на основу видео записа емисије ТВ квиза „Слагалица". Креирање апликације која на основу интеракције са корисником посредством интерфејса командне линије спроводи имплементирано решење проблема.

Ментор:

Др Владимир Миловановић, доцент

Сагласан, претходни ментор,

Др Мина Васковић Јовановић, доцент

Крагујевац, 21. X 2019.

Садржај

1 Увод **3**
1.1 Оптичко препознавање знакова (OCR) . . . 3
1.1.1 Фаза предобраде . . . 3
1.1.2 Фаза препознавања знакова . . . 5
1.1.3 Фаза пост-обраде . . . 5
1.2 Основни елементи видео датотека . . . 6

2 Главни проблеми имплементације апликативног решења **7**
2.1 Налажење скупа фрејмова који садрже све речи из обе итерације асоцијација . . . 7
2.1.1 Класификација фрејмова улазне видео датотеке . . . 9
2.1.2 Проналажење последњег фрејма обе итерације игре асоцијација на основу класификованих података . . . 14
2.2 Сегментација речи из пронађеног скупа фрејмова и њихово претварање у текстуални облик . . . 15
2.2.1 Класа *EpisodeMasks* . . . 15
2.2.2 Обрада слика изолованих текстуалних поља . . . 20
2.2.3 Превођење обрађених слика изолованих поља у текстуални формат . . . 21
2.3 Имплементација корисничког интерфејса . . . 23
2.3.1 Komanda *urls* . . . 23
2.3.2 Команда *sync* . . . 31
2.3.3 Команда *random* . . . 31
2.3.4 Команда *all* . . . 33
2.3.5 Команда *local* . . . 34

3 Пример употребе **36**

4 Закључак **38**

Литература **39**

Резиме

Циљ овог дипломског рада је развој апликације која из улазних видео датотека емисије „ТВ Слагалица“ помоћу технике оптичког препознавања знакова извлачи речи у оквиру дела квиза под називом „асоцијације.“ Асоцијације се, као и остали делови квиза, играју двапут при чему се мења играч који има право да повуче први потез у игри. Апликација треба да детектује и извуче све речи које су приказане у оквиру обе итерације ове игре. Главни проблеми које решава апликација јесу детектовање последњег фрејма видео датотеке обе итерације асоцијација где су откривене све речи, сегментација поља које садрже речи из напоменутих фрејмова, додатна обрада резултујућих слика поља и њихово прослеђивање специјализованом језгру за *оптичко препознавање знакова* (OCR), чији се излазни подаци складиште у датотеку која представља одређену емисију ТВ Слагалице.

Кључне речи: ТВ слагалица, асоцијације, фрејм, сегментација, OCR.

Abstract

The purpose of this thesis is to develop an application that extracts words from input video files of the show “TV Slagalica” using optical character recognition within a segment of the quiz named “associations.” The associations, like other segments of the quiz, are played twice where the player who has the right to make the first move changes. The application should detect and extract all words that are displayed within both iterations of this game. The main problems that the application solves are the detection of the last video frame of both iterations of associations where all words are revealed, the segmentation of fields containing words from said frames, further processing of the resulting field images and forwarding them to a specialized core for *optical character recognition* (OCR), whose output is stored in a file representing a particular TV Slagalica episode.

Keywords: TV Slagalica, associations, frame, segmentation, OCR.

1 Увод

Пре разматрања проблематике пројектовања апликативног решења задатог проблема који је тема овог дипломског рада биће представљени концепт оптичког препознавања знакова као и основни елементи видео датотеке који се разматрају при решавању задатог проблема.

1.1 Оптичко препознавање знакова (OCR)

Оптичко препознавање знакова (енгл. *Optical Character Recognition (OCR)*) представља превођење слика са штампаним или ручно писаним текстом у секвенцу знакова кодираних у формату који је препознатљив рачунару. OCR се увелико користи као метода уноса података са штампаних формулара као што су пасоши, лични карте, коверте и сличне штампане исправе, у облику који је препознатљив рачунару који може да врши евентуалну обраду улазних података. Такође се широко примењује при скенирању докумената и књига ради њихове дигитализације.

Процес оптичког препознавања знакова се састоји из три главне фазе:

1. Предобрада
2. Препознавање знакова
3. Пост-обрада

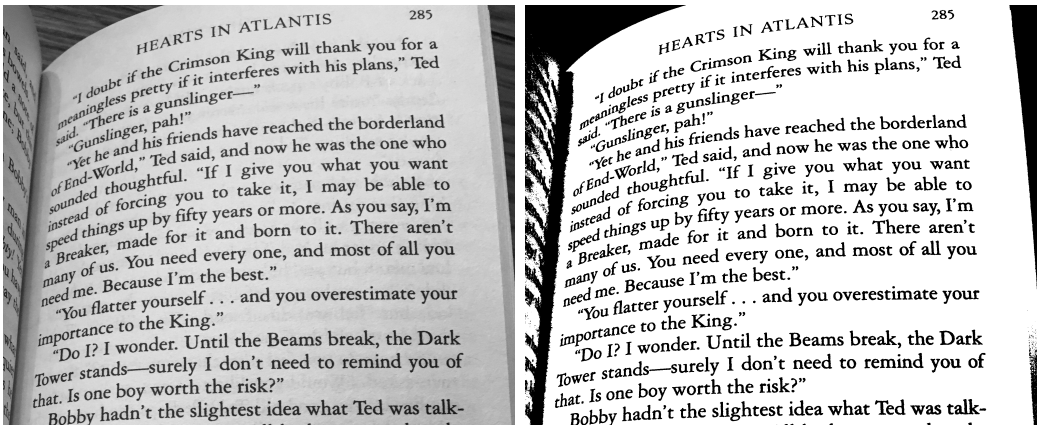
Даље следи опис поменутих фаза процеса оптичког препознавања знакова.

1.1.1 Фаза предобраде

Најпре се врши обрада улазне слике на начин који поспешује успешно препознавање знакова. У овој фази, у зависности од врсте улазне слике могуће је извршити разне технике обраде слика, од којих су најзначајније:(1)

- *De-skew*: Уколико текст на улазној слици није хоризонталан, већ је ротиран за одређени број степени, најпре се врши ротирање слике тако да текст буде хоризонталан. Пошто ротација текста утиче на квалитет препознавања знакова, потребно је извршити ову операцију над улазном сликом како би се повећала успешност препознавања.
- *Бинаризација*: Представља процес превођења слике у боји у црно-белу слику тако да резултујућа слика садржи само две могуће боје: потпуно црну и потпуно белу. Овакав формат слике погодан је за оптичко препознавање знакова зато што језгро за препознавање знакова добија јасну информацију о деловима (пикселима) слике који су окупирани од стране знакова. Односно, процесом бинаризације се одваја текст од позадине на слици.(2)

  Слика у боји се најпре претвори у свој *greyscale* облик, који за сваки пиксел, уместо три вредности које представљају редом вредност црвене, зелене и плаве компоненте боје садржи само једну вредност која представља *количину светлости* у боји, односно *интензитет боје*.



Слика 1: Процес бинаризације улазне слике

После следи процес који се назива *thresholding*, који представља најједноставнији процес сегментације слике тако што се врши избор вредности прага (threshold) за *greyscale* слику где се пиксели у слици претварају у потпуно црне или потпуно беле пикселе у односу на то да ли је вредност њиховог интензитета боје испод или изнад вредности прага.

Сваки пиксел бинаризоване слике може се представити једним битом чија вредност означава црну или белу боју. Пиксели бинаризоване слике не морају се нужно представљати као пиксели црне или беле боје. За њихово графичко представљање може се користити било који пар боја, док је за језго за препознавање знакова од интереса само бинарна вредност сваког појединачног пиксела.

На слици 1 редом су приказани *greyscale* верзија улазне слике и бинаризована улазна слика.

- *Уклањање линија*: Уколико после процеса бинаризације на слици остану одређени артефакти у облику линија и квадратића, примењује се посебан алгоритам који детектује и уклања овакве артефакте са слике. Линије у бинаризованој слици се најчешће јављају зато што су боје слова и улазних поља, оквира и других графичких елемената приближно истих интензитета.

- *Морфолошке трансформације*:(3) Основне морфолошке трансформације бинаризованих слика су ерозија (енгл. *erosion*) и дилатација (енгл. *dilation*). У контексту примене за OCR, ерозија је операција која „стањује“ знакове и успут елиминише шумове у облику тачкастих артефакта малог пречника, док дилатација служи за подебљавање знакова и елиминисање шума који се јавља у облику „рупа“ које се налазе у знаковима. Основне морфолошке трансформације ерозија и дилатација могу се комбиновати да би се добиле изведене морфолошке трансформације као што су *opening* и *closing*.

- *Детектовање речи и редова*: Секвенце знакова на слици се уоквиравају као групе знакова који представљају речи или редове речи. Информације

о оквирима се прослеђују језгру за препознавање знакова, које су корисне при детектовању речи у тексту.

- *Сегментација знакова*: Појединачни знакови се изолују и анализирају. Уколико су на слици неки знакови спојени, врши се њихово раздвајање. Слично, уколико је један знак, због присуства шума на слици, раздвојен у више фрагмената, врши се спајање тих фрагмената у једну целину која представља знак.

1.1.2 Фаза препознавања знакова

Након одговарајуће обраде улазних слика може се наставити са главном фазом у којој се заправо одвија оптичко препознавање знакова. Постоје два главна алгоритма препознавања знакова:(4) препознавање образаца (енгл. *pattern recognition*) и издвајање облика (енгл. *feature extraction*).

Препознавање образаца подразумева матрично упоређивање бинаризоване слике знака са бинаризованим сликама постојећих знакова у оквиру језгра за препознавање знакова. Упоређивање се врши пиксел по пиксел и за сваки постојећи евидентирани знак се рачуна његова сличност улазном знаку. Постојећи знак који је најсличнији улазном знаку сматра се интерпретацијом улазног знака.

Издвајање облика је техника анализе бинаризованог улазног знака која знакове представља на апстрактан начин преко вектора. Улазни знак се анализира и извлаче се информације о отвореним и затвореним линијама, смеровима линија, тачкама пресецања линија и другим карактеристикама облика знака. Добијени параметри складиште се у оквиру вектора који представља улазни знак, који се после пореди са векторима предефинисаних знакова унутар језгра за препознавање знакова и налази се најсличнија репрезентација улазног знака која представља његову интерпретацију.

1.1.3 Фаза пост-обраде

У оквиру фазе препознавања знакова могуће је да језгро за препознавање знакова погреши у интерпретацији улазних знакова. Језгро се најчешће користи за дititализацију једне или више речи на улазној слици, при чему може доћи до погрешне интерпретације једног или више знакова једне или више речи. У оваквим случајевима, излазне речи из језгра за препознавање знакова могу даље да се обрађују и из обраде да резултују исправне речи, уколико је дошло до грешке у интерпретацији знакова.

Један од начина пост-обраде јесте ограничавање скупа излазних речи на скуп који је дефинисан речником, односно базом дозвољених излазних речи. Уколико је реч погрешно интерпретирана и њена интерпретација се не налази у речнику, интерпретација речи може да се замени речју из речника која је њој најближа по броју измена појединачних слова. Уколико постоји више таквих кандидата за замену, узима се онај кандидат коме је придружена највећа вредност вероватноће да се појави у било ком тексту у датом тренутку.

Могуће је да се реч погрешно интерпретира и да се њена интерпретација налази у речнику који дефинише скуп дозвољених излазних речи. Овај проблем се избегава тако што се разматрају вероватноће појављивања парова суседних дозвољених речи. Уколико се у излазним подацима језгра за препознавање знакова налази пар дозвољених речи чија је вероватноћа суседног појављивања изузетно мала, претпоставља се да је дошло до грешке у интерпретацији речи и тај пар се мења са паром који има највећу вероватноћу суседног појављивања и који садржи једну заједничку реч са паром који се мења.

1.2 Основни елементи видео датотека

Свака видео датотека представљена је секвенцом слика које се редом приказују, где тренутна слика смењује претходну одређеном брзином при приказу видео датотеке. Секвенци слика суперпониран је звучни сигнал који се емитује при приказу видео датотеке током процеса периодичног смењивања слика, односно *анимације*.

Свака појединачна слика која је део видео датотеке назива се *фрејмом* (енгл. *frame*). Учестаност смењивања слика при приказу видео датотеке назива се *брзином смењивања фрејмова* (енгл. *frame rate*) и она се мери у броју фрејмова по секунди. Њена мерна јединица има ознаку **fps** (*frames per second*).

Сви фрејмови у оквиру једне видео датотеке су истих димензија, односно имају константну ширину (енгл. *width*) и висину (енгл. *height*), чија се вредност изражава у *пикселима* (*px*; енгл. *pixel*, *picture element*). Ширина и висина фрејмова чине *резолуцију* видео датотеке која се означава као w x h, где су w и h ширина и висина фрејмова, респективно. Један пример видео резолуције је 1024 x 768.

Данас се на разним онлајн платформама за приказ видео датотека (нпр. YouTube) користи алтернативна ознака видео резолуције *hp* где је са h означена релативна висина фрејмова. Примери оваквих ознака су: 144p, 240p, 360p, 480p, 720p (HD), 1080p (Full HD), 1440p (Quad HD), 2160p (Ultra HD / 4K), 4320p (8K). Ове ознаке не означавају нужно и ширину фрејмова, међутим данас видео датотеке прате одређене односе димензија (енгл. *aspect ratio*) које се дефинишу као однос ширине и висине фрејмова видео датотеке. Најчешће коришћени односи димензија су 5:4, 16:9 и 16:10.

2 Главни проблеми имплементације апликативног решења

Проблем који апликација треба да реши јесте извлачење речи из обе итерације асоцијација и њихово складиштење у текстуалну датотеку за сваку улазну видео датотеку емисије ТВ Слагалица. Овај проблем може се разложити на три потпроблема:

1. Налажење скупа фрејмова који садрже све речи из обе итерације асоцијација
2. Сегментација речи из пронађеног скупа фрејмова и њихово претварање у текстуални облик
3. Спровођење корисничког интерфејса који омогућава кориснику преузимање, складиштење и обраду видео датотека емисије ТВ Слагалица као и приступ излазним информацијама апликације

Пре даљег разматрања погодно је напоменути да се изворни код апликације може преузети са *Github* репозиторије преко следеће адресе:
`https://github.com/konda-x1/slagalica-ocr`

Даље ће бити разматрани сваки од наведених потпроблема као и њихова решења.

2.1 Налажење скупа фрејмова који садрже све речи из обе итерације асоцијација

Потребно је из видео датотеке изоловати онолико фрејмова колико је потребно за репрезентацију свих речи из обе итерације игре асоцијација.

Структура игре асоцијација је таква да су на почетку сва текстуална поља сакривена. Играчи се у сваком потезу смењују, а у оквиру једног потеза играч може да открије једно од неоткривених поља означених са А1 до А4, Б1 до Б4, В1 до В4 и Д1 до Д4. Након тога играч може да погађа реч која се налази иза једног од неоткривених поља А, Б, В и Г или неоткривеног поља коначног решења који је од почетка игре до његовог откривања означен са „???“. Уколико играч успешно претпостави реч иза једног од неоткривених поља А, Б, В, и Г, он добија додатну шансу да погоди коначно решење асоцијације. На слици 2 дат је изглед једне итерације игре асоцијација у њеном почетку.

Ова игра је прогресивног типа, што значи да се временом све више поља откривају и остају откривена до краја игре, односно откривена поља се не могу поново сакрити. Захваљујући овој чињеници за сваку игру асоцијације довољно је изоловати тачно један фрејм који садржи сва откривена текстуална поља. Тај фрејм може бити један од последњих фрејмова игре асоцијација. Најлакше и најсигурније је изоловати баш последњи фрејм асоцијације, који ће увек садржати сва откривена текстуална поља.

Слика 2: Једна итерација игре асоцијација у почетку игре

На слици 3 дат је изглед једне итерације игре асоцијација на њеном крају. Сличан садржај има последњи фрејм било које игре асоцијација.

Слика 3: Једна итерација игре асоцијација на крају игре

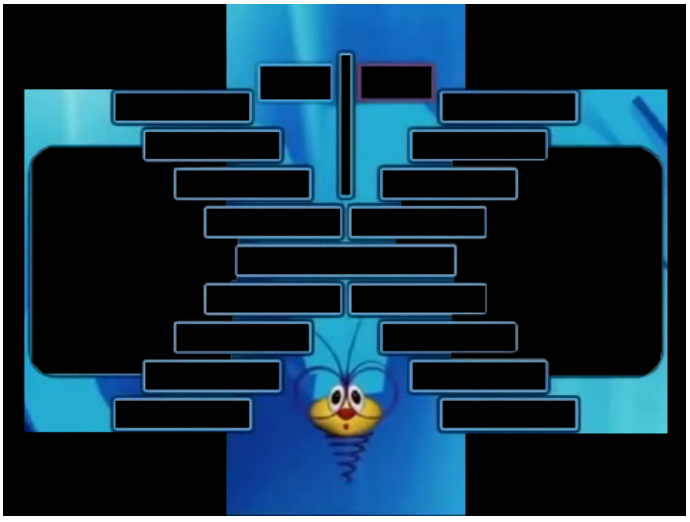
Да би се решио овај проблем, он може даље да се разложи на два потпроблема:

1. Класификација фрејмова улазне видео датотеке у оне које представљају фрејмове игре асоцијација и оне које не представљају фрејмове игре асоцијација
2. Проналажење последњег фрејма обе итерације игре асоцијација на основу класификованих података

Даље следи опис наведених потпроблема као и њихових решења.

2.1.1 Класификација фрејмова улазне видео датотеке

Проблем одређивања припадности фрејма игри асоцијација може се решити тако што се одређени региони фрејма упореде са одговарајућим регионима *референтне слике* која садржи заједничи статички део свих фрејмова игре асоцијација. Слика 4 представља једну такву референтну слику.



Слика 4: Референтна слика игре асоцијација

Може се приметити да дата референтна слика садржи црне пикселе у свим могућим областима где се фрејмови игре асоцијација мењају, а задржани су сви пиксели који су приближно заједнички. Региони референтне слике где би могао да се нађе „РТС“ лого радио телевизије Србије су такође обојени црном бојом.

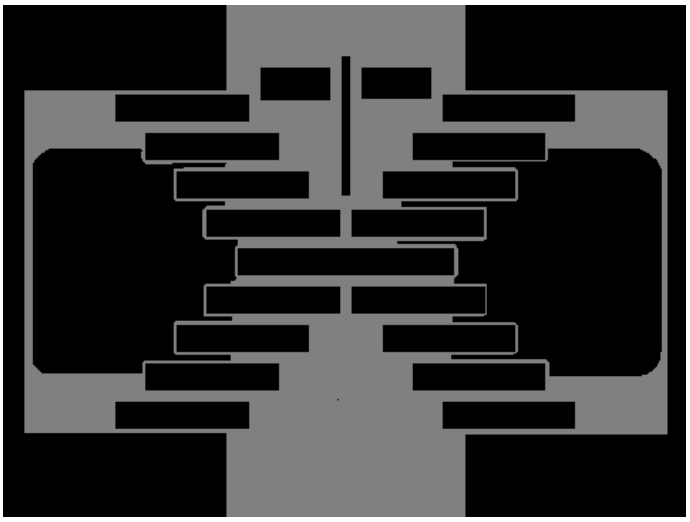
На основу дате рефернтне слике се најпре генерише *маска референтне слике* која садржи вредности нула на местима црних пиксела и вредности јединица на местима пиксела који нису потпуно црни у рефернтној слици. Ради илустрације, уколико се вредности сваког пиксела овако добијене маске помноже одговарајућим вредностима пиксела слике која садржи само једну нијансу сиве боје у свим својим пикселима, као резултат добија се слика која је представљена на слици 5.

Дакле, резултат множења одређене слике маском референтне слике игре асоцијација је такав да резултујућа слика има потпуно црне пикселе на истим позицијама као у референтној слици, док остали пиксели у датој слици остају непромењени.

Емисије које се емитују од 4. новембра 2018. године користе другачију графичку репрезентацију квиза у односу на оне које су се емитовале пре 4. новембра 2018. године. Ова графичка измена обухвата и сегмент игре асоцијација. На слици 6 дат је нови изглед једне итерације игре асоцијација у њеном почетку.

На слици 7 дат је нови изглед једне итерације игре асоцијација на њеном крају.

На слици 8 дата је референтна слика игре асоцијација по новом изгледу.



Слика 5: Резултат множења сиве слике са маском добијеном из референтне слике игре асоцијација

Слика 6: Нови изглед једне итерације игре асоцијација у почетку игре

Уколико се било који фрејм игре асоцијација помножи маском референтне слике, резултујућа слика ће бити скоро идентична референтној слици. Уколико се фрејм који не припада игри асоцијација помножи маском референте слике, резултујућа слика ће се знатно разликовати од референтне слике игре асоцијација.

На основу изложених чињеница може се применити функција која рачуна растојање између две слике, односно степен разлике између две слике и сваком фрејму улазне видео датотеке може се придружити реалан број који представља његово одстојање од референтне слике игре асоцијација. Класификовањем ових вредности у две класе може се одредити који од фрејмова улазне видео датотеке заправо представљају фрејмове игре асоцијација.

Имплементација функције која рачуна растојање између две слике дата је у листингу 1.

Слика 7: Нови изглед једне итерације игре асоцијација на крају игре

Слика 8: Референтна слика игре асоцијација по новом изгледу

```
def imagedist(ima, imb, mask, nonzero_pixelcount):
    assert (ima * mask == ima).all()
    imc = (ima - imb) ** 2
    dist = np.sum(imc) / nonzero_pixelcount
    return dist
```

Листинг 1: Имплементација функције *imagedist* која рачуна степен разлике између две слике

Параметри *ima* и *imb* дате функције представљају слике између којих треба да се израчуна међусобно одстојање. У овој имплементацији се претпоставља да је *ima* заправо референтна слика са којом се пореди слика *imb*, док параметар *mask* садржи маску референтне слике *ima*. Растојање између слика се рачуна као средње квадратно одстојање сваког пиксела слике *imb* од одговарајућих пиксела слике *ima*, изузев потпуно црних пиксела.

Ради убрзавања анализе видео датотека не анализирају се сви фрејмови дате датотеке. Први фрејм који се анализира налази се на половини секвенце фрејмова видео датотеке, пошто је игра асоцијација последњи сегмент емисије који се увек налази у другој половини исте. Такође се не анализира сваки појединачни фрејм од половине до краја секвенце фрејмова, него се анализира сваки

фрејм који је временски удаљен три секунде од претходног.

Вредности одстојања сваког анализираног фрејма од референтне слике игре асоцијација може се графички представити као на слици 9. Са слике јасно могу да се виде две групе фрејмова које припадају једној од две итерације игре асоцијација. Те две групе су раздвојене фрејмовима који припадају кратком сегменту емисије који се увек емитује између две итерације игре асоцијација. Ова информација је од значајне користи за одређивање последњег фрејма обе итерације игре асоцијација.

Вредност временске раздаљине између два узастопна анализирана фрејма треба се одредити као компромис између брзине анализе видео датотека емисије и могућности да се обухвати бар један фрејм који припада кратком сегменту емисије који се емитује између две узастопне итерације игре асоцијација. Експериментално је одређена оптимална вредност од три секунде као временска раздаљина између два узастопна анализирана фрејма видео датотеке.

Изворни код функције која врши одабир фрејмова видео датотеке за анализу и рачунање њихових одстојања од референтне слике игре асоцијација дат је у листингу 2.

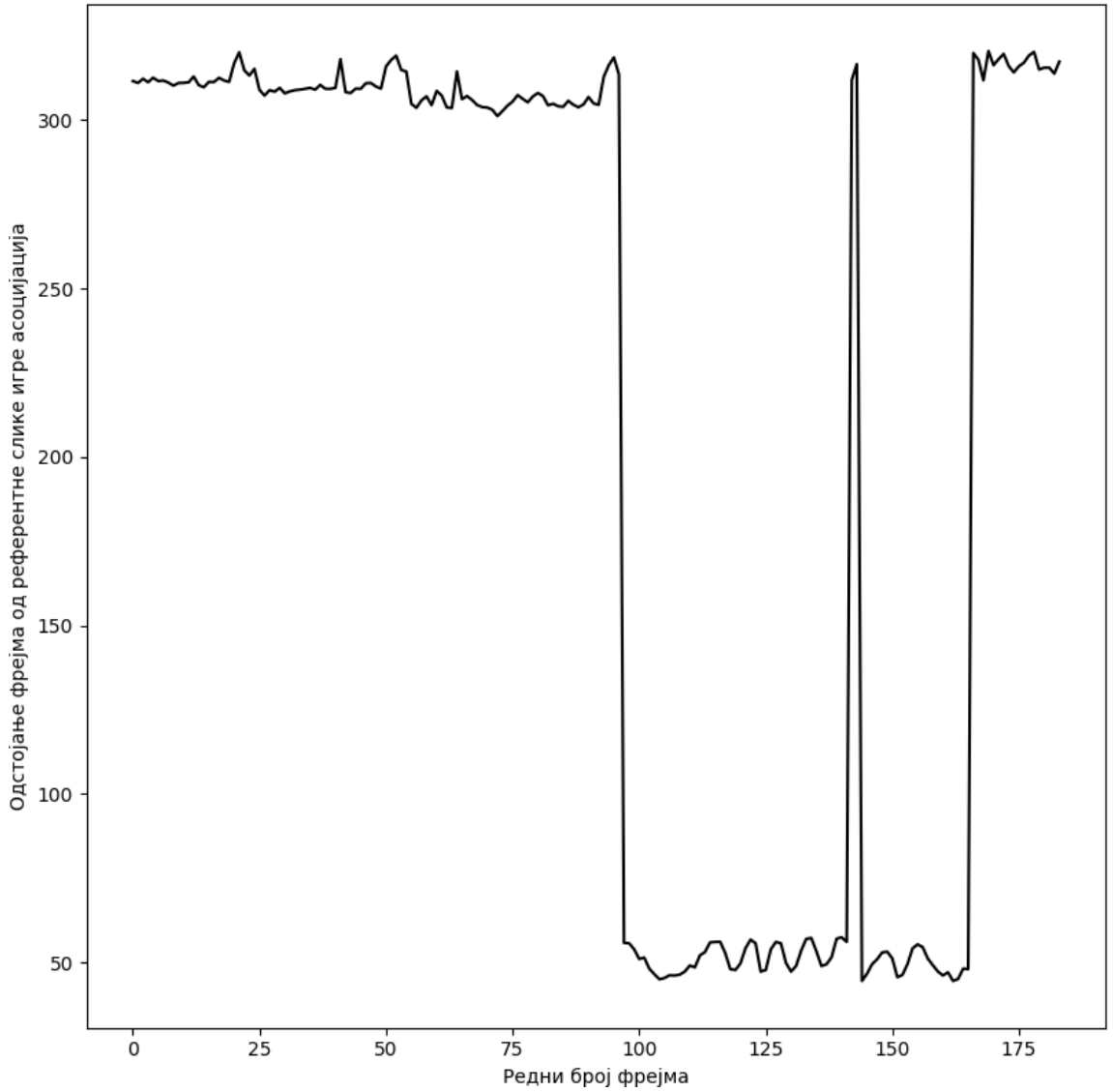
```
def _process_frames(self, cap, total_frames, framerate,
                    mask, delta_secs = 3.0):
    fi = total_frames / 2
    i = int(fi)
    start_frame = int(fi)
    cap.set(cv2.CAP_PROP_POS_FRAMES, start_frame)
    cap.grab()

    prev_i = i
    frames = []
    dists = []
    while(i < total_frames):
        assert i - prev_i >= 0
        for j in range(i - prev_i):
            ret = cap.grab()

        ret, frame = cap.retrieve()
        if not ret:
            break
        frames.append(frame)

        dist = imagedist(frame * mask.detect, mask.orig,
            mask.detect, mask.orig_pixelcount)
        dists.append(dist)

        prev_i = i
```



Слика 9: Графичка репрезентација одстојања сваког анализираног фрејма од референтне слике игре асоцијација

```
            fi += delta_secs * framerate
            i = int(fi)
    return frames, dists
```

Листинг 2: Имплементација функције *_process_frames* која рачуна одстојања фрејмова видео датотеке од референтне слике игре асоцијација

Назив ове функције почиње са карактером „_” зато што је она заправо приватна метода класе *EpisodeProcessor*. Међутим, ова разлика није од значаја за овај дипломски рад тако да ће се у даљем тексту методе ове класе сматрати засебним функцијама, што и није грешка узимајући у обзир природу метода класа у програмском језику Python.

Као улазне податке наведена функција прихвата редом објекат класе

VideoCapture модула *cv2*, укупан број фрејмова у видео датотеци, брзину смењивања фрејмова, маску референтне слике и временску раздаљину између два узастопна анализирана фрејма видео датотеке, чија је подразумевана врендост од три секунде, која је експериментално утврђена као оптимална. Функција као излазне вредности враћа листу анализираних фрејмова, као и листу њихових одстојања од референтне слике игре асоцијација.

С обзиром на то да је значајна разлика између одстојања фрејмова који припадају и који не припадају сегменту емисије о игри асоцијација од референтне слике игре асоцијација, за класификацију фрејмова довољно је одредити вредност прага одстојања испод које се за фрејм видео датотеке сматра да припада сегменту емисије о игри асоцијација. Експериментално је утврђено да је оптимална вредност прага једнака средњој вредности одстојања свих анализираних фрејмова од референтне слике игре асоцијација, која је умањена за 50% вредности њеног одстојања од минималне вредности одстојања фрејмова. Имплементација функције која одређује вредност прага на основу листе вредности одстојања фрејмова дата је у листингу 3.

```
def _get_threshold(self, dists):
    meandist = np.mean(dists)
    mindist = np.min(dists)
    threshold = mindist + (meandist - mindist) * 0.5
    return threshold
```

Листинг 3: Имплементација функције *_get_threshold* која рачуна вредност прага класификације одстојања фрејмова видео датотеке од референтне слике игре асоцијација

2.1.2 Проналажење последњег фрејма обе итерације игре асоцијација на основу класификованих података

За одређивање последњег фрејма за обе итерације игре асоцијација довољно је наћи фрејм који је класификован као део игре асоцијација, после којег се налази фрејм који је класификован као да није део игре асоцијација. Ово се лако постиже итерирањем кроз листу одстојања фрејмова од референтне слике игре асоцијација, узимајући у обзир вредност прага класификације. Пошто индекси у листи одстојања одговарају индексима у листи фрејмова, лако се може доћи до самог фрејма који је последњи за одређену итерацију игре асоцијација.

На слици 9 се може приметити да су фрејмови игре асоцијација знатно ближи крају видео датотеке него њеној средини. Из тог разлога је ефикасније итерирати кроз листу одстојања фрејмова почевши од краја листе. Први фрејм који буде детектован као последњи фрејм одређене итерације игре асоцијација представљаће последњи фрејм друге асоцијације, док ће други такав детектован фрејм представљати последњи фрејм прве асоцијације. После проналаска последњег фрејма прве асоцијације процес итерирања може се зауставити.

Описан алгоритам је имплементиран у оквиру функције чији је изворни код дат у листингу 4.

```
def _get_asoc_frames(self, frames, dists, threshold):
    asocs = []
    prev_dist = 500.0
    count = 0
    for i, dist in reversed(list(enumerate(dists))):
        if dist < threshold and prev_dist >= threshold:
            count += 1
            frame = frames[i]
            asocs.append(frame)
            if count > 1:
                break
        prev_dist = dist
    return list(reversed(asocs))
```

Листинг 4: Имплементација функције *_get_asoc_frames* која враћа последње фрејмове обе итерације игре асоцијација

Листа коју ова функција враћа садржи фрејмове асоцијација у правилном редоследу, тако да је први елемент ове листе последњи фрејм прве асоцијације, док је други елемент ове листе последњи фрејм друге асоцијације.

2.2 Сегментација речи из пронађеног скупа фрејмова и њихово претварање у текстуални облик

Као што је напоменуто, пронађени скуп фрејмова који садржи сва текстуална поља игре асоцијација се састоји из последњих фрејмова појединачних итерација ове игре. Сада је потребно из ових фрејмова изоловати поља која садрже знакове и над њима извршити операцију оптичког препознавања знакова. За изоловање појединачних текстуалних поља из фрејмова игре асоцијација користи се помоћна класа *EpisodeMasks*, која за свако поље садржи координате пара тачака које представљају правоугаоник који обухвата текстуално поље. Након изоловања појединачних текстуалних поља асоцијација врши се процес обраде појединачних слика изолованих текстуалних поља ради поспешивања тачности процеса оптичког препознавања знакова, који следи одмах након њихове обраде.

Најпре ће бити разматрана класа *EpisodeMasks* која служи за изоловање појединачних текстуалних поља, након чега ће бити разматрана обрада резултујућих слика изолованих текстуалних поља и на крају њихово превођење у текстуалне податке који су разумљиви рачунару.

2.2.1 Класа *EpisodeMasks*

У листингу 5 дата је дефиниција класе *EpisodeMasks* са њеним конструктором.

```
class EpisodeMasks(object):
    mask_types = [1, 2]

    def __init__(self, mask_type):
        if mask_type not in self.mask_types:
            raise ValueError("invalid mask type")
        self.mask_type = mask_type
        self._load_masks()
        self._configure()
```

Листинг 5: Дефиниција класе *EpisodeMasks*

Статичка променљива *mask_types* садржи информације о подржаним типовима маски. Подржана су два типа маски које представљају за сада две различите званичне графичке репрезентације игре асоцијација. Маске типа „1“ представљају графичку репрезентацију игре асоцијација из емисија које су емитоване пре 4. новембра 2018. године, док маске типа „2“ представљају нову графичку репрезентацију игре.

При иницијализацији објекта класе позивају се две помоћне приватне методе *_load_masks* и *_configure* које су тема даље анализе ове класе.

Имплементација помоћне приватне методе *_load_masks* дата је у листингу 6.

```
def _load_masks(self):
    def readmask(masknum, maskkey, ret_orig = False):
        orig = cv2.imread(r"masks/mask" + str(masknum) +
            str(maskkey) + ".png")

        mask = orig.copy() if ret_orig else orig
        mask[np.any(mask != [0, 0, 0], axis=-1)] = [1, 1, 1]
        return (orig, mask) if ret_orig else mask

    self.orig, self.detect = readmask(self.mask_type,
        "", True)
    self.fields = dict()
    for a in ["A", "B", "V", "G"]:
        for b in ["1", "2", "3", "4", ""]:
            key = a + b
            self.fields[key] = readmask(self.mask_type, key)
    self.fields["K"] = readmask(self.mask_type, "K")
```

Листинг 6: Имплементација методе *_load_masks* класе *EpisodeMasks*

Наиме, у директоријуму *masks/* за тип маске X налазе се следеће датотеке:

- maskX.png – референтна слика игре асоцијација

- maskXA1.png – маска поља А1
- maskXA2.png – маска поља А2
- maskXA3.png – маска поља А3
- maskXA4.png – маска поља А4
- maskXA.png – маска поља А
- maskXB1.png – маска поља Б1
- maskXB2.png – маска поља Б2
- maskXB3.png – маска поља Б3
- maskXB4.png – маска поља Б4
- maskXB.png – маска поља Б
- maskXV1.png – маска поља В1
- maskXV2.png – маска поља В2
- maskXV3.png – маска поља В3
- maskXV4.png – маска поља В4
- maskXV.png – маска поља В
- maskXG1.png – маска поља Г1
- maskXG2.png – маска поља Г2
- maskXG3.png – маска поља Г3
- maskXG4.png – маска поља Г4
- maskXG.png – маска поља Г
- maskXK.png – маска поља коначног решења асоцијације

Задатак методе *_load_masks* је да учита све наведене датотеке и да их складишти у објекат класе *EpisodeMasks* при његовој иницијализацији. За сваку слику маске се такође врши замена вредности пиксела који нису потпуно црне боје вредностима јединица за све три компоненете боје пиксела. Референтна слика игре асоцијација се складишти у пољу *orig* објекта, док се њена маска складишти у пољу *detect* објекта. Маске појединачних поља се чувају у оквиру поља *fields* који је типа речника са кључевима који означавају појединачна поља за која се складиште маске (нпр. *A1*, *A2*, *A*, *K*).

Након успешног учитавања маски из одговарајућих датотека следи конфигурација објекта класе *EpisodeMasks* преко методе *_configure* чија је имплементација дата у листингу 7.

```
def _set_orig_pixelcount(self):
    self.orig_pixelcount = np.count_nonzero(self.detect)//3

def _configure(self):
    self._set_orig_pixelcount()
    crops_filename = r"masks/mask" + str(self.mask_type) +
        "crops.npy"
    try:
        self.field_crops = np.load(crops_filename,
            allow_pickle=True).item()
    except FileNotFoundError:
        self.field_crops = dict()
        for key, msk in self.fields.items():
            self.field_crops[key]= get_autocrop_coordinates(
                msk, [0, 0, 0]
            )
        try:
            np.save(crops_filename, self.field_crops,
                allow_pickle=True)
        except OSError:
            print("Failed to save field crops for \
                mask_type = %d." % self.mask_type)
```

Листинг 7: Имплементација методе *_configure* класе *EpisodeMasks*

Задатак ове методе је да одреди број пиксела који нису потпуно црне боје у оквиру слике која је садржана у пољу *orig* објекта, као и да за маске свих појединачних поља асоцијације одреди парове координата правоугаоника који садрже одговарајућа текстуална поља асоцијације. За проналажење парова координата над сваком маском се позива помоћна функција *get_autocrop_coordinates* која одређује тражене парове координата улазне слике, узимајући додатни параметар који означава боју позадине слике која може да се одстрани да би се изоловао сам садржај слике. Имплементација помоћне функције *get_autocrop_coordinates* дата је у листингу 8.

```
def get_autocrop_coordinates(img, background = None):
    if type(background) == type(None):
        background = img[0][0]
    w = img.shape[1]
    h = img.shape[0]
    min_x, min_y = w-1, h-1
    max_x, max_y = 0, 0
    for x in range(img.shape[1]):
        for y in range(img.shape[0]):
            if any(img[y][x] != background):
                if x < min_x:
```

```
                min_x = x
            if x > max_x:
                max_x = x
            if y < min_y:
                min_y = y
            if y > max_y:
                max_y = y
    return min_y, max_y, min_x, max_x
```

Листинг 8: Имплементација помоћне функције *get_autoctop_coordinates*

Процес одређивања парова координата који обухватају јединичне пикселе сваке појединачне маске захтева значајно време за извршавање, па се резултати овакве обраде чувају у датотеци *masks/maskXcrops.npy*, где је са X означен тип маске. Уколико ова датотека постоји у директоријуму *masks/*, она се најпре учитава и тиме се прескаче процес проналажења парова координата правоугаоника за сваку појединачну маску.

Вредности парова координата за сваку маску поља игре асоцијација складиште се у оквиру поља *field_crops* који има исту структуру као поље *fields* само што се уместо маски чувају вредности парова координата поља.

Уколико су фрејмови видео датотеке другачијих димензија од димензија учитаних масака, маске треба да промене своје димензије тако да буду исте као димензије фрејмова са којима се упоређују. За ову сврху постоје методе *resize* и *resized_copy* чије су имплементације дате у листингу 9.

```
def resize(self, framewidth, frameheight):
    Rx = framewidth / self.framewidth
    Ry = frameheight / self.frameheight

    # Resize masks
    self.orig = cv2.resize(self.orig,
        (framewidth, frameheight))
    self.detect = cv2.resize(self.detect,
        (framewidth, frameheight),
        interpolation=cv2.INTER_NEAREST)
    for key, msk in self.fields.items():
        self.fields[key] = cv2.resize(msk,
            (framewidth, frameheight),
            interpolation=cv2.INTER_NEAREST)

    # Scale crops
    for key, val in self.field_crops.items():
        self.field_crops[key] = (round(Ry * val[0]),
            round(Ry * val[1]), round(Rx * val[2]),
            round(Rx * val[3]))
```

```
def resized_copy(self, framewidth, frameheight):
    copy = EpisodeMasks(self.mask_type)
    if framewidth != self.framewidth or
        frameheight !=self.frameheight:
        copy.resize(framewidth, frameheight)
    return copy
```

Листинг 9: Имплементација метода *resize* и *resized_copy* класе *EpisodeMasks*

Коначно, класа *EpisodeMasks* садржи методу *cropped_field* која служи за изоловање одређеног текстуалног поља фрејма игре асоцијација. Имплементација ове методе дата је у листингу 10.

```
def cropped_field(self, img, field_key):
    if field_key not in self.fields:
        raise ValueError("invalid field key")
    min_y, max_y, min_x, max_x = self.field_crops[field_key]
    return crop(img, min_y, max_y, min_x, max_x)
```

Листинг 10: Имплементација методе *cropped_field* класе *EpisodeMasks*

Као улазне параметре ова метода прихвата слику/фрејм и идентификациони кључ поља који треба да се изолује из слике. За изоловање одређеног поља из слике ова метода се служи помоћном функцијом *crop*, чија је имплементација дата у листингу 11.

```
def crop(img, min_y, max_y, min_x, max_x):
    return img[min_y : max_y + 1, min_x : max_x + 1].copy()
```

Листинг 11: Имплементација помоћне функције *crop*

2.2.2 Обрада слика изолованих текстуалних поља

Саме изоловане слике текстуалних поља игре асоцијација, када би се проследиле језгру за оптичко препознавање знакова у свом изворном облику, не би произвеле задовољавајуће резултате препознавања знакова. Из тог разлога је потребна додатна обрада тих слика. Процес обраде слика изолованих текстуалних поља биће описан кроз један пример.

Нека је изоловано поље асоцијације које садржи реч „СЕЋАЊЕ“ које је дато на слици 10.

Најпре се слика преводи у њену *greyscale* форму и на тај начин припрема за бинаризацију. *Greyscale* верзија изворне слике представљена је сликом 11.

Онда се извршава скалирање слике преко којег се слика увећава пет пута у обе димензије. Након тога се извршава *thresholding* преко којег се постиже бинаризована слика поља асоцијације. Избор вредности прага за *thresholding* врши

Слика 10: Изоловано поље асоцијације у изворном облику

Слика 11: *Greyscale* слика изолованог поља асоцијације

се применом *Otsu-ове методе*, која аутоматски класификује вредности пиксела у *greyscale* слици у предње и позадинске. Резултат операција скалирања и *threshold*-овања приказан је на слици 12.

Слика 12: Бинаризована слика изолованог поља асоцијације

На крају се, ради смањивања степена повезаности суседних знакова на слици, врши операција ерозије, чији је резултат приказан на слици 13.

Процес обраде слике појединачних изолованих поља обухваћен је у оквиру помоћне функције *process_image*, чија је имплементација дата у листингу 12.

```
def process_image(img):
    greyscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    scale = cv2.resize(greyscale, (0, 0), None, 5, 5,
        cv2.INTER_LANCZOS4)
    res, threshold = cv2.threshold(scale, 0, 255,
        cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    kernel = np.ones((5,5), np.uint8)
    erosion = cv2.erode(threshold, kernel, iterations = 1)
    return erosion
```

Листинг 12: Имплементација помоћне функције *process_image*

2.2.3 Превођење обрађених слика изолованих поља у текстуални формат

Python модул *pytesseract* библиотеке *Tesseract* садржи функцију *image_to_string* која као улазне податке прихвата слику и конфигурационе параметре а на излазу даје резултат препознавања знакова из улазне слике.

За потребе препознавања текста из поља игре асоцијација користи се помоћна функција *im2str* која уз улазну слику прослеђује погодне конфигурационе па-

Слика 13: Коначна обрађена слика изолованог поља асоцијације

раметре функцији *image_to_string*. Имплементација помоћне функције *im2str* дата је у листингу 13.

```
def im2str(img):
    return pytesseract.image_to_string(img,
        config="--psm 7", lang="srp")
```

Листинг 13: Имплементација помоћне функције *im2str*

Tesseract-у се прослеђују конфигурациони параметри *lang="srp"* који назначава да је текст на слици написан на српском језику и *config="--psm 7"* који наговештава да се текст на улазној слици састоји из једног реда.

Класа *EpisodeProcessor* садржи приватну методу *_process_asoc_frames* у оквиру које је садржан комплетан процес сегментације поља игре асоцијација и пропознавања знакова у оквиру изолованих поља. Имплементација поменуте методе дата је у листингу 14.

```
def _process_asoc_frames(self, asocs, mask):
    words = {"1":{}, "2":{}}
    images = {"1":{}, "2":{}}
    for mask_key in mask.fields:
        for i, asoc in enumerate(asocs):
            cropped = mask.cropped_field(asoc, mask_key)
            processed = process_image(cropped)
            images[str(i + 1)][mask_key] = processed
            words[str(i + 1)][mask_key] = im2str(processed)
                .upper()
    return images, words
```

Листинг 14: Имплементација методе *process_asoc_frames* класе *EpisodeProcessor*

Као улазне податке ова метода прихвата листу последњих фрејмова обе итерације игре асоцијација и објекат класе *EpisodeMasks* преко којег се врши изоловање појединачних поља из фрејмова игре асоцијација.

За свако појединачно поље игре асоцијација врши се његово изоловање из датог фрејма, након чега се врши његова обрада позивом помоћне функције *process_image* и коначно се врши препознавање знакова из обрађене слике изолованог поља позивом функције *im2str*.

Као излазне податке ова метода враћа речнике обрађених слика и препознатих речи свих изолованих поља фрејмова обе итерације игре асоцијација.

2.3 Имплементација корисничког интерфејса

Кориснички интерфејс апликације је имплементиран на такав начин да се интеракција између корисника и апликације одвија преко командне линије. Кориснички интерфејс имплементиран је у командном језику *Bash* (*Bourne Again SHell*).

Интеракција између корисника и апликације реализује се тако што корисник позива *Bash* скрипту која је садржана у оквиру датотеке *run.sh*. Као параметре овој скрипти корисник задаје једну од следећих команди, заједно са додатним параметрима дате команде:

- urls [URL] [URL...] – скрипта врши преузимање и анализу емисија ТВ Слагалице на основу једне или више адреса које су прослеђени комнанди
- sync – скрипта врши преузимање листе URL-ова свих емисија ТВ Слагалице. Коришћење дате команде мора претходити коришћењу команди all и random
- random [N] – скрипта насумично бира N URL-ова из локалне листе преузетих URL-ова преко sync команде и врши преузимање и анализу одговарајућих емисија
- all – скрипта врши преузимање и анализу свих емисија ТВ Слагалице на основу локалне листе преузетих адреса преко команде sync
- local [filename] [filename...] – скрипта врши анализу локално преузетих видео датотека емисије ТВ Слагалица које су одређене путањама датотека прослеђених као параметри команде

Даље следи опис имплементације сваке од наведених команди скрипте *run.sh*.

2.3.1 Komanda *urls*

Најпре ће бити разматрана имплементација команде *urls*, која покрива највећи део функционалности скрипте *run.sh*. Делови функционалности ове команде апстраховани су у засебне функције које користе и друге команде скрипте. Имплементација команде *urls* у оквиру скрипте *run.sh* дата је у листингу 15.

```
cmd=$1
if [ "$cmd" == "urls" ]
then
    if [ $# -lt 2 ]
    then
        >&2 echo "error: no URLs supplied"
        exit -2
```

```
    fi
    via_urls "${@:2}"
fi
```

Листинг 15: Имплементација команде *urls* скрипте *run.sh*

Први аргумент који је прослеђен скрипти се складишти у оквиру промењиве *cmd*. Уколико је вредност те промењиве једнака ниски „urls“ скрипта препознаје да јој је задата команда *urls*. Након тога се проверава број аргумената прослеђених скрипти. Уколико је тај број мањи од два, скрипта закључује да корисник није проследио ниједан URL емисије, након чега се штампа порука грешке и скрипта престаје са радом, враћајући вредност повратног кода -2.

Уколико је корисник скрипту снабдео једном или више адреса ка емисијама ТВ Слагалице, те адресе се прослеђују функцији *via_urls* чија је имплементација дата у листингу 16.

```
via_urls ()
{
    youtube-dl -f"bestvideo[height<=720]/best[height<=720]"\
        -o "downloads/%(id)s" --write-info-json\
        --ignore-errors "$@"
    filenames=`python3 url_video_ids.py "downloads/" "$@"`
    localvids=''
    if ! [ -z "$filenames" ]
    then
        newfilenames=`python3 info_rename.py $filenames`
        if ! [ -z "$newfilenames" ]
        then
            mkdir -p local
            localvids=`mv -v $newfilenames local/ |\
                sed 's/.*->\'\''\(.*\)'\''/\1/'`
            mvjsons=''; for i in $newfilenames
                do mvjsons="$mvjsons $i.info.json"
            done
            mv $mvjsons local/
        fi
    fi
    via_filenames $localvids
}
```

Листинг 16: Имплементација функције *via_urls* скрипте *run.sh*

Најпре се позива програм *youtube-dl* који врши преузимање видео датотека на основу прослеђених URL-ова програму. Параметри који су прослеђени програму конфигуришу начин његовог рада на следећи начин:

- За сваки прослеђени URL врши се преузимање најквалитетније видео датотеке до и укључујући резолуцију 720p видео датотеке. Најпре се захтева преузимање видео датотеке која у себи не садржи звук. Уколико таква видео датотека не постоји, онда се захтева видео датотека која садржи звук.
- Преузете видео датотеке се складиште у директоријуму *downloads/*, док име видео датотеке представља њен *id*, односно њену јединствену идентификациону ниску на YouTube платформи. Имена преузетих видео датотека не садрже екстензију датотеке у називу.
- Поред преузимања видео датотеке такође се врши и преузимање одговарајуће *.info.json* датотеке која је именована на идентичан начин као њена придружена видео датотека, изузевши екстензију *.info.json* у њеном називу. Ова датотека садржи информације придружене видео датотеке са YouTube платформе попут наслова, описа, URL-а, id-а и друге информације видео датотеке које су доступне на YouTube платформи.

Након тога се врши позив помоћне скрипте написане у програмском језику Python која је складиштена у оквиру датотеке *url_video_ids.py*. Главни задатак ове скрипте је да извуче информације о *id*-у сваке видео датотеке на основу њеног URL-а. Резултујућим подацима додаје се префикс, који је у овом случају представљен ниском „downloads/.“ Изворни код поменуте помоћне скрипте дат је у листингу 17.

```
import sys
import urllib.parse as urlparse
from urllib.parse import parse_qs

prefix = sys.argv[1]
urls = sys.argv[2:]
ids = []
for url in urls:
    parsed = parse_qs(urlparse.urlparse(url).query)['v'][0]
    ids.append(prefix + parsed)
print(*ids)
sys.exit(0)
```

Листинг 17: Имплементација скрипте *url_video_ids.py*

Следећи корак функције *via_urls* представља позивање помоћне Python скрипте *info_rename.py*. Главна улога ове помоћне скрипте је да за задате путање видео датотека изврши њихово копирање и преименовање у формату *cABCeDEa*, где је са *ABC* означен редни број циклуса емисије, а са *DE* означен редни број емисије у оквиру тог циклуса.

До вредности *ABC* и *DE* помоћна скрипта долази посредством анализе придружене *.info.json* датотеке. Наиме, свака емисија ТВ Слагалице која је отпремљена на *YouTube* платформи садржи број циклуса и број емисије у оквиру тог циклуса

у опису видео датотеке. На сличан начин се утврђује датум емитовања одређене емисије ТВ слагалице – информација о датуму је садржана у оквиру наслова сваке отпремљене видео датотеке на *YouTube* платформи. Ова информација се не подудара увек са датумом отпреме видео датотеке на *YouTube* платформи, те се до тачне информације о датуму емитовања емисије може доћи искључиво преко њеног наслова на *YouTube* платформи.

Као повратне вредности помоћна скрипта враћа имена нових датотека, с тим што се прескаче обрада датотека емисија за које је утврђено да су раније већ биле анализиране у оквиру процеса препознавања текста у пољима игре асоцијација. За редни број *DE* емисије циклуса *ABC* може се утврдити да ли је она већ раније била анализирана провером постојања датотеке *out/cABCeDEa.txt*. Наведена датотека представља излазну датотеку апликације која садржи текстуалне вредности свих поља обе итерације игре асоцијација.

Поред копирања видео датотека у нове видео датотеке са називима у формату *sABCeDEa* такође се креирају одговарајуће *sABCeDEa.info.json* датотеке које у себи садрже исправан датум емитовања емисије, број циклуса, број емисије у оквиру циклуса, као и URL емисије на *YouTube* платформи.

Изворни код помоћне скрипте *info_ rename.py* дат је у листингу 18.

```
import os
import sys
import json
import re
import random
import shutil

def get_info_dict(filename):
    with open(filename) as jsonfile:
        info = json.load(jsonfile)
    return info

def get_season_episode(info, filename):
    results = re.findall(r'\d+', info['description'])[:2]
    season = results[0] if len(results) >= 1 else "000"
    episode = results[1] if len(results) >= 2 else "00"
    got_episode = len(results) >= 2
    if season == "000" or episode == "00":
        episode = "%02d" % random.randrange(0, 100)
        print("Could␣not␣detect␣both␣season␣and␣episode\
␣number␣from␣video␣description␣of␣video␣'%s'" % filename)
    return season, episode

def get_date(info, filename):
    title = info["fulltitle"]
    dateparts = re.findall(r'\d+', title)[:3]
```

```
    date = ''
    if len(dateparts) < 3:
        print("Could not detect full airing date from video\
 title of video '%s'. Using upload date instead."\
            % filename)
        udate = info["upload_date"]
        date = "%s. %s. %s." % (udate[6:], udate[4:6],
            udate[:4])
    else:
        date = '%02d. %02d. %04d.' % (int(dateparts[0]),
            int(dateparts[1]), int(dateparts[2]))
    return date

def create_empty_file(filename):
    open(filename, 'a').close()

def get_video_dict(info, season, episode, filename):
    d = dict()
    d["season"] = season
    d["episode"] = episode
    d["date"] = get_date(info, filename)
    d["url"] = info["webpage_url"]
    return d

filenames = sys.argv[1:]
newnames = []
for fn in filenames:
    filename = fn + ".info.json"
    info = get_info_dict(filename)

    season, episode = get_season_episode(info, filename)
    newname = "c%03de%02da" % (int(season), int(episode))

    # If episode was already processed before
    processed = os.path.isfile(r"out/" + newname + ".txt")
    if not processed:
        dir = re.findall(r'.*/|$', filename)[0]
        fullnewname = dir + newname
        newnames.append(fullnewname)

        shutil.copyfile(fn, fullnewname)

        d = get_video_dict(info, season, episode, filename)
        with open(fullnewname + ".info.json", "w+")\
            as jsonfile:
            json.dump(d, jsonfile)
```

```
print(*newnames)
sys.exit(0)
```

Листинг 18: Имплементација скрипте *info_rename.py*

Након позива помоћне скрипте *info_rename.py*, функција *via_urls* скрипте *run.sh* врши премештање нових видео датотека заједно са придруженим *.info.json* датотекама у директоријум *local/*, након чега путање премештених видео датотека прослеђује функцији *via_filenames*, чија је имплементација дата у листингу 19.

```
via_filenames ()
{
    python3 extract.py "$@"
}
```

Листинг 19: Имплементација функције *via_filenames* скрипте *run.sh*

Ова функција прослеђује све своје аргументе скрипти *extract.py*, која садржи главни изворни код апликације. Унутар ове датотеке се налазе дефиниције класа *EpisodeMasks* и *EpisodeProcessor*, као и све помоћне функције које методе тих класа користе. Ова датотека се понаша као скрипта којој се прослеђују једна или више путања локално складиштених видео датотека емисије ТВ Слагалица. Имплементација скрипте у оквиру датотеке *extract.py* дата је у листингу 20.

```
filenames = sys.argv[1:]
if len(filenames) == 0:
    print("No input files to process. Exiting...")
    sys.exit(0)

# Check if all input files exist
for fn in filenames:
    if not os.path.isfile(fn):
        print("error: file '%s' doesn't exist" % fn,
              file = sys.stderr)
        sys.exit(-2)
    filename = fn + ".info.json"
    if not os.path.isfile(filename):
        print("error: file '%s' doesn't exist" % filename,
              file = sys.stderr)
        sys.exit(-2)

ep = EpisodeProcessor()
for fn in filenames:
    filename = fn + ".info.json"
```

```
    info = get_info_dict(filename)
    ep.process_episode(fn, info["season"], info["episode"],
        info["date"], info["url"])
sys.exit(0)
```

Листинг 20: Имплементација скрипте *extract.py*

Наведена скрипта најпре проверава да ли јој је прослеђена бар једна путања до локално складиштене видео датотеке емисије ТВ Слагалица. После се врши провера постојања свих видео датотека чије су путање прослеђене скрипти, као и провера постојања свих придружених *.info.json* датотека.

Уколико све потребне датотеке постоје, редом се процесира свака видео датотека чија је путања прослеђена скрипти. Врши се иницијализација објекта класе *EpisodeProcessor*, након чега се добављају сви потребни подаци о емисији из истоимене датотеке са *.info.json* екстензијом позивом помоћне функције *get_info_dict*, чија је имплементација дата у листингу 21.

```
def get_info_dict(filename):
    with open(filename) as jsonfile:
        info = json.load(jsonfile)
    return info
```

Листинг 21: Имплементација помоћне функције *get_info_dict*

Следећи корак састоји се из позива методе *process_episode* објекта класе *EpisodeProcessor*, којој се прослеђују путања видео датотеке и подаци о одговарајућој емитованој емисији ТВ Слагалице. Имплементација ове методе дата је у листингу 22.

```
def process_episode(self, filename, season, episode, date,
        url):
    print("Processing␣video␣file␣'%s'" % filename)
    season, episode, date, url = \
        self._format_episode_params(season, episode,
            date, url)
    code = "c" + season + "e" + episode + "a"

    cap = cv2.VideoCapture(filename)
    width, height, total_frames, framerate = \
        self._get_video_params(cap)

    print("␣␣Processing␣frames")
    mask = self._get_mask(season, episode, width, height)
    frames, dists = self._process_frames(cap, total_frames,
        framerate, mask)
```

```
        print(dists)

        print("␣␣Detecting␣final␣frames␣for␣the␣association\
␣quiz␣segments")
        threshold = self._get_threshold(dists)
        asocs = self._get_asoc_frames(frames, dists, threshold)

        print("␣␣Performing␣OCR")
        images, words = self._process_asoc_frames(asocs, mask)
        #print(words)

        print("␣␣Outputting␣results")
        outroot, outdir = self._make_dirs(code)
        self._writeout_txt(outroot, code, season, episode, date,
            url, words)
        self._writeout_asoc_frames(outdir, asocs)
        self._writeout_images(outdir, images)

        print("␣␣Done")
```

Листинг 22: Имплементација методе *process_episode* klase *EpisodeProcessor*

Обраду улазне видео датотеке метода *process_episode* класе *EpisodeProcessor* врши у следећим корацима:

1. Правилно форматирање података о емисији у облику ниски позивом методе *_format_episode_params*
2. Одређивање ширине и висине фрејмова, укупног броја фрејмова у видео датотеци као и брзину смењивања фрејмова позивом методе *_get_video_params*
3. Добављање објекта класе *EpisodeMasks* чије маске одговарају величини фрејмова видео датотеке позивом методе *_get_mask*
4. Селекција и анализа фрејмова видео датотеке и израчунавање њихових одстојања од референтне слике игре асоцијација позивом методе *_process_frames*
5. Одређивање вредности прага класификације фрејмова видео датотеке позивом методе *_get_threshold*
6. Одређивање последњих фрејмова обе итерације игре асоцијација позивом методе *_get_asoc_frames*
7. Изоловање појединачних поља фрејмова игре асоцијација, њихова обрада као и оптичко препознавање знакова обрађених слика позивом методе *_process_asoc_frames*
8. Прављење резултујућих излазних датотека позивом метода *_writeout_txt*, *_writeout_asoc_frames* и *_writeout_images*, од којих је од највећег интере-

са за корисника апликације излазна текстуална датотека чије се прављење извршава у оквиру методе *_writeout_txt*.

2.3.2 Команда *sync*

Имплементација команде *sync* у оквиру скрипте *run.sh* дата је у листингу 23.

```
if [ "$cmd" == "sync" ]
then
    echo "Downloading playlist info..."
    youtube-dl -j --flat-playlist --playlist-reverse\
        "`cat playlist_url.txt`"\
        | jq -r '"https://www.youtube.com/watch?v=\(.id)"'\
        > playlist.txt
    echo "Done"
fi
```

Листинг 23: Имплементација команде *sync* скрипте *run.sh*

Програму *youtube-dl* прослеђује се URL *плеј листе* емисија ТВ Слагалице која садржи све отпремљене видео датотеке емисије. URL дате плеј листе садржан је у оквиру датотеке *playlist_url.txt*. Прослеђивањем параметра *-j* програму *youtube-dl* остварује се исписивање података о свакој отпремљеној емисији у JSON формату на стандардном излазу програма. Ови излазни подаци се прослеђују ка стандардном улазу програма *jq*, који на основу параметара који су му прослеђени исписује URL-ове сваке видео датотеке на стандардном излазу. Стандардни излаз програма *jq* се преусмерава у датотеку *playlist.txt* која ће садржати URL-ове свих емисија ТВ Слагалице.

2.3.3 Команда *random*

Имплементација команде *random* у оквиру скрипте *run.sh* дата је у листингу 24.

```
if [ "$cmd" == "random" ]
then
    urls=`python3 randlines.py playlist.txt $2`
    ecode=$?
    if [ "$ecode" -eq "0" ]
    then
        via_urls $urls
    elif [ "$ecode" -eq "253" ]
    then
        >&2 echo "Did you forget to run the 'sync' command?"
        exit -1
```

```
    fi
fi
```

Листинг 24: Имплементација команде *random* скрипте *run.sh*

Задатак ове команде је да покрене преузимање и анализу N случајно изабраних видео датотека емисије ТВ Слагалица на основу URL-ова садржаних у оквиру датотеке *playlist.txt*. За случајан избор N URL-ова из ове датотеке врши се позив помоћне Python скрипте која је садржана у оквиру датотеке *randlines.py*.

Уколико је извршавање помоћне скрипте успешно, повратна информација те скрипте у облику листе URL-ова се прослеђује функцији *via_urls*. Уколико је неуспешно извршавање помоћне скрипте услед непостојања датотеке *playlist.txt* или уколико је та датотека празна, излазни код помоћне скрипте биће једнак 253, при чему се обавештава корисник скрипте *run.sh* да би најпре требао да скрипти проследи *sync* команду.

Имплементација помоћне Python скрипте *randlines.py* дата је у листингу 25.

```
import sys
import random

filename = ''
if len(sys.argv) > 1:
    filename = sys.argv[1]
else:
    print("error: unspecified input file",
        file = sys.stderr)
    sys.exit(-1)

num_lines = 0
if len(sys.argv) > 2:
    try:
        num_lines = int(sys.argv[2])
        if num_lines < 1:
            print("error: non-positive number of items",
                file = sys.stderr)
            sys.exit(-2)
    except ValueError:
        print("error: invalid number of items",
            file = sys.stderr)
        sys.exit(-2)
else:
    print("error: unspecified number of items",
        file = sys.stderr)
    sys.exit(-1)
```

```
try:
    with open(filename) as f:
        lines_in = [line.rstrip('\n') for line in f]
except FileNotFoundError:
    print("error: file '%s' not found" % filename,
        file = sys.stderr)
    sys.exit(-3)

if len(lines_in) == 0 or len(lines_in) == 1 and
        lines_in[0] == '':
    print("error: file '%s' is empty" % filename,
        file = sys.stderr)
    sys.exit(-3)

if num_lines > len(lines_in):
    num_lines = len(lines_in)
    print("warning: specified number of items exceeds\
 number of lines in input file.\
 Trimming number of items to %d."\
        % num_lines)
lines_out = random.sample(lines_in, num_lines)
print(*lines_out)
sys.exit(0)
```

Листинг 25: Имплементација скрипте *randlines.py*

2.3.4 Команда *all*

Сврха команде *all* је да покрене процес преузимања и анализе свих видео датотека емисије ТВ слагалица чији су URL-ови садржани у оквиру датотеке *playlist.txt*. Њена имплементација дата је у листингу 26

```
if [ "$cmd" == "all" ]
then
    [ -s "playlist.txt" ] &&\
    { urls=`cat playlist.txt` && ! [ -z "$urls" ]; }\
    && via_urls $urls ||\
    { >&2 echo -e "error: 'playlist.txt' is nonexistant\
 or empty.\nDid you forget to run the 'sync' command?";\
    exit -1; }
fi
```

Листинг 26: Имплементација команде *all* скрипте *run.sh*

Најпре се врши провера постојања датотеке *playlist.txt* као и тога да ли је она празна или није. Уколико датотека постоји и није празна, њен садржај се про-

слеђује функцији *via_urls*. У супротном, корисник се обавештава о томе да треба најпре да скрипти зада команду *sync*.

Ова команда имплементирана је применом технике под називом *short circuit evaluation* која је доступна у Bash командном језику. Овакав облик имплементације представља погодну алтернативу *if/elif/else блоковима* уколико је логичка структура условних редоследа израза сложена.

2.3.5 Команда *local*

Све до сада објашњене команде су користиле URL-ове као вид идентификације видео датотека емисије ТВ Слагалица. Међутим, може постојати потреба корисника за обрадом видео датотека емисије које су складиштене локално. Ту функционалност омогућава команда *local*, чија је имплементација дата у листингу 28.

```
if [ "$cmd" == "local" ]
then
    if [ $# -lt 2 ]
    then
        >&2 echo "error: no file names supplied"
        exit -2
    fi
    via_filenames "${@:2}"
fi
```

Листинг 27: Имплементација команде *local* скрипте *run.sh*

Начин функционисања ове команде је врло једноставан. Уколико је команди прослеђена бар једна путања локалне видео датотеке емисије ТВ Слагалица, скрипта прослеђује задате путање функцији *via_filenames*. У супротном, корисник се обавештава да мора команди да проследи и путање видео датотека за анализу.

Треба напоменути да путање видео датотека које се прослеђују овој команди треба искључиво да воде до директоријума *local/* зато што су у оквиру тог директоријума садржане *.info.json* датотеке које у себи садрже податке о броју циклуса, броју емисије у оквиру циклуса и исправном датуму емитовања емисије.

```
if [ "$cmd" == "local" ]
then
    if [ $# -lt 2 ]
    then
        >&2 echo "error: no file names supplied"
        exit -2
```

```
    fi
    via_filenames "${@:2}"
fi
```

Листинг 28: Имплементација команде *local* скрипте *run.sh*

3 Пример употребе

Функционалност апликације биће демонстрирана кроз један пример њене употребе. Погодан пример за демонстрацију рада апликације јесте преузимање и анализа једне насумично изабране емисије ТВ Слагалице.

Најпре треба преузети листу URL-ова свих епизода емисије ТВ Слагалица. У командној линији треба покренути скрипту *run.sh* и задати јој команду *sync*. Следи приказ примера извршења наведене команде.

```
$ ./run.sh sync
Downloading playlist info...
Done
```

После се може наставити са задавањем команде *random* скрипти и прослеђивањем вредности 1 параметра команде:

```
$ ./run.sh random 1
[youtube] oxFty6EnCA4: Downloading webpage
[youtube] oxFty6EnCA4: Downloading video info webpage
[info] Writing video description metadata as JSON to:
downloads/oxFty6EnCA4.info.json
[download] Destination: downloads/oxFty6EnCA4
[download] 100% of 61.09MiB in 00:30
[ffmpeg] Fixing aspect ratio in "downloads/oxFty6EnCA4"
Processing video file 'local/c091e45a'
  Processing frames
  Detecting final frames for the association quiz segments
  Performing OCR
  Outputting results
  Done
```

Испоставља се да је преузета и обрађена видео датотека емисије ТВ Слагалица редног броја циклуса 91 и редног броја емисије 45 у оквиру тог циклуса. Сходно томе, излазна датотека која садржи резултате препознавања речи из поља игре асоцијација има путању *out/c091e45a.txt*. Следи приказ садржаја ове излазне датотеке.

```
091. циклус
45. емисија
15. 10. 2015.
https://www.youtube.com/watch?v=oxFty6EnCA4


A1 - ' HTL'K'JUA U'LF
'LKPUJHAQA
A2 - ЗНАКОВИ
A3 - АУТО
A4 - МПЕЧНИ
```

```
 А - ПУТ
Б1 - ПРИПРЕМА
Б2 - РЕАПИЗАЦ.
Б3 - БИ РО
Б4 - ’’НВОТ
 Б - ПРОЈЕКАТ
В1 - СПАРТАК
В2 - КАРПОШ
В3 - БОКСЕРИ
В4 - ХЕРЦЕГОВ.
 В - YСТАНАК
Г1 - М. МОРЕ
Г2 - ДАНГА
Г3 - СТРАДИЈА
Г4 - С.!НИНИСТРА
 Г - РАДОЈЕД.
??? - ВОВА

А1 -
А2 - НАФТА
А3 - СПЛАВ
А4 - ДИОГЕН
 А - SYP E
Б1 - MAUJHA
Б2 - ГУС ЕНША
Б3 - ЕФЕКАТ
Б4 - СТАКЛО
 Б - ЛЕПТИР
В1 - РОГО ВИ
В2 - ОРАЊЕ
В3 - БЕЧ
В4 - БЕЧ
 В - ВО
Г1 - СИР'ЋЕ
Г2 - УЊЕ
Г3 - РУСКА
Г4 - ЗЕПЕНА
 Г - САЛАТА
??? - КУПУС
```

Као што се може приметити, чак и поред примене предобраде слика изолованих поља игре асоцијација резултати препознавања речи су далеко од савршених.

4 Закључак

Оптичко препознавање знакова представља поље истраживања које спаја домене препознавања облика, вештачке интелигенције и рачунарског вида. Иако је старо неколико деценија, резултати у овом пољу су далеко од савршених. Пораст популарности примене оптичког препознавања знакова у корисничким апликацијама, посебно од настанка и развоја паметних телефона, за последицу има и пораст заинтересованости у ово поље истраживања. Технике оптичког препознавања знакова нису усавршене и њихова права моћ и применљивост ће бити очигледне тек у будућности.

Сваки процес оптичког препознавања знакова састоји се из више фаза, где је међу првим фаза обраде улазних слика. Улазну слику треба изманипулисати на начин који поспешује тачност препознавања знакова. Након погодних трансформација улазне слике треба изоловати њене делове који садрже искључиво текст. Додатном обрадом изолованог текста са слике може се постићи смањивање присутности шума као и акцентовање карактеристичних одлика знакова. После погодне предобраде улазних слика може се наставити са алгоритмима препознавања знакова.

Иако су тренутни резултати оптичког препознавања знакова далеки од идеалних, процесом пост-обраде могу се исправити грешке језгра за препознавање знакова на основу познатих информација о језику текста и других карактеристичних чињеница које су корисне при проналажењу грешака у интерпретацији знакова. Један од примера пост-обраде је коришћење речника за одређени језик ради утврђивања грешке у интерпретацији речи или парова речи. Погрешне речи могу се заменити најсличнијим исправним речима са највећом вероватноћом појединачног или заједничког појављивања у тексту.

Процес оптичког препознавања знакова може се оптимизовати ако се језгру за оптичко препознавање знакова наговесте неке чињенице о тексту на сликама које би њему биле од користи. На пример, могу се наговестити језик текста, фонт текста, као и назнаке о генералној структури текста. Уколико текст на слици садржи једну реч, један ред речи, више редова речи, или само неструктуирану секвенцу знакова, та чињеница може се наговестити језгру за оптичко препознавање знакова што би оптимизовало његову обраду слике и довело до повећања тачности у препознавању знакова.

Литература

[1] *Optical Character Recognition (OCR) – How it works* (приступљено у октобру 2019. године):
https://www.nicomsoft.com/optical-character-recognition-ocr-how-it-works/

[2] Sezgin, Mehmet; Sankur, Bulent, *Survey over image thresholding techniques and quantitative performance evaluation*, Journal of Electronic Imaging, 2004.

[3] *Morphological Image Processing* (приступљено у октобру 2019. године):
https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

[4] *OCR Introduction* (приступљено у октобру 2019. године):
http://www.dataid.com/aboutocr.htm

[5] *Python 3.8.0 documentation* (приступљено у октобру 2019. године):
https://docs.python.org/3/

[6] *Bash Reference Manual* (приступљено у октобру 2019. године):
https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html

[7] *OpenCV: OpenCV Modules* (приступљено у октобру 2019. године):
https://docs.opencv.org/4.1.2/

[8] *Documentation · tesseract-ocr/tesseract Wiki* (приступљено у октобру 2019. године):
https://github.com/tesseract-ocr/tesseract/wiki/Documentation

[9] *NumPy Reference – NumPy v1.17 Manual* (приступљено у октобру 2019. године):
https://docs.scipy.org/doc/numpy/reference/

[10] *youtube-dl/README.md at master · ytdl-org/youtube-dl* (приступљено у октобру 2019. године):
https://github.com/ytdl-org/youtube-dl/blob/master/README.md

[11] *jq Manual* (приступљено у октобру 2019. године):
https://stedolan.github.io/jq/manual/

[12] *sed, a stream editor* (приступљено у октобру 2019. године):
https://www.gnu.org/software/sed/manual/sed.html