



.....
Srednja šola za kemijo,
elektrotehniko in računalništvo

OCR algoritem za pomoč gibalno oviranim osebam

Raziskovalna naloga

Področje:

Aplikativni in inovacijski projekti

Avtorja:

Domen Hribernik, R4a

Tilen Koren, R4a

Mentor:

Timej Pirš

Mestna občina Celje, Mladi za Celje

Celje, december 2020

» Resnična nevarnost
ni, da računalniki
začnejo razmišljati
kot ljudje, ampak da
ljudje začnejo
razmišljati kot
računalniki.«

Sydney J. Harris (1917-1986)

ZAHVALA

Za nasvete in predloge ter za pomoč pri nastajanju naloge se zahvaljujema najinemu mentorju, gospodu Timeju Piršu, ki naju je strokovno usmerjal in nama s konstruktivnimi napotki pomagal, da napisano še izboljšava.

Zahvaljujema se mu za možnost, da sva se naučila nekaj novega.

Domen Hribernik, Tilen Koren

1 POVZETEK

Glavni namen raziskovalne naloge je bil raziskati delovanje tehnologije OCR in jo uporabiti za pomoč gibalno oviranim osebam pri parkiranju. Prav problem parkiranja danes pesti veliko število gibalno oviranih oseb, ki bi si želeli na tem področju videti spremembe. Eden izmed glavnih problemov je zloraba parkirnih mest za gibalno ovirane osebe.

V nadaljevanju je predstavljen program, ki deluje tako, da se preko registracije uporabnikovi podatki vpišejo v podatkovno bazo, od koder jih na mestu parkiranja program preveri in uporabnika ob uspešnem preverjanju spusti na parkirni prostor. Preverjanje na mestu parkiranja zmanjša možnost zlorabe ter gibalno oviranim osebam omogoči nemoteno možnost parkiranja.

Ključne besede: OCR, Python, programiranje, Tesseract, GOO

KAZALO VSEBINE

1 POVZETEK	3
2 UVOD	8
2.1 Opredelitev področja in raziskovalnega problema	8
2.2 Cilji raziskovalne naloge	8
2.3 Hipoteze	9
2.4 Metodologija in omejitve pri raziskavi	9
3 PYTHON	10
3.1 Python	10
3.2 Zgodovina	10
3.3 Uporaba	11
3.4 Skupnost Python	12
4 OCR	13
4.1 Kaj je OCR?	13
4.2 Zgodovina	13
4.3 Uporaba	13
4.4 Vrste OCR	14
4.5 Postopek OCR	14
4.5.1 Predhodna obdelava	14
4.5.2 Prepoznavanje besedila	15
4.5.3 Naknadna obdelava	16
4.5.4 Specificirana optimizacija za aplikacije	16
4.6 Alternativne rešitve	17
4.6.1 Forsiranje boljšega vnosa	17
4.6.2 Crowdsourcing	17
4.7 Natančnost	17
5 PROGRAM	19
5.1 Ideja o uporabi programa	19
5.1.1 Registracija	19
5.1.2 Prijava	19
5.1.3 Prepoznavna obraza	20
5.1.4 Vpis podatkov invalidske izkaznice	20
5.1.5 Shranjevanje v bazo	20

5.1.6	OCR.....	21
5.2	Delovna sredstva spletne strani	21
5.2.1	Bootstrap.....	21
5.2.2	Node	22
5.2.3	Mongo.....	22
5.2.4	Ajax.....	22
5.2.5	EJS	23
5.2.6	Npm	23
5.2.7	Express	23
5.2.8	Heroku	23
5.3	Prepoznavna obraza	24
5.4	Prepoznavanje uporabnika z OCR	25
5.4.1	Prepoznavanje invalidske izkaznice	26
5.4.2	Prepoznavanje registrske tablice.....	29
6	IZDELAVA OCR IN PRIMERJAVA S TESSERACT.....	31
6.1	Izdelava OCR.....	31
6.2	Tesseract OCR	35
6.3	Primerjava OCR-jev	37
7	ANALIZA ANKETE	38
7.1	Izbor in struktura vzorca anketiranih.....	38
7.2	Rezultati primarne raziskave	39
8	OVREDNOTENJE HIPOTEZ.....	41
9	ZAKLJUČEK.....	43
10	LITERATURA	44

KAZALO SLIK

Slika 1: Registracija	19
Slika 2: Prijava	20
Slika 3: Vpis podatkov	20
Slika 4: Podatkovna baza	21
Slika 5: Koda za prepoznavanje obraza	24
Slika 6: Rezultat prepoznavanja obraza	25
Slika 7: Črno-bela izkaznica	26
Slika 8: Filtrirana slika izkaznice	27
Slika 9: Odstranjevanje ozadja izkaznice	27
Slika 10: Obrezana slika izkaznice	28
Slika 11: Rezultat OCR izkaznice	28
Slika 12: Koordinate izkaznice	28
Slika 13: Črno-bela slika avta	29
Slika 14: Filtrirana slika avta	29
Slika 15: Odstranjevanje ozadja registrske tablice	30
Slika 16: Obrezana slika registrske tablice	30
Slika 17: Rezultat OCR registrske tablice	30
Slika 18: Koordinate registrske tablice	30
Slika 19: Primer algoritma KNN	31
Slika 20: Baza MNIST	32
Slika 21: Spreminjanje slik v enodimenzionalne slike	32
Slika 22: Metoda za izvajanje KNN	33
Slika 23: Metodi za izračun razdalje	33
Slika 24: Rezultat OCR	33
Slika 25: Koda za shranjevanje slik	34
Slika 26: Prepoznane slike	34
Slika 27: Prepoznavanje najine številke	34
Slika 28: Vključevanje pytesseract	35
Slika 29: Koda za izris okvirja in besedila	35
Slika 30: Rezultat pytesseract	36
Slika 31: Rezultat branja v konzoli	36

KAZALO GRAFOV

Graf 1: Starost	38
Graf 2: Spol.....	38
Graf 3: Pogostost parkiranja na mestih za GOO	39
Graf 4: Zadovoljstvo s številom parkirnih mest	39
Graf 5: Mnenje o uporabnosti programa.....	40

KAZALO PRILOG

Priloga 1: Vprašalnik o parkiranju na parkirnih mestih za gibalno ovirane osebe.....	45
--	----

2 UVOD

2.1 Opredelitev področja in raziskovalnega problema

Umetna inteligenca iz dneva v dan zmeraj bolj vstopa v naša življenja na vseh področjih. Uporabljamo jo praktično vsak dan, a se večina ljudi tega ne zaveda, saj se pojavlja že v našem lastnem žepu, v službi in šoli, doma, v avtu, itd. Zaradi te hitre in učinkovite dobe razvoja umetna inteligenca velja za gonilo novega veka v človeški zgodovini. Vsa predvidevanja za človeško prihodnost temeljijo na napravah, ki uporabljajo umetno inteligenco.

Eno izmed pomembnih področij je tudi vpliv umetne inteligence na življenje gibalno oviranih oseb (GOO). Od zvokovnih ukazov do prvih samovozečih avtomobilov je velik del umetne inteligence v veliko pomoč GOO pri opravljanju njihovih vsakdanjih opravkov in dejavnosti. Eden izmed ključnih dejavnikov pri njihovem obiskovanju različnih krajev so parkirni prostori, ki so namenjeni samo njim. Včasih pa se zgodi, da prav te prostore zasedajo osebe, ki tam ne bi smele parkirati in tako odvzamejo za nekatere izjemno pomemben prostor.

V raziskovalni nalogi nameravam raziskati, kako deluje tehnologija OCR in kako jo lahko uporabiva v pomoč GOO. Raziskovanje bo usmerjeno v tehnologijo OCR, ki jo bova realizirala s programskim jezikom Python. Glavna ideja uporabe te tehnologije je bila ustvariti parkirišče, namenjeno GOO, do katerega bi dostopali preko zapore, ki bi prepoznala registrsko tablico avtomobila in preverila, ali se tablica ujema s tisto iz podatkovne baze. V to bazo pa bi se GOO vpisali preko spletne aplikacije, kjer bi za dokaz svoje invalidnosti priložili svojo invalidsko izkaznico in sliko svojega obraza za dokazovanje na mestu parkiranja. S tem bi dosegla tristopenjsko varnostno preverjanje.

2.2 Cilji raziskovalne naloge

Cilji v teoretičnem delu:

- predstaviti programski jezik Python,
- predstaviti tehnologijo OCR.

Cilji v praktičnem delu:

- izdelava lastnega OCR algoritma,
- primerjava lastnega algoritma z že obstoječimi,
- izdelava programa za prepoznavanje registrske tablice in invalidske izkaznice,
- izdelava programa za prepoznavanje obrazov,
- izdelava spletne aplikacije za vpis v bazo,
- izdelava podatkovne baze za shranjevanje oseb.

2.3 Hipoteze

H1: Izdelek bo v veliko pomoč GOO.

H2: Izdelek bo uporabljen na več mestih.

H3: Občina bo bolj prijazna do GOO.

H4: Posledica je manjša možnost zlorabe.

H5: Težko bo anketirati reprezentativno število GOO.

H6: Anketo bodo v večini izpolnili starejše GOO.

2.4 Metodologija in omejitve pri raziskavi

Raziskovanja sva se lotila postopoma. Najprej sva pregledala elektronske vire, ki se nanašajo na obravnavano tematiko. Na osnovi tega sva opredelila problem, oblikovala cilje in hipoteze.

Nato sva se lotila podrobnejšega proučevanja elektronskih virov. Glede na to, da se o programskem jeziku Python še nismo učili v šoli, sva morala najprej podrobneje raziskati to področje. Nato sva raziskovala tehnologijo OCR in programe, ki se usmerjajo na isto področje.

V nadaljevanju sva se lotila še primarnega raziskovanja. Izdelala sva anketni vprašalnik, s pomočjo katerega sva želela podrobneje ugotoviti parkirne navade GOO in njihovo mnenje o najini ideji. Primarno raziskovanje je potekalo na vzorcu 75 GOO – večinoma iz različnih društev za GOO celjske regije.

Omejitve pri raziskavi so bile naslednje:

- kratek čas, nekaj mesecev,
- omejeni pisni in elektronski viri o tehnologiji OCR,
- slabo predhodno poznavanje področja raziskovanja,
- velikost in omejenost vzorca, raziskavo sva zaradi omejenega časa izvedla na vzorcu 75 GOO (večinoma iz društev za GOO).

3 PYTHON

3.1 Python

Python je interpretiran, objektno usmerjen programski jezik na visoki ravni z dinamično semantiko. Vgrajene podatkovne strukture na visoki ravni v kombinaciji z dinamičnim tipkanjem in dinamično vezavo so zelo privlačne za hiter razvoj aplikacij, pa tudi za uporabo kot skriptni ali lepilni jezik za povezovanje obstoječih komponent. Pythonova enostavna sintaksa, ki se je enostavno naučiti, poudarja berljivost. Python podpira module in pakete, kar spodbuja modularnost programa in ponovno uporabo kode. Tolmač Python in obsežna standardna knjižnica sta na voljo v izvorni ali binarni obliki brezplačno za vse večje platforme in ju je mogoče prosto distribuirati.

3.2 Zgodovina

Zgodovina programskega jezika Python se začne v osemdesetih letih prejšnjega stoletja z Guidom Van Rossum. Ta decembra 1989 začne delati na aplikacijah v Centrum Wiskunde & Informatica (CWI), ki se nahaja na Nizozemskem. Na začetku je bil to hobi projekt, ker je iskal zanimiv projekt, s katerim bi bil zaposlen med božičem. Programski jezik, v katerem naj bi Python uspel, je ABC programski jezik, ki je imel povezavo z operacijskim sistemom Amoeba in je imel funkcijo obravnave izjem. V svoji tedanji karieri je Rossum pomagal ustvariti ABC programski jezik, a je v njem videl veliko težav, vendar pa mu je bila vseč večina funkcij, ki jih je uporabljal jezik. Nato pa se je domislil zelo pametne ideje. Za svoj programski jezik je vzel sintakso ABC programskega jezika in nekatere njegove dobre lastnosti. Prišlo je tudi veliko pritožb, zato je te težave popolnoma odpravil in ustvaril dober skriptni jezik, ki je odpravil vse pomanjkljivosti. Navdih za ime Python je dobil iz BBC-jeve TV oddaje - 'Leteči cirkus Montyja Pythona', saj je bil velik oboževalec TV oddaje in si je za svoj izum želel kratko, edinstveno in nekoliko skrivnostno ime.

Jezik je bil končno izdan leta 1991. Ko je izšel, je za izražanje konceptov uporabil veliko manj kod, če ga recimo primerjamo s programskimi jeziki Java, C++ in C. Tudi njegova oblikovalska filozofija je bila kar dobra. Njegov glavni cilj je zagotoviti berljivost kode in napredno produktivnost razvijalcev. Ko je bil izdan, je imel več kot dovolj zmogljivosti za zagotavljanje razredov z dedovanjem, obdelavo izjem nekaj osnovnih podatkovnih tipov in funkcije.

Python 2.0, izdan leta 2000, je predstavil nove funkcije, kot so razumevanje seznamov in sistem za zbiranje smeti s štejetjem referenc, leta 2020 pa je bila ukinjena različica 2.7. Python 3.0, izdan leta 2008, je bil pomembna obnovitev jezika, ki ni popolnoma združljiva za nazaj in veliko kode Python 2 ne deluje nespremenjena na Python 3. Z iztekom življenjske dobe Pythona 2 je podprt samo Python 3.6.x in novejši, starejše različice pa še vedno podpirajo npr. Windows 7. Tolmači Python so podprti za običajne operacijske sisteme (v preteklosti pa za veliko več). Globalna skupnost programerjev razvija in vzdržuje CPython, brezplačno in odprtokodno referenčno izvedbo. Neprofitna organizacija, Python Software Foundation, upravlja in usmerja vire za razvoj Pythona in CPythona. Januarja 2021 se je Python uvrščal na tretje mesto v indeksu najpopularnejših programskih jezikov TIOBE, za C in Javo, saj je pred tem osvojil drugo mesto in nagrado za najbolj priljubljeno rast za leto 2020.

Kot omenjeno v prejšnjem odstavku, je Python trenutno na voljo v različicah 2.7 ali 3.X. Python 3 je čistejši in hitrejši, vendar moramo upoštevati, da nekateri paketi zunanjih

uporabnikov še vedno nudijo samo podporo 2.7. Običajno so trenutni paketi napisani ali posodobljeni za uporabo s Python 3.

3.3 Uporaba

Tako kot drugi kodni jeziki je tudi Python eden od nevidnih elementov, od katerih imamo koristi, ne da bi ga poznali. Danes je uporabljen na številnih področjih našega vsakdanjega življenja:

- podatkovna znanost,
- strojno učenje,
- spletni razvoj,
- izobraževanje na področju računalništva,
- računalniški vid in obdelava slik,
- razvoj iger,
- zdravstvo in farmakologija,
- biologija in bioinformatika,
- nevroznanost in psihologija,
- astronomija,
- robotika,
- avtonomna vozila,
- poslovanje,
- meteorologija,
- razvoj grafičnih uporabniških vmesnikov (GUI).

Zakaj pa se danes uporablja prav Python in ne kakšen drug programski jezik?

Python je zmogljiv programski jezik za splošno uporabo in postaja vse bolj priljubljeno orodje v raziskavah. Za učenje je intuitiven, ima cvetočo spletno skupnost in je odprtokoden (brezplačen za vas in druge). Njegova priljubljenost deloma izhaja iz enostavne, vsestranske funkcionalnosti. Python lahko opravi večino vsakodnevnih raziskovalnih nalog in se lahko uporablja v več korakih raziskovalnega cevovoda (npr. izvajanje poskusov z udeleženci, organizacija podatkov, obdelava/manipulacija podatkov, statistična analiza/modeliranje in vizualizacija). Namesto da za izvajanje različnih nalog uporablja različne programe, lahko Python raziskovalcem prihrani veliko časa in težav.

Glede na premik k odprti in ponovljivi znanosti Python ponuja prednost pred drugo lastniško programsko opremo, ki pogosto zahteva drage licence in je zato nekaterim raziskovalcem nedostopna. Ko Python napišete in delite svojo "kodo", jo lahko drugi enostavno dostopajo in uporabljajo, ne da bi naleteli na plačilne stene ali težave z licenciranjem. Ta vidik, ki ga vodi skupnost, razvijalcem omogoča uvajanje "paketov" drugih proizvajalcev (imenovanih tudi "knjižnice") ali svežnjev kode, ki jih je mogoče enostavno deliti (pogosto vključujejo dokumentacijo, primere podatkov in vadbice), ki razširjajo osnovno funkcionalnost Pythona. Paketi vam prihranijo precej časa. Če imate težavo (npr. statistika, risanje, filtriranje podatkov), jo je nekdo verjetno že rešil in je namestil paket, odprt za uporabo, preko katerega lahko ta problem rešite.

Znane organizacije, ki uporabljajo Python:

- Mozilla, najbolj znana po Firefoxu, pravi, da ima v Pythonu napisanih več kot "230 tisoč vrstic kode".
- Google deli svoje notranje usposabljanje za Python.
- Microsoft spodbuja razvoj Pythona s svojo IDE, Visual Studio Code.
- Netflix deli svojo široko uporabo Pythona za vse, od regionalne programske opreme za spremljanje odpovedi do znanosti o podatkih.
- Uber pravi, da za izmenjavo podatkov uporablja Jupyter Notebook in IPython.
- Reddit je v veliki meri napisan v Pythonu in ima izvorno kodo na GitHub.
- Dropbox je bil javni zagovornik Pythona 3 v svoji infrastrukturi.
- Slack, Digital Ocean, Lyft, Sauce Labs in Fastly vsi omenjajo uporabo Pythona v članku Increment.
- Mnoga finančna podjetja, kot so CapitalOne, Bloomberg in JPMorgan, zaposlujejo razvijalce Pythona.

Poleg tega so številne tehnologije infrastrukture IT napisane v Pythonu. Je primarni jezik, ki se uporablja za obsežen projekt računalništva v oblaku OpenStack, ki poganja zasebne in javne oblake v podatkovnih centrih po vsem svetu. Programska oprema za avtomatizacijo infrastrukture Ansible je napisana tudi v Pythonu.

Uporablja se tudi za pisanje namizne programske opreme, kot sta Caliber in OpenShot. Blender je med številnimi aplikacijami, napisanimi v drugih jezikih, ki uporabnikom omogočajo pisanje skriptov v Pythonu. Je tudi priljubljen jezik za strojno učenje ter znanstveno, statistično, matematično in druge vrste specializiranega računalništva. Tudi Raspberry Pi je svoje ime dobil po načrtu ustanoviteljev za uporabo platforme za poučevanje Pythona.

3.4 Skupnost Python

Python ima ogromno uporabniško skupnost. Pythonova priljubljenost je vzrok in učinek njegove skupnosti. Glede na razvrstitev IEEE Spectrum je bil leta 2018 prvi programski jezik in je po podatkih raziskave razvijalcev StackOverflow za leto 2019 najbolj priljubljen in najbolj priljubljen jezik. Pythonistas, kot se sami imenujejo člani skupnosti, se na konferencah PyCon srečujejo po tisočih po vsem svetu.

To pomeni, da ne glede na težavo, ki jo poskušate rešiti, obstaja verjetnost, da že obstajajo ljudje, ki delajo na rešitvi. Velika verjetnost je tudi, da imajo skupno kodo, dokumentacijo, vadnice in primere za pomoč pri programiranju rešitve v Pythonu. Na izbiri je veliko IDE-jev in drugih razvojnih orodij, na voljo pa je tudi na tisoče odprtokodnih paketov za razširitev Pythona do skoraj vsega, kar si lahko zamislite.

4 OCR

4.1 Kaj je OCR?

Optično prepoznavanje znakov ali OCR je orodje, ki pretvarja sliko besedila v digitalno obliko. Slika je lahko skeniran dokument, slikan dokument, posnetek zaslona, itd. OCR se zelo pogosto uporablja za pretvarjanje dokumentov, ki so natisnjeni na papir v digitalno obliko, kot potni list, razni računi, bančni izpiski itd. To je zelo uporabno, saj so lahko po prevajanju v digitalno obliko ti dokumenti spremenjeni, digitalno shranjeni in tudi obdelani z drugimi procesi, kot prevajanje besedila v drug jezik ali pa prevajanje besedila v govor.

Na področju OCR se uporablja prepoznavanje vzorcev, umetna inteligenca in računalniški vid. Novejši programi se začnejo učiti prepoznavanja slik samo z enim znakom in tudi v enem fontu. Naprednejši programi, ki jih uporabljamo danes, pa so sposobni prepoznati več besed z veliko natančnostjo in tudi delujejo z večino fontov. Nekateri programi lahko tudi pretvorijo grafične podatke o formatu kot slike, odstavke, ter druge netekstovne komponente.

4.2 Zgodovina

OCR se prvič pojavi v tehnologiji, povezani s telegrafijo in ustvarjanjem naprav, ki omogočajo branje slepim. V letu 1914 je Emanuel Goldberg razvil napravo, ki lahko prebere znake in jih pretvori v standardno telegrafsko kodo. Temu je sledil izum Edmuda Fournierja in to je bil optofon. To napravo si premikal čez stran in vračala je za vsak različen znak specifičen ton. Kasneje v letih 1920 in 1930 je Emanuel Goldberg izumil napravo, ki jo je poimenoval »Statistical Machine«, ta naprava se je uporabljala za iskanje po arhivu dokumentov, glede na optično prepoznavanje kode.

V letu 1974 je Ray Kurzweil nadaljeval z razvojem OCR, ki bi lahko prepoznal natisnjen tekst, v katerem koli fontu. Kurzweil je menil, da je najboljša aplikacija njegove tehnologije naprava, ki lahko prebere tekst slepim. Naprava bi prepoznala tekst in ga naglas prebrala preko računalnika. Da bi lahko naredili to napravo, sta bili potrebni 2 odkritji, boljši optični tiskalnik in sintetizator besedila v govor. Leta 1976 so uspešno dokončali to napravo. Leta 1978 je Kurzweil začel prodajati programsko verzijo OCR, podjetje LexisNexis je bilo eno izmed prvih kupcev in je uporabilo program za shranjevanje dokumentov ter novic v svojo podatkovno bazo. Dve leti kasneje, leta 1980, je Kurzweilovo podjetje kupilo Xerox in ga preimenovalo v Scansoft.

V dvajsetem stoletju je OCR postal dostopen kot storitev na internetu (WebOCR). Prednost je bila v tem, da si ga lahko uporabil na pametnih telefonih. OCR je lahko uporabljen v mobilnih napravah, ki imajo dostop do interneta za shranjevanje teksta, ki je zajet s kamero naprave, ali pa celo prevajanje teksta iz tujega jezika v jezik, ki ga izbere uporabnik. Nekateri naprave že imajo OCR funkcionalnost vgrajeno v operacijski sistem, tiste, ki tega še nimajo, pa lahko uporabljajo OCR API. Ta API vrne tekst in dodatne informacije kot lokacija teksta iz zajete slike in ga tudi pošlje naprej za dodatno procesiranje kot prevajanje ali pretvarjanje besedila v govor.

4.3 Uporaba

Danes se OCR uporablja za zelo veliko različnih stvari in je tudi razvit različno za

specifična polja dela. To specifično funkcijo programi opravijo po prevajanju slike v besedilo.

Pogosta uporaba OCR je:

- Vnos podatkov za poslovne dokumente.
- Prepoznavanje registrskih tablic.
- V letališčih za branje podatkov iz potnih listov.
- Pridobivanje ključnih informacij iz zavarovalnih dokumentov.
- Prepoznavanje prometnih znakov.
- Hitrejše pretvarjanje v digitalno obliko iz tiskanih dokumentov.
- Omogočanje iskanja po elektronskih slikah tiskanih dokumentov.
- Pretvorba rokopisa v realnem času za nadzor računalnika.
- Premagovanje sistemov CAPTCHA, čeprav so posebej zasnovani za preprečevanje OCR. Namen je lahko tudi preverjanje učinkovitosti sistemov CAPTCHA.
- Podporna tehnologija za slepe in slabovidne uporabnike.
- Pisanje navodil za vozila z identifikacijo slik CAD.
- Omogočanje iskanja optično prebranih dokumentov s pretvorbo v PDF.

4.4 Vrste OCR

Poznamo več vrst OCR:

- OCR (Optical Character Recognition) – namenjen za tipkano besedilo, preverja en znak naenkrat.
- OWE (Optical Word Recognition) – namenjen za tipkano besedilo, preverja eno besedo v jezikih, ki uporabljajo presledke kot ločilo besed, spada pod OCR.
- ICR (Intelligent Character Recognition) – lahko prepozna tudi rokopis, preverja en znak naenkrat, vključuje strojno učenje.
- IWR (Intelligent word recognition) – lahko prepozna tudi rokopis, preverja eno besedo, lahko prepozna tiskane črke in tudi kurzivno pisavo.

OCR za najpogostejšo uporabo ne potrebuje internetne povezave, saj je nameščen na sistemu. Obstaja pa več različnih spletnih storitev, ki jih zagotavlja OCR API. Primer tega je analiza gibanja rokopisa, ki se lahko uporablja kot vhod za prepoznavanje rokopisa. Glavna razlika, ki jo omogoča, je to, da ne prepozna oblike znakov in besed, ta tehnika prepozna gibe v vrstnem redu risanja, smeri risanja in tudi če je bilo vmes pisalo dvignjeno. Te dodatne informacije lahko naredijo postopek natančnejši. Ta tehnologija zajemanja je znana kot spletno prepoznavanje znakov, dinamično prepoznavanje znakov, prepoznavanje znakov v realnem času in inteligentno prepoznavanje znakov.

4.5 Postopek OCR

4.5.1 Predhodna obdelava

Programi, ki omogočajo OCR, pogosto predhodno obdelajo slike, da izboljšajo možnost uspešnega prepoznavanja. Poznamo več načinov predhodne obdelave:

- De-skew – če dokument med optičnim branjem ni bil pravilno poravnan, ga je potrebno nagniti za nekaj stopinj, zato da bodo vrstice besedila popolnoma vodoravne ali navpične. Poleg tega je lahko slika zajeta v napačni perspektivi (če dokument, ki ga beremo, ni raven). V tem primeru so vrstice krive in jih je potrebno naravnati.
- Despeckle – odstranjevanje lis iz prebranega dokumenta, popraviljanje detajlov s svetlobo. Primer je, če damo dokument pretvoriti v sliko z optičnim čitalnikom, saj se lahko v čitalniku nahajajo delci prahu, ki tudi vplivajo na končni rezultat. S to metodo se znebimo teh lis.
- Binarizacija – pretvarjanje slike iz barvne ali sive v črno-belo, imenujemo jo binarna slika, ker obstajata samo dve barvi, črna in bela. Ta proces se izvede kot preprost način ločevanja besedila ali katere druge zelene slikovne komponente odzadaj. Naloga binarizacije je nujna, saj večina komercialnih algoritmov za prepoznavanje deluje samo na binarnih slikah, saj se je izkazalo za preprostejše. Poleg tega binarizacija v veliki meri vpliva na kakovosti stopnje prepoznavanja znakov. Pomembno je tudi, da premislimo, preden se dokončno odločimo za izbiro binarizacije. Obstaja namreč več vrst binarizacije, ki so tudi različne kakovosti. Zavedati pa se moramo, da kakovost izbrane binarne metode vpliva na končni rezultat. Torej je binarizacija odvisna od optično prebranega dokumenta, besedila in stanja ohranjenosti dokumenta.
- Odstranjevanje nepomembnih elementov – če je OCR narejen specifično za prevajanje teksta, lahko v tem postopku odstranimo slike, grafe, črte in vse druge elemente, ki niso povezani s tekstom.
- Analiza postavitve – določi stolpce, odstavke, napise in jih prepozna kot ločene bloke. To je še posebej pomembno pri postavitvah in tabelah z več stolpci.
- Zaznavanje črt in besed – določi osnovno črto, na kateri so postavljene črke in besede, ter obliko besed in znakov, po potrebi tudi loči besede.
- Izolacija znakov ali segmentacija – pri znakovnem OCR je potrebno ločiti več znakov, ki so povezani zaradi slikovnih artefaktov in združiti posamezne znake, ki so zaradi artefaktov razdeljeni na več kosov. Ti slikovni artefakti se pojavijo zaradi nizko kvalitetnega načina zajemanja slike, zato se lahko temu procesu izognemo z dobro kvaliteto slike. Končni cilj procesa pa je, da znakovni OCR dobi samo znake.
- Normalizacija – potrebno je ohraniti razmerje in merilo.

Segmentacija pisav, kjer črke zavzemajo enako širino in višino, poteka preprosto tako, da sliko poravnamo na mrežo glede na to, kje bodo navpične črte mreže najmanj pogosto sekale črna območja. Za bolj kompleksne fonte so potrebne bolj izpopolnjene tehnike, težava se pojavi, ker je presledek med črkami lahko včasih večji od presledka med besedami, navpične črte pa lahko sekajo več kot en znak.

4.5.2 Prepoznavanje besedila

Obstajata dve osnovni vrsti algoritma OCR, ki lahko ustvarita razvrščen seznam kandidatnih znakov.

Prvi metodi pravimo matrično ujemanje, ta vključuje primerjavo slike s shranjenim znakom na osnovi slikovnih pik. Matrični način se zanaša na to, da je vhodni znak pravilno izoliran od ostale slike in da je shranjen znak v podobni pisavi ter enakem merilu. Ta tehnika

deluje najbolje s tipkanim besedilom in ne deluje dobro, ko naletimo na nove pisave. To tehniko je implementiral zgodnji foto-celični OCR.

Druga metoda se imenuje pridobivanje lastnosti. Ta način razgradi črke in besede v lastnosti, kot so črte, zaprte zanke, smer črt in presečišča črt. Pridobljene lastnosti naredijo postopek prepoznavanja računsko bolj učinkovit. Te lastnosti primerjamo z abstraktno vektorsko podobo lika, ki se lahko zmanjša na enega ali več prototipov znakov. Splošne tehnike zaznavanja lastnosti v računalniškem vidu so uporabne za to vrsto OCR, kar lahko vidimo pri inteligentnem prepoznavanju rokopisa in pri najbolj sodobni OCR programski opremi. Algoritmi za najbližjega soseda se uporabljajo za primerjavo slikovnih lastnosti s shranjenimi lastnostmi znaka in izbere tisti znak, ki se najbolj ujema. To naredijo tako, da vsakemu znaku določijo številko ujemanja in tisti, ki ima največjo število, je izbran.

Programska oprema, kot sta Cuneiform in Tesseract, uporablja dvoprehodni pristop k prepoznavanju znakov. Ta način je znan kot prilagodljivo prepoznavanje in uporablja oblike črk, ki so prepoznane z veliko verjetnostjo pravilnosti pri prvem prehodu, da bolje prepozna preostale črke pri drugem prehodu. To je ugodno za nenavadne fonte ali nekakovostne optične bralnice, kjer je pisava popačena, zamegljena, zbledela.

Moderna OCR programska oprema, kot sta OCRopus ali Tesseract, uporablja nevronske mreže, ki so bile usposobljene za prepoznavanje več vrstic besedila, namesto da bi se osredotočali na posamezne znake.

4.5.3 Naknadna obdelava

Natančnost OCR je mogoče povečati, če je izhod omejen z leksikonom (seznamom besed, ki so dovoljene v dokumentu). To so lahko na primer vse besede v slovenskem jeziku ali bolj tehničen leksikon za določeno področje. Ta tehnika je lahko problematična, če dokument vsebuje besede, ki niso v leksikonu. Tesseract s svojim slovarjem vpliva na korak segmentacije znakov za večjo natančnost.

Izhod je lahko navadno besedilo ali datoteka znakov, vendar bolj izpopolnjeni OCR sistemi lahko ohranijo obliko besedila in ustvarijo PDF, ki vključuje izvorno sliko kot tudi besedilo.

Analiza najbližjega soseda lahko uporabi frekvenco sočasnih pojavov za odpravo napak, pri čemer ugotovi, da nekatere besede pogosto vidimo skupaj. Na primer »Washington D.C.« se veliko večkrat pojavi kot »Washington DOC« in s to tehniko lahko OCR odpravi to napako. Poznavanje slovnice skeniranega jezika lahko pomaga tudi ugotoviti, ali je beseda glagol ali samostalnik, kar omogoči večjo natančnost, saj lahko program skrbi, da je besedilo slovnično pravilno.

V nadaljnji optimizaciji OCR se velikokrat uporablja Levensteinov algoritem za razdaljo. Levenshteinova razdalja je nizovna metrika za merjenje razlike med dvema zaporedjema. Neformalna razdalja med dvema besedama je najmanjše število enoznakovnih popravkov (vstavkov, izbrisov ali zamenjav), potrebnih za spremembo ene besede v drugo.

4.5.4 Specificirana optimizacija za aplikacije

V zadnjih letih so glavni ponudniki tehnologije OCR začeli prilagajati sistem, da bi se učinkoviteje spoprijeli s posebnimi vrstami vhodnih podatkov. Poleg leksikona, ki je za vsako aplikacijo specifičen, lahko dosežemo boljše uspešnost z upoštevanje poslovnih pravil in bogatih informacij, ki jih vsebujejo barvne slike. Ta strategija se imenuje

aplikacijsko usmerjen OCR in je uporabljen za prevajanje registrskih tablic, računov, posnetkov zaslona, osebnih izkaznic, voznških dovoljenj in tudi v avtomobilski proizvodnji.

Dober primer je New York Times, ki je tehnologijo OCR prilagodilo v orodje Document Helper, ki omogoča njihovi ekipi novinarjev pospešeno obdelavo dokumentov, ki jih je treba pregledati. Ugotovili so, da jim omogoča obdelovanje kar 5400 strani na uro.

4.6 Alternativne rešitve

Obstaja več načinov reševanja problema prepoznavanja znakov z drugimi sredstvi, razen z izboljšanimi algoritmi OCR.

4.6.1 Forsiranje boljšega vnosa

Obstajajo posebni fonti, kot so OCR-A, OCR-B ali MICR, ki imajo natančno določeno velikost, razmik in značilno obliko znakov, to omogoča višjo stopnjo natančnosti med prepisovanjem pri obdelavi bančnih računov. Ironično pa je, da je bilo nekaj OCR algoritmov zasnovanih za zajemanje besedila v priljubljenih fontih, kot sta Arial ali Times New Roman, in ne morejo zajeti besedila v teh pisavah, ki so specializirane in se precej razlikujejo od priljubljenih pisav.

Vnaprej lahko tudi natisnemo polja, ki spodbujajo ljudi k bolj čitljivemu pisanju – en znak na polje. Ta polja so pogosto natisnjena v izstopajoči barvi, ki jo lahko OCR zlahka odstrani, kot rumena, rdeča ali oranžna.

Palm OS je uporabil poseben niz znakov, znakov kot »Grafiti«, ki so podobni tiskanim angleškemu znaku, vendar poenostavljeni ali spremenjeni za lažje prepoznavanje. Ampak uporabniki bi se morali naučiti, kako pisati v tej pisavi.

4.6.2 Crowdsourcing

Crowdsourcing ljudi za prepoznavanje znakov lahko hitro prevaja slike kot OCR, vendar z večjo natančnostjo za prepoznavanje slik, kot jo dobimo pri računalnikih. Finska nacionalna knjižnica je razvila spletni vmesnik, s katerim lahko uporabniki popravljajo besedila, ki jih je ustvaril OCR v standardiziranem formatu ALTO. Crowdsourcing se ne uporablja samo za neposredno prepoznavanje znakov, temveč tudi za povabilo razvijalcev programske opreme, da razvijejo algoritme za obdelavo slik na primer z uporabo tekmovalne teorije.

4.7 Natančnost

Ameriško ministrstvo za energijo je naročilo Inštitutu za raziskovanje informacijskih znanosti, naj poskuša izboljšati avtomatizirane tehnologije za prepoznavanje tiskanih dokumentov. Posledično so izvedli najbolj verodostojni test natančnosti OCR do leta 1996.

Ena raziskava, ki temelji na prepoznavanju strani časopisov iz 19. in zgodnjega 20. stoletja, je prišla do odkritja, da se natančnost OCR za komercialno programsko opremo razlikuje od 81 % do 99 %. Popolno natančnost lahko dosežemo s človeškim pregledom ali uporabo podatkovnega slovarja. Druga področja, vključno s prepoznavanjem rokopisa, kurzivne pisave in natisnjene besedila v drugih pisavah (zlasti azijski jeziki, ki imajo za

en znak veliko potez), so še vedno predmet aktivnih raziskav. Baza podatkov MNIST se običajno uporablja za testiranje sposobnosti sistemov za prepoznavanje ročno napisanih števil.

Stopnjo natančnosti je mogoče izmeriti na več načinov, način merjenja pa vpliva na stopnjo natančnosti. Če se na primer leksikon ne uporablja za popravilo programske opreme pri iskanju neobstoječih besed, je lahko stopnja napake znakov 1 %, ampak se poslabša 5 % ali še bolj, če meritev temelji na tem, ali je bila cela beseda prepoznana brez napačnih črk. Uporaba dovolj velikega nabora podatkov je tudi pomembna v sistemih za prepoznavanje rokopisa, ki temeljijo na nevronskih mrežah, po drugi strani pa je izdelava takšnih sistemov zapletena in dolgotrajna.

Spletni sistemi OCR pa so za prepoznavanje tiskanih črk postali precej znani kot komercialni izdelki. Zmožni so doseči stopnjo natančnosti od 80 % do 90 % za lepo, na roke napisano pisavo v tiskanih črkah. To naredi s pomočjo posebnega pisala, ki ga uporabnik premika od leve proti desni, kot bi bral besedilo. Vendar ta stopnja natančnosti še vedno pomeni na desetine napak na stran, zaradi česar je uporaba tehnologije zelo omejena.

Večina programov dovoli uporabnikom, da nastavijo stopnjo zaupanja, to pomeni, če program ne doseže določenega odstotka natančnosti, program obvesti uporabnika in ta lahko ročno pregleda dokument ter odpravi napake.

5 PROGRAM

5.1 Ideja o uporabi programa

Ideja najinega programa je omogočiti GOO parkiranje, brez možnosti zlorabe. Uporabnik bi se registriral na najino spletno stran, poslal svojo sliko (program potrdi, da je GOO) in vpisal podatke svoje kartice. Ti podatki bi bili nato shranjeni v bazo. Ko bi uporabnik prispel do parkirišča, bi bila tam kamera, ki bi pridobila invalidsko izkaznico uporabnika in njegovo registrsko tablico. Z OCR prebere ti dve sliki in rezultate primerja s podatki iz baze. Če se ujemajo, ga spusti naprej, v nasprotnem primeru pa mu ne pusti parkirati.

V nadaljevanju je predstavljena uporaba programa od spletne strani do prepoznave uporabnika na parkirišču.

5.1.1 Registracija

Uporabnik se najprej usmeri na najino spletno stran, kjer izbere možnost registracije. Pri postopku registracije poda svoje osebne podatke in na koncu registrira svoj račun.

Ustvarite račun!

Ime

Priimek

Email naslov

Geslo

Ponovi geslo

Datum rojstva: dd. mm. llll

Registriraj račun

[Ste pozabili geslo?](#)

[Že imate račun? Prijavite se!](#)

Slika 1: Registracija

5.1.2 Prijava

Po uspešno opravljeni registraciji se lahko uporabnik prijavi v svoj račun, kjer bo moral podati še nekaj dodatnih informacij glede svoje identitete in invalidske kartice.



Dobrodošli!

Vnesite svoj email naslov...

Vnesite svoje geslo

Prijava

[Ste pozabili geslo?](#)
[Ustvari račun!](#)

Slika 2: Prijava

5.1.3 Prepoznavna obraza

Preden lahko uporabnik vpiše podatke invalidske izkaznice, bo moral poslati sliko invalidske izkaznice in sliko svojega obraza. Program bo preveril, če se obraza osebe ujemata. Ko določi, da se obraza ujemata, to shrani v bazo in dovoli uporabniku, da vpiše podatke invalidske izkaznice.

5.1.4 Vpis podatkov invalidske izkaznice

Po uspešno poslanih slikah obraza in invalidske izkaznice mora uporabnik podati še dodatne informacije glede svoje invalidske izkaznice. Poleg svojih podatkov, ki se avtomatično izpišejo iz baze, mora vpisati serijsko številko ter datum veljavnosti kartice.

Dodajte podatke o karticil!

Ime Priimek

Datum rojstva:

Serijska št:

Veljavnost do:

Registrska št:

Dodaj podatke

Slika 3: Vpis podatkov

5.1.5 Shranjevanje v bazo

Vsi zbrani podatki se shranjujejo v mongoose podatkovni bazi, od koder se lahko ti podatki nato uporabijo pri OCR delu uporabe. Na sliki 4 pa je prikazana shema podatkovne baze.

```

const userSchema = new mongoose.Schema({
  name: { type: String, required: true, max: 100 },
  surname: { type: String, required: true, max: 100 },
  password: { type: String, required: true },
  email: { type: String, required: true },
  date_of_birth: {type: String, required: true},
  serial_number:{type: Number, required: true },
  expiration_date:{type: String, required: true},
  resetPasswordToken: {type: String, required: false},
  created: {
    type: Date,
    default: Date.now,
  },
});

```

Slika 4: Podatkovna baza

Zaradi evropske zakonodaje GDPR (Splošna uredba o varstvu podatkov) lahko uporabnik po vpisu osebnih podatkov v bazo zahteva izbris svojih osebnih podatkov. Vsak, ki bi želel izbris, lahko to doseže v najkrajšem možnem času.

5.1.6 OCR

Na parkirišču bo program čakal, da mu uporabnik pokaže svojo izkaznico. Ko zazna izkaznico, bo izvedel OCR na sliki in prebral podatke s kartice. Nato bo primerjal te podatke s tistimi, ki so v bazi. Če se ujemajo, bo prižgal kamero za zaznavanje registrske tablice. Ko zazna številko registrske tablice, jo primerja z vrednostjo v bazi. Če se tudi registrska številka ujema, potem program odpre zaporo in dovoli parkiranje, v nasprotnem primeru parkiranja ne dovoli.

5.2 Delovna sredstva spletne strani

Pri izdelavi spletne aplikacije sva uporabila veliko različnih programov in drugih orodij. Ti bodo v nadaljevanju kratko opisani.

5.2.1 Bootstrap

Bootstrap je močan pripomoček za delo s spletnimi stranmi. Bootstrap je zbirka orodij HTML, CSS in JavaScript za ustvarjanje in izdelavo spletnih strani ter spletnih aplikacij. Gre za brezplačen in odprtokoden projekt, ki ga gosti GitHub.

Bootstrap je prilagodljiv in enostaven za delo. Njegove glavne prednosti so, da je oblikovalsko odziven, ohranja široko združljivost brskalnika, ponuja dosledno zasnovano uporabo komponent, ki se lahko ponovno uporabljajo, poleg tega pa je zelo enostaven za uporabo. Ponuja bogato razširljivost z uporabo JavaScript, pri čemer ima vgrajeno podporo za vtičnike jQuery in programski jezik JavaScript API. Bootstrap se lahko uporablja s katerim koli integriranim razvojnim okoljem ali urejevalnikom ter s katero koli tehnologijo in jezikom na strani strežnika, od ASP.NET do PHP do Ruby on Rails.

Z uporabo Bootstrap-a se lahko spletni razvijalci osredotočijo na razvojno delo, ne da bi skrbeli za oblikovanje. Z njegovo uporabo lahko hitro postavijo lepo in dobro delujoče spletno mesto.

5.2.2 Node

Node.js je med platformirano odprto kodno okolje, ki ga poganja motor V8 JavaScript, ki je jedro Google Chroma. To omogoča, da je Node.js zelo zmogljiv.

Aplikacija Node.js se zažene v enem samem postopku, ne da bi za vsako zahtevo ustvarila novo nit. Node.js ponuja nabor asinhronih vhodno/izhodno primitivov v svoji standardni knjižnici, ki preprečujejo blokiranje kode JavaScript, na splošno pa so knjižnice v Node.js pisane po paradigmah, ki ne blokirajo, zaradi česar je vedenje blokade bolj izjema.

Ko mora Node.js izvesti vhodno/izhodno operacijo, kot je branje iz omrežja, dostop do baze podatkov ali datotečnega sistema, bo Node.js nadaljeval z delom, ko se odgovor vrne, namesto da bi blokiral nit in zapravljaj čakalne cikle procesorja.

To omogoča Node.js, da upravlja več tisoč sočasnih povezav z enim samim strežnikom, ne da bi bremenil upravljanje sočasnosti niti, kar bi lahko bil pomemben vir napak.

5.2.3 Mongo

Mongo ali MongoDB, kar pomeni Mongo Database, je program za baze podatkov. Klasificiran je kot NoSQL database program. Ima 4 izdaje: MongoDB Community Server, MongoDB Enterprise Server, MongoDB Atlas, MongoDB Stitch.

Uporabili smo MongoDB Atlas, ki je globalna storitev podatkovnih baz v oblaku za sodobne aplikacije. Uvedemo lahko popolnoma upravljan MongoDB prek AWS, Azure ali GCP. Avtomatizacija najboljšega razreda in preizkušene prakse zagotavljajo razpoložljivost, razširljivost in skladnost z najzahtevnejšimi standardi varnosti ter zasebnosti podatkov. Uporabimo lahko močan ekosistem gonilnikov, integracije in orodja MongoDB za hitrejšo gradnjo in manjšo porabo časa za upravljanje baze podatkov.

Podobno kot relacijske baze podatkov temeljijo na tabelah, tudi na dokumente usmerjene baze podatkov, kot je MongoDB, temeljijo na zbiranju dokumentov, pri čemer je vsak od teh dokumentov sestavljen iz atributov ključ/vrednost. Posamezen dokument je lahko enakovreden vrstici v tabeli, pri čemer je vsak ključ podoben imenu stolpca in vrednost vsakega ključa je podobna vrednosti posamezne vrstice. Glavna razlika je v tem, da dokument ni omejen na določeno shemo ali stolpce znotraj tabele. Dva dokumenta bi lahko delila podobne elemente, kot je ID polja, pa tudi popolnoma drugačne elemente.

Ker MongoDB omogoča dinamične spremembe sheme, je enostavno spremeniti spremembe, ne da bi bilo treba ponovno oblikovati obstoječo bazo podatkov, da bi podprli nova polja, na primer dodajanje novega izdelka zalog s svojim edinstvenim naborom atributov. Poleg tega hierarhija dokumentov zlahka preslika hierarhije v aplikacijski kodi, kar poenostavi postopke ustvarjanja, branja, posodabljanja in brisanja.

5.2.4 Ajax

AJAX pomeni **A** sinhrono **J**a vaScript in **X** ML. AJAX je nova tehnika ustvarjanja boljših, hitrejših in bolj interaktivnih spletnih aplikacij s pomočjo XML, HTML, CSS in Java Script. Običajna spletna aplikacija prenaša informacije v in iz groba s sinhronimi zahtevami. To pomeni, da izpolnite obrazec, pritisnete oddajo in se usmerite na novo stran z novimi informacijami s strežnika. Ko AJAX ob pritisku na oddajo JavaScript vloži zahtevo

strežniku, razloži rezultate in posodobi trenutni zaslon. V najčistejšem smislu uporabnik nikoli ne bi vedel, da se je karkoli preneslo na strežnik.

Ajax ni ena sama tehnologija, temveč skupina skupin tehnologij. HTML in CSS se lahko uporabljata v kombinaciji za označevanje in slog podatkov. Spletno stran lahko nato JavaScript spremeni, tako da dinamično prikaže in omogoči interakcijo uporabnika z novimi informacijami. Vgrajeni objekt XMLHttpRequest ali od leta 2017 nova funkcija "fetch ()" znotraj JavaScript se običajno uporablja za izvajanje Ajaxa na spletnih straneh, ki omogoča spletnim mestom nalaganje vsebine na zaslon, ne da bi osvežili stran. Ajax ni nova tehnologija ali drug jezik, ampak je le obstoječa tehnologija, ki se uporablja na nov način.

5.2.5 EJS

EJS je preprost jezik za predloge, ki omogoča ustvarjanje označevanja HTML z navadnim JavaScriptom.

EJS pomeni embedded ali vgrajeni JavaScript. Omogoča nam vdlati kodo JavaScript (spremenljivke, if stavek, zanke) v naš HTML. Uporablja se v node.js, ko deluje s programom Express kot predloga za predloge, da pomaga pri upodabljanju kode JavaScript na strani odjemalca.

5.2.6 Npm

Npm je upravitelj paketov za platformo Node JavaScript. Njegova naloga je, da postavi module, da jih node najde in v nadaljevanju inteligentno upravlja s konflikti odvisnosti.

Je izjemno nastavljiv za podporo najrazličnejših primerov uporabe. Najpogosteje se uporablja za objavljanje, odkrivanje, namestitvev in razvoj node programov.

Npm je sestavljen iz ukazne vrstice, ki deluje z oddaljenim registrom. Uporabnikom omogoča porabo in distribucijo modulov JavaScript, ki so na voljo v registru. Paketi v registru so v obliki CommonJS in vključujejo datoteko z metapodatki v obliki JSON. V glavnem registru npm je na voljo več kot 477.000 paketov. V registru ni postopka preverjanja za oddajo, kar pomeni, da so tam najdeni paketi nizke kakovosti, nevarni ali zlonamerni. Namesto tega se npm zanaša na poročila uporabnikov, da odstrani pakete, če kršijo pravilnike, ker so nizke kakovosti, negotovi ali zlonamerni.

Npm pa sodeluje tudi z zelo znanim in trenutno popularnim podjetjem GitHub.

5.2.7 Express

Express je priljubljen spletni okvir, ki je napisan v JavaScriptu in je hostan znotraj okolja Node.js. Express je minimalen in prilagodljiv okvir spletnih aplikacij Node.js, ki nudi močan nabor funkcij za razvoj spletnih in mobilnih aplikacij. Omogoča hiter razvoj spletnih aplikacij na nodu. Nekatere glavne značilnosti okvira Express: omogoča nastavitve srednjih vmesnikov za odzivanje na zahteve HTTP, določi tabelo usmerjanja, ki se uporablja za izvajanje različnih dejanj na podlagi metode HTTP in URL-ja, omogoča dinamično upodabljanje HTML strani na podlagi posredovanja argumentov v predloge.

5.2.8 Heroku

Najina stran se nahaja na Heroku strežniku pod brezplačno domeno. Gostovanje je povezano z Github računom, v katerem se nahaja repozitorij z vsemi potrebnimi mapami in datotekami. Poleg gostovanja pa Heroku omogoča še mnogo več kot so npr. podatkovne baze, integrirano okolje za razvoj, mnoge dodatke povezane z programskimi jeziki in bazami, itd.

5.3 Prepoznavna obraza

Preden bo uporabnik vnesel podatke svoje kartice bo zahtevano, da pošlje svojo sliko in tudi sliko svoje invalidske izkaznice. To mora poslati zato, da lahko s pomočjo prepoznavanja obraza program preveri, ali sta oseba na sliki in na izkaznici ista oseba. S tem korakom zagotovimo to, da je uporabnik gibalno ovirana oseba, saj če ni, ne bo mogel priložiti invalidske izkaznice.

Uporabnik bo lahko svoje podatke poslal samo po tem, ko bo program za prepoznavanje vrnil vrednost true, kar pomeni, da se obraza na slikah ujemata.

Potem ko uporabnikove slike pridobi, jih spremeni v črno-bele in zmanjša na določeno ločljivost, lahko program primerja obraza. Program izračuna lokacijo izkaznice in slike osebe ter nariše kvadrat okoli njihovega obraza, namen tega je vizualna predstava, kaj primerja program. Obraza primerja s funkcijo `compare_faces`, primerja pa kodirani številki obrazov (knjižnica `face_recognition` pretvori slike v kodirano število). Na koncu še napiše rezultat primerjanja (true ali false) in številko, ki pove ujemanje obrazov. Na sliki 5 je koda, ki kodira obraza, ju primerja in izpiše na ekran.

```
cardLoc = face_recognition.face_locations(imgCard)[0]
encodeCard = face_recognition.face_encodings(imgCard)[0]
cv2.rectangle(imgCard, (cardLoc[3], cardLoc[0]), (cardLoc[1], cardLoc[2]), (0, 0, 255), 2)

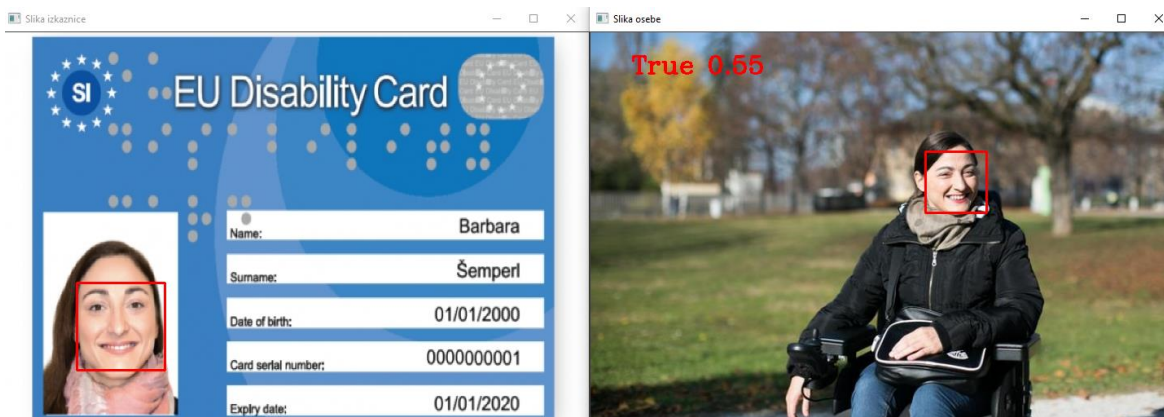
personLoc = face_recognition.face_locations(imgPerson)[0]
encodePerson = face_recognition.face_encodings(imgPerson)[0]
cv2.rectangle(imgPerson, (personLoc[3], personLoc[0]), (personLoc[1], personLoc[2]), (0, 0, 255), 2)

results = face_recognition.compare_faces([encodeCard], encodePerson)
faceDis = face_recognition.face_distance([encodeCard], encodePerson)
print(results, faceDis)
cv2.putText(imgPerson, f'{results[0]} {round(faceDis[0], 2)}', (50, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2)

cv2.imshow('Slika izkaznice', imgCard)
cv2.imshow('Slika osebe', imgPerson)
cv2.waitKey(0)
```

Slika 5: Koda za prepoznavanje obraza

Na sliki 6 je primer izpisa na ekran. Na levi je slika izkaznice, na desni pa slika osebe. Na sliki osebe je izpisan rezultat primerjanja in ujemanje obrazov.



Slika 6: Rezultat prepoznavanja obraza

5.4 Prepoznavanje uporabnika z OCR

Končni program bo prepoznal gibalno ovirano osebo po njegovi invalidski izkaznici in registrski tablici. To bo naredil tako, da bo pridobil vse uporabnike iz baze in primerjal njihove podatke s tistimi, ki jih dobi OCR od prepoznavanja uporabnikovih podatkov. Če se podatki na izkaznici ujemajo, bo lahko prebral tudi registrsko številko in jo primerjal s to, ki se nahaja v bazi. Tako bova dosegla dvonivojsko avtentikacijo, saj mora oseba imeti invalidsko izkaznico, ki je v bazi in tudi biti v avtomobilu, od katerega registrska tablica je vpisana v bazi. Če se vsi podatki ujemajo, program vrne true in osebi dovoli parkirati, v nasprotnem primeru pa osebi ne dovoli parkiranja.

Prepoznavanje izkaznice in registrske tablice deluje na enak način, edina razlika je v tem, da pri registrski tablici želimo pridobiti le eno vrednost, pri izkaznici pa več, zato se samo zadnji korak razlikuje.

Prvi korak je pretvoriti sliko v črno-belo, to storimo, ker naslednji korak deluje bolje s črno-belo sliko. V tem koraku tudi preberemo sliko s cv2, kar nam omogoča nadaljnje spreminjanje te slike.

V naslednjem koraku sliko zameglimo, da se znebimo podrobnosti, kar izboljša kvaliteto branja slike. Za tem pa zaznamo robove na sliki. To storimo s pomočjo algoritma Canny, z uporabo cv2.

Tretji korak je to, da najdemo oblike v sliki. To naredimo s funkcijo findContours, ki nam jo omogoči cv2. Ta funkcija vrne preproste oblike, tako da lahko pričakujemo kvadrat za našo izkaznico in tudi za registrsko tablico. Nato te oblike shranimo s pomočjo knjižnice imutils. Te oblike moramo tudi razvrstiti. To naredimo glede na njihovo površino. Ko imamo najboljše kandidate za obrobo kartice ali pa registrske tablice, moramo preveriti, če je oblika štirikotnik, saj sta oba predmeta, ki ju želiva iz slike pridobiti, tudi štirikotnika. Tako lahko preverimo, če je oblika štirikotnik, potem lahko program nadaljuje, če pa ne najde štirikotnika, pa vrne napako.

Nadaljujemo tako, da naredimo masko in ločimo kartico ali registrsko tablico od ostale slike tako, da ozadje pobarvamo črno. Tako ločimo del slike, ki ga želimo prebrati, od nepotrebnih podatkov.

Do zdaj imamo samo sliko na črnem ozadju, zato jo je potrebno obrezati, da vsebuje samo sliko, ki smo jo prepoznali.

Zdaj lahko končno izvedemo prepoznavanje znakov. To sva poskusila z uporabo

pytesseract in tudi easyocr. Obrezana slika je poslana OCR-ju in ta vrne podatke, ki so dvodimenzionalni seznam, na prvem mestu imamo koordinate prepoznanih znakov, nato imamo prepoznano besedilo in na koncu natančnost, ki nam v našem primeru pomaga.

Zadnji korak je izpis podatkov, pri registrski tablici je precej preprosto, saj ima samo en podatek, pri kartici je pa malo bolj kompleksno, saj je potrebno prebrati vse besedilo, ki ga vsebuje. Program s cv2 izriše zelen okvir okoli obrezane slike in z zelenim besedilom izpiše tekst, ki ga vsebuje. Če pa vsebuje notranje podatke, okoli teh nariše rdeče okvirje in nad njih izpiše prebrane podatke. Rezultate tudi filtrira v seznam, tako da pridobimo samo zelene podatke.

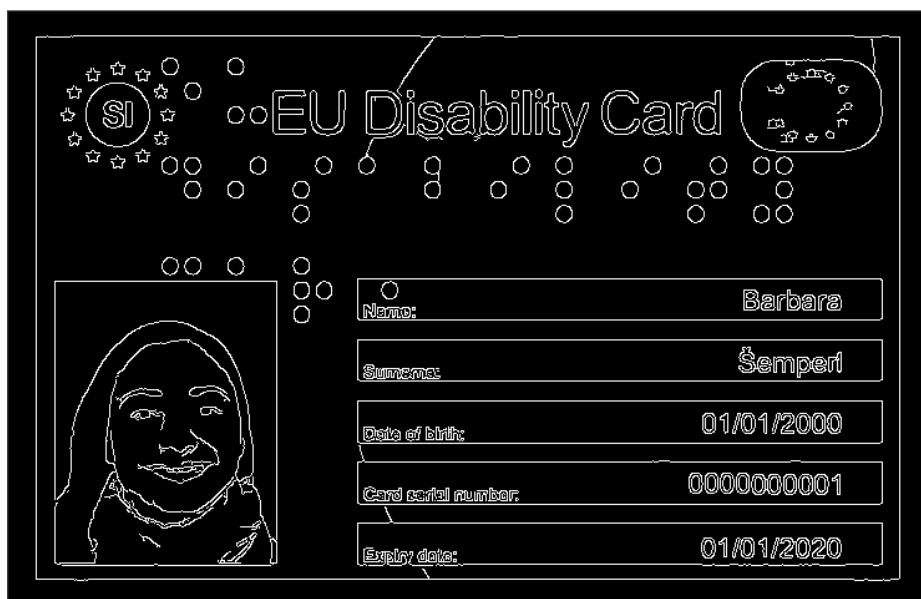
5.4.1 Prepoznavanje invalidske izkaznice

Na sliki 7 vidimo sliko, ki je pretvorjena v črno-belo.



Slika 7: Črno-bela izkaznica

Na sliki 8 vidimo filtrirano sliko s algoritmom Conny, tako da je lažje videti oblike.



Slika 8: Filtrirana slika izkaznice

Na sliki 9 vidimo izkaznico, ki je bila prepoznana kot štirikotnik in ozadje, ki je bilo zamenjano s črno barvo.



Slika 9: Odstranjevanje ozadja izkaznice

Na sliki 10 vidimo obrezano sliko. Ta slika je brez ozadja in pripravljena za branje z OCR.



Slika 10: Obrezana slika izkaznice

Na sliki 11 vidimo rezultat branja izkaznice. Okoli zaznane izkaznice je narisana zelena obroba in okoli prepoznanih podatkov je narisana rdeča obroba z rdečim besedilom.



Slika 11: Rezultat OCR izkaznice

Na sliki 12 je prikazan seznam rezultatov, ki ga vrne OCR in pod njim filtriran seznam, ki vsebuje pomembne podatke.

```
[[[50, 48], [84, 48], [84, 78], [50, 78]], 'SI', 0.8357054286733868), ([[183, 29], [559, 29], [559, 98], [183, 98]], 'EU Disability Card', 0.8127274572746234), ([[261, 213], [309, 213], [309, 229], [261, 229]], 'Name:', 0.9977981940438 986), ([[564, 198], [650, 198], [650, 224], [564, 224]], 'Barbara', 0.9999981977244236), ([[261, 263], [329, 263], [32 9, 279], [261, 279]], 'Surname:', 0.992669379927074), ([[559, 245], [650, 245], [650, 277], [559, 277]], 'Semperl', 0.9 98201933772613), ([[263, 311], [349, 311], [349, 327], [263, 327]], 'Date of birth:', 0.6136602700518423), ([[530, 296 ], [650, 296], [650, 322], [530, 322]], '01/01/2000', 0.822728347756123), ([[263, 361], [391, 361], [391, 377], [263, 377]], 'Card serial number:', 0.8258353575027362), ([[520, 346], [650, 346], [650, 372], [520, 372]], '0000000001', 0. 7606784612735847), ([[258, 405], [342, 405], [342, 429], [258, 429]], 'Expiry date:', 0.7173901778907255), ([[530, 396 ], [650, 396], [650, 422], [530, 422]], '01/01/2020', 0.510334998001776)] ['SI', 'EU Disability Card', 'Barbara', 'Semperl', '01/01/2000', '0000000001', '01/01/2020']
```

Slika 12: Koordinate izkaznice

5.4.2 Prepoznavanje registrske tablice

Na sliki 13 vidimo sliko avta, ki je pretvorjena v črno-belo.



Slika 13: Črno-bela slika avta

Na sliki 14 vidimo filtrirano sliko s algoritmom Conny, tako da je lažje videti oblike.



Slika 14: Filtrirana slika avta

Na sliki 15 vidimo tablico, ki je bila prepoznana kot štirikotnik, in ozadje, ki je bilo zatemnjeno.



Slika 15: Odstranjevanje ozadja registrske tablice

Na sliki 16 vidimo obrezano sliko. Ta slika je brez ozadja in pripravljena za branje z OCR.



Slika 16: Obrezana slika registrske tablice

Na sliki 17 vidimo rezultat branja tablice. Okoli zaznane tablice je narisana zelena obroba in pod njo je izpisana registrska številka.



Slika 17: Rezultat OCR registrske tablice

Na sliki 18 je prikazan seznam rezultatov (v tem primeru samo eden), ki ga vrne OCR in pod njim filtriran seznam, ki vsebuje registrsko številko.

```
[[[[[0, 0], [244, 0], [244, 53], [0, 53]], 'H982 FKL', 0.9757375847177501]]  
['H982 FKL']
```

Slika 18: Koordinate registrske tablice

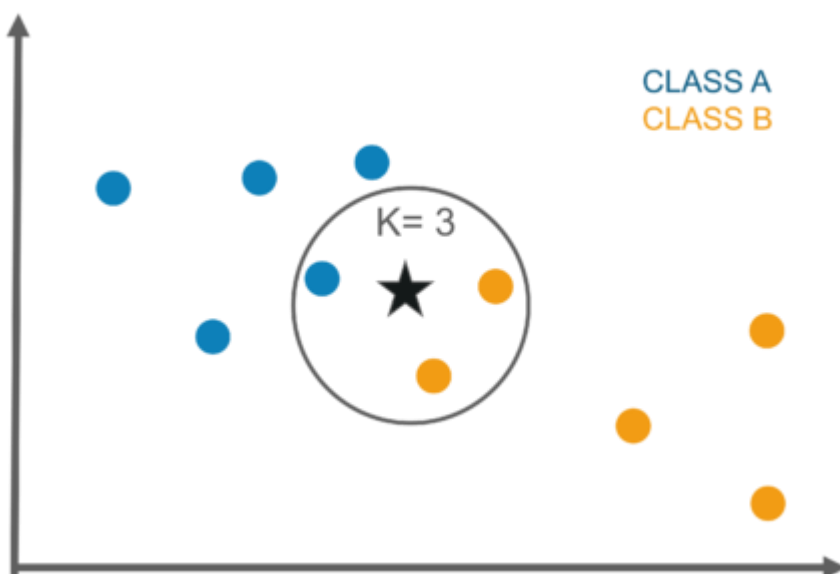
6 IZDELAVA OCR IN PRIMERJAVA S TESSERACT

Poleg same uporabe knjižnic `pytesseract` in `easyocr` za opravljanje OCR naju je zanimalo, kako deluje OCR in če bi lahko zamenjala knjižnico s svojim OCR, ki bi deloval z dovolj veliko natančnostjo, da bi ga lahko uporabila za prepoznavanje uporabnikovih podatkov.

6.1 Izdelava OCR

Ugotovila sva, da je OCR zelo težko realizirati z najinim trenutnim znanjem, saj nimava izkušenj na področju umetne inteligence in strojnega učenja, zato sva se odločila za najpreprostejši način opravljanja OCR in to je z uporabo algoritma K-najbližjih sosedov (KNN) za prepoznavanje s primerjanjem slikovnih pik.

K-najbližjih sosedov je metoda strojnega učenja, ki shranjuje učne primere in ko dobi nov primer, algoritem poišče »najbližje sosede«, to pomeni tiste, ki so mu najbolj podobni in neznanemu elementu tako določi razred, ki mu pripada največje število sosedov. Na sliki 19 lahko vidimo primer rezultata algoritma KNN. Zvezda predstavlja neznan element in v krogu so zajeti trije najbližji sosedi in ker je večina elementov razreda B, algoritem določi, da je tudi neznan element razreda B. Za ta algoritem je najbolj priročno izbrati liha števila za K-je, saj ne moremo dobiti enakega števila sosedov razreda A in B.



Slika 19: Primer algoritma KNN

Najin program bo uporabljal bazo MNIST, ki je zbirka ročno napisanih števil, ki vsebuje 60000 unikatno napisanih števil od nič do devet. Poleg števil ima tudi seznam njihovih vrednosti in tudi števila, primerna za testiranje (ta števila imajo tudi rezultate, ampak ti bodo uporabljeni le za ugotavljanje, kje se je program zmotil in za izračun natančnosti programa).

Vsaka slika številke je črno-bela, velikosti 28 x 28 bitov in poravnana na sredino, to omogoči zelo preprosto prepoznavo, saj slik ni potrebno predčasno procesirati za boljšo natančnost. Na sliki 20 je nekaj primerov MNIST števil.



Slika 20: Baza MNIST

Prva stvar, ki jo je potrebno narediti, je prebrati učne podatke iz dokumenta. To je zelo preprosto, saj je ta strukturiran zelo logično. Prvih 32 bitov nam pove tip datoteke, naslednjih 32 bitov je število slik, to je v tem primeru 60000. Sledi število stolpcev in vrstic, ki tudi vsak zasedeta 32 bitov, v našem primeru sta oba 28, saj je to dimenzija naše slike. Vsi biti, ki sledijo, so pike slik. Datoteka imen je podobna, le da ne vsebuje dimenzij slik, tako da tip datoteke, število imen in nato podatki.

Zdaj imamo podatke, ki so shranjeni v dvodimenzionalnih seznamih (stolpci in vrstice). Te podatke je zdaj potrebno spremeniti v samo en vektor slikovnih pik. Tako naš seznam slik spremenimo iz dvodimenzionalnega seznama slik in stolpcev v enodimenzionalnega slik (saj so te dimenzije 1 x 784, namesto 28 x 28). Na sliki 21 je prikazano, kako sva dosegla enodimenzionalni seznam. Spremenljivka `X_train` predstavlja dvodimenzionalni seznam, ki ga dobi funkcija `extract_features` kot `X`, ta funkcija pokliče funkcijo `flatten_list`, ki pa prejeme stolpce kot spremenljivko `l` in vrne pike.

```
def flatten_list(L):  
    return [pixel for sublist in l for pixel in sublist]  
  
def extract_features(X):  
    return [flatten_list(sample) for sample in X]  
  
X_train = extract_features(X_train)
```

Slika 21: Spreminjanje slik v enodimenzionalne slike

Zdaj lahko izvedemo algoritem KNN. Ta algoritem bo prejel parametre `X_train` (učni podatki), `Y_train` (imena učnih podatkov), `X_test` (neznani podatki, ki jih mora algoritem ugotoviti) in `k`, ki je avtomatsko nastavljen na 3. `K` je hiperparameter, ki določa število sosedov, ki bodo določali vrednost neznanih podatkov. Na sliki 22 lahko vidimo metodo `knn`, ki izvede ta algoritem.


```

def knn(X_train, Y_train, X_test, k=3):
    Y_pred = []
    for test_sample_idx, test_sample in enumerate(X_test):
        print(test_sample_idx, end=' ', flush=True)
        training_distances = get_training_distance(X_train, test_sample)
        sorted_distance_indices = [pair[0] for pair in sorted(enumerate(training_distances), key=lambda x: x[1])]
        candidates = [Y_train[idx] for idx in sorted_distance_indices[:k]]
        best_candidate = get_most_frequent_element(candidates)
        Y_pred.append(best_candidate)
    print()
    return Y_pred

```

Slika 22: Metoda za izvajanje KNN

Na začetku naredimo seznam Y_pred. V ta seznam bo algoritem shranil rezultate (vrednosti, ki jih bo algoritem določil neznanim podatkom). Program bo z for zanko prehajal skozi vse neznane podatke in bo na začetku izpisal indeks trenutnega podatka, zato da lahko vidimo, na kateri številki je program (algoritem KNN je zelo počasen, saj mora primerjati vse učne podatke z neznanim in teh učnih primerov je lahko v našem primeru 60000). Nato pokliče metodo get_training_distances, ki vrne razdaljo vseh učnih podatkov od trenutnega testnega podatka.

Ta metoda pa pokliče metodo dist, ki izračuna razdaljo od ene učne slike do testne. Ta metoda deluje s formulo: $d(p, q) = \sqrt{(p - q)^2}$. Glavni program dobi nazaj razdalje vseh učnih podatkov od neznanega podatka. Ti metodi lahko vidimo na sliki 23.

```

def dist(x, y):
    return sum([
        (bytes_to_int(x_i) - bytes_to_int(y_i)) ** 2
        for x_i, y_i in zip(x, y)
    ]) ** (0.5)

def get_training_distance(X_train, test_sample):
    return [dist(train_sample, test_sample) for train_sample in X_train]

```

Slika 23: Metodi za izračun razdalje

Naslednji korak je sortiranje razdalj elementov v nov seznam ter pridobiti kandidate. Candidates je seznam elementov, ki so najbližje neznanemu podatku, njegova dolžina je odvisna od k. Zadnja stvar, ki jo je potrebno narediti, je izbrati tistega kandidata, ki se največkrat pojavi. To program naredi s funkcijo max, za število ponovljenih vrednosti. Na koncu funkcija doda kandidata na seznam Y_pred in ta seznam tudi vrne, ko konča zanko.

```

Dataset: mnist
n_train: 5000
n_test: 7
k: 5
0 1 2 3 4 5 6
Predicted labels: [7, 2, 1, 0, 4, 1, 4]
Accuracy: 100.0%

Process finished with exit code 0

```

Slika 24: Rezultat OCR

Na sliki 24 vidimo izpis, ki ga program vrne, potem ko izvede KNN. Na začetku izpiše osnovne podatke, to so: knjižnica, število učnih podatkov, število neznanih testnih podatkov in k (število sosedov). Nato izpisuje indeks za vsako številko, ki jo je že obdelal. Na koncu pa izpiše seznam števil in tudi natančnost, ki jo izračuna z uporabo seznama `Y_test`. Do sedaj ta OCR dela brez vključevanja knjižnic, ampak sva na koncu vključila 2 knjižnici. To sta Pillow in Numpy, ti knjižnici ne pomagata pri prepoznavanju števil, ampak omogočita vizualno prikazati črke, ki so bile prepoznane.

```
write_image = True

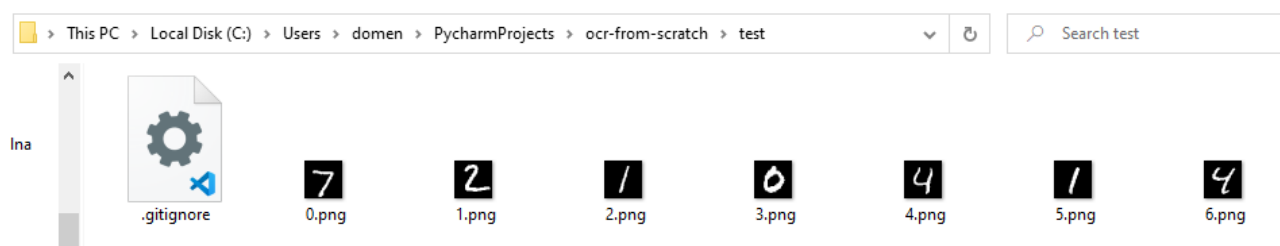
if write_image:
    from PIL import Image
    import numpy as np

    def read_image(path):
        return np.asarray(Image.open(path).convert('L'))

    def write_image(image, path):
        img = Image.fromarray(np.array(image), 'L')
        img.save(path)
```

Slika 25: Koda za shranjevanje slik


Na sliki 25 je prikazan del, ki vključi ti dve knjižnici. Njegova naloga je, da slike, ki so za program neznanе, shrani v mapo test, zato da lahko tudi grafično vidimo, da so številke, ki jih je uganil OCR, res pravilne. Vsebuje tudi spremenljivko `write_image`, ki jo lahko nastavimo na `true` ali `false` in v primeru, da je `false`, program ne bo shranil slik v mapo test. Rezultat te kode lahko vidimo na sliki 26.



Slika 26: Prepoznane slike

OCR lahko tudi prepozna števila iz slik, ki jih naredi uporabnik, potrebno je le poskrbeti, da je slika v png formatu, da je velikost slike 28 x 28 in da je številka bele barve na črnem ozadju. Na spodnjih slikah je koda, ki shrani sliko in številka, ki jo prepozna program. `X_test` je nastavljen na pot slike, `Y_test` pa na vrednost slike (samo za preverjanje).

```
X_test = [read_image(f'{DATA_DIR}our_test.png')]
Y_test = [8]
```



Slika 27: Prepoznavanje najine številke

6.2 Tesseract OCR

Python-Tesseract je orodje, ki omogoča prepoznavanje besedila iz slik. Narejen je po Googlovem Tesseract OCR orodju. Podpira vse slikovne datoteke, ki jih podpirajo druge knjižnice za delo z dokumenti kot Pillow. To so: jpeg, png, gif, bmp, tiff, itd. Tako da različni tipi slik niso problem za ta OCR. Beleženje rezultatov omogoča v isti program ali pa v novo datoteko.

Pytesseract je zelo preprost za uporabo in zahteva zelo malo kode, da prebere besedilo slike. Na sliki 28 je primer vključitve knjižnic pytesseract in cv2, s pomočjo katerih lahko preberemo besedilo slike in tudi grafično prikažemo rezultat.

```
import cv2
import pytesseract
pytesseract.pytesseract.tesseract_cmd = 'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'
img = cv2.imread('izkaznica2.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
print(pytesseract.image_to_string(img))
```

Slika 28: Vključevanje pytesseract

Potem ko vključimo knjižnici, je potrebno določiti pot od tesseract.exe in zdaj lahko kličemo metodo `image_to_string` ter mu podamo sliko, on pa nam vrne besedilo. Cv2 na zgornjem primeru samo pretvori sliko v RGB, saj tesseract zahteva RGB, cv2 pa BGR.

Na sliki 29 je nadaljevanje programa, kjer tesseract zazna besede, nato pa cv2 izpiše sliko, okoli zaznanih besed nariše rdeč kvadrat in tudi nad besede izpiše prebrane črke.

```
hImg, wImg, t = img.shape
boxes = pytesseract.image_to_data(img)
for x, b in enumerate(boxes.splitlines()):
    if x != 0:
        b = b.split()
        if len(b) == 12:
            x,y,w,h = int(b[6]), int(b[7]), int(b[8]), int(b[9])
            cv2.rectangle(img, (x, y), (w+x, h+y), (0,0,255), 1)
            cv2.putText(img, b[11], (x, y), cv2.FONT_HERSHEY_COMPLEX, 1, (50, 50, 255), 2)
```

Slika 29: Koda za izris okvirja in besedila

Na začetku program pridobi višino, širino in t, ki ni pomemben. Pytesseract nato pretvori sliko v podatke, ki so shranjeni v spremenljivko `boxes`, skozi katero ponavlja zanko. Vsak element `b` razrežemo z metodo `split()`, kar ustvari seznam z 12 elementi v primeru, da je OCR zaznal besedilo. Če besedila ni, tudi ni zadnjega elementa. To pomeni, da če program zazna besedo, nadaljuje z izvajanjem, če pa ne zazna besede, pa skoči na naslednjo besedo. Ko loči besede, pa s cv2 izriše kvadrat okoli besede in tudi to besedo izpiše nad kvadrat. Primer izpisa lahko vidimo na sliki 30.

Mild Splendour of the various-vested Night!
 Mother of wildly-working visions! hail!
 I watch thy gliding, while with watery light
 Thy weak eye glimmers through a fleecy veil;
 And when thou lovest thy pale orb to shroud
 Behind the gather'd blackness lost on high;
 And when thou dartest from the wind-rent cloud
 Thy placid lightning o'er the awaken'd sky.

Slika 30: Rezultat pytesseract

Kot vidimo na sliki, je OCR pravilno prebral vse besede, kar ni presenetljivo glede na to, da je slika primerna za branje. Edina stvar, ki ni pravilna na rezultatu, je to, da je namesto opuščajev program izpisal vprašaje. To je napaka knjižnice cv2 in ne tesseract, kar pa tudi ne bo problem za nas, saj bomo brali podatke v slovenskem jeziku, ki nima tega znaka. Na sliki 31 lahko vidimo izpis v konzoli, ki dokazuje, da je tesseract pravilno prebral sliko.

```
Mild Splendour of the various-vested Night!  

Mother of wildly-working visions! hail!  

I watch thy gliding, while with watery light  

Thy weak eye glimmers through a fleecy veil;  

And when thou lovest thy pale orb to shroud  

Behind the gather'd blackness lost on high;  

And when thou dartest from the wind-rent cloud  

Thy placid lightning o'er the awaken'd sky.
```

Slika 31: Rezultat branja v konzoli

Ta primer prikazuje branje besed, ampak tesseract ima več možnosti branja. Lahko tudi bere znake, številke ali pa samo eno številko. Ti postopki branja so zelo podobni prikazanem, razlika je samo v seznamu podatkov, ki jih vrne tesseract. Pri besedah vrne 12-mestni seznam, pri znakih pa samo 4-mestni, zato je program potrebno prilagoditi na te podrobnosti.

6.3 Primerjava OCR-jev

Najin OCR deluje, ampak ima veliko problemov. Največji problem je to, da zahteva zelo specifično obliko podatkov, slike znakov morajo biti velike 28 x 28, slika mora biti črno-bela, znak mora biti na sredini slike, slika mora biti ostra itd. Ampak kljub vsem tem pogojem program ne prebere vseh znakov pravilno in tudi rabi zelo veliko časa z večjim številom učnih slik.

Algoritem sva poskusila zagnati s 10000 učnih slik in s 100 testnimi slikami z namenom, da izračunava natančnost programa. Pod temi pogoji je imel natančnost 94 %, za to, da je prebral vse številke, pa je potreboval približno 10 minut.

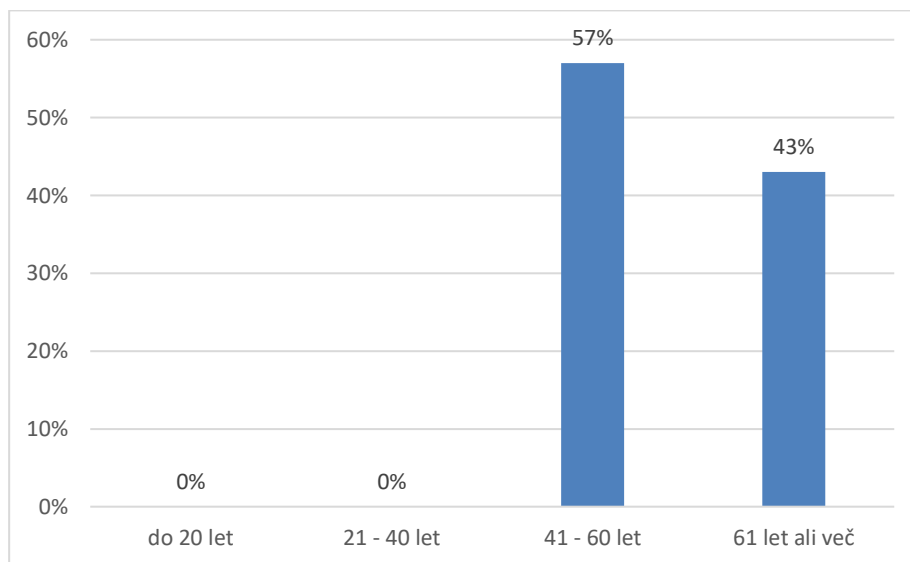
Tesseract nima nobene od teh pomanjkljivosti, sliko prebere hitro, z večjo natančnostjo in tudi nima tako veliko zahtev kot najin program. Če samo vključiva tesseract, se ni več potrebno ukvarjati s tem, kako bova dobila rezultate, saj je to preprost klic metode. Zaradi teh razlogov sva v najin program vključila izvedbo z uporabo knjižnice tesseract.

7 ANALIZA ANKETE

7.1 Izbor in struktura vzorca anketiranih

Raziskavo smo izvedli s pomočjo spletnega vprašalnika. V raziskavi je sodelovalo 75 gibalno oviranih oseb. Anketa je bila poslana vsem društvom GOO v Sloveniji.

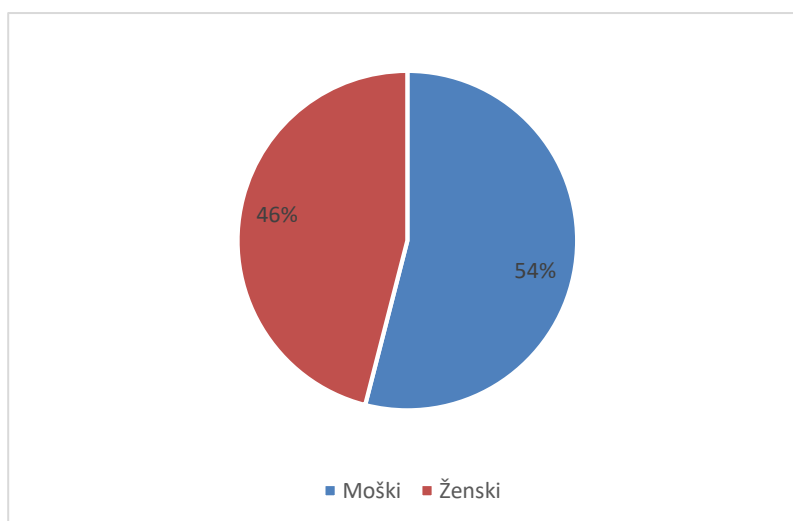
Starost:



Graf 1: Starost

Anketa je bila namenjena vsem starostnim skupinam nad 18 let, saj nimajo izkušenj s parkiranjem, ampak rezultati pokažejo, da so vsi anketiranci stari 40 ali več, kar potrjuje najino hipotezo.

Spol:

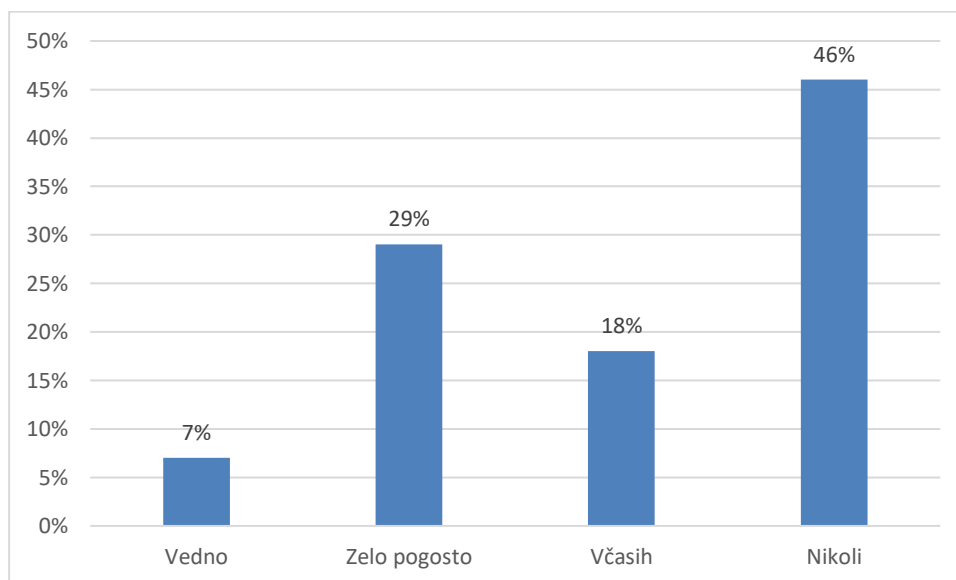


Graf 2: Spol

Med anketiranci sta bila enakomerno zastopana oba spola.

7.2 Rezultati primarne raziskave

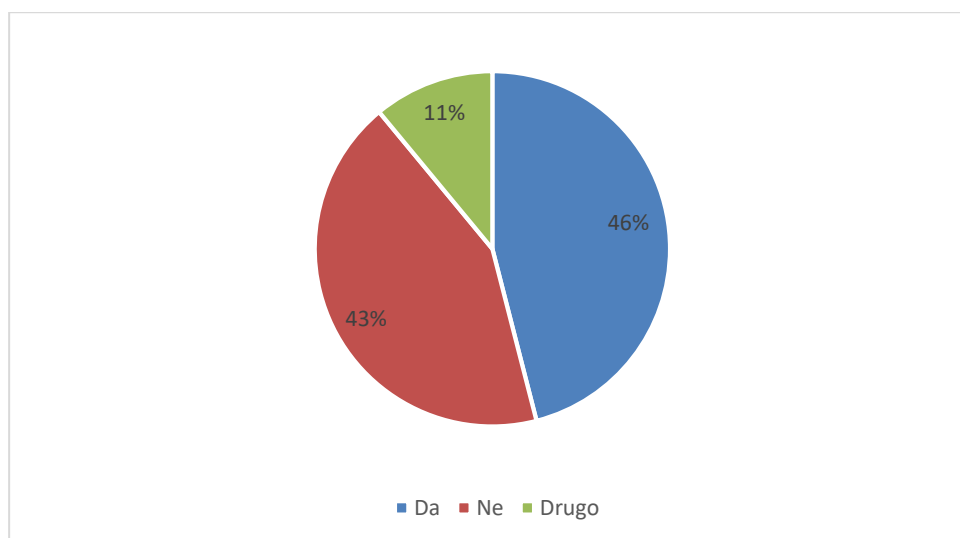
Kako pogosto parkirate na parkirnih mestih za GOO?



Graf 3: Pogostost parkiranja na mestih za GOO

Največ anketirancev (kar 46 %) je izjavila, da sploh ne parkira na parkirnih mestih za GOO, kar naju je presenetilo. Veliko anketirancev pa je izjavilo, da zelo pogosto parkirajo na mestih za GOO.

Ali ste zadovoljni s številom parkirnih mest za GOO v svoji okolici?

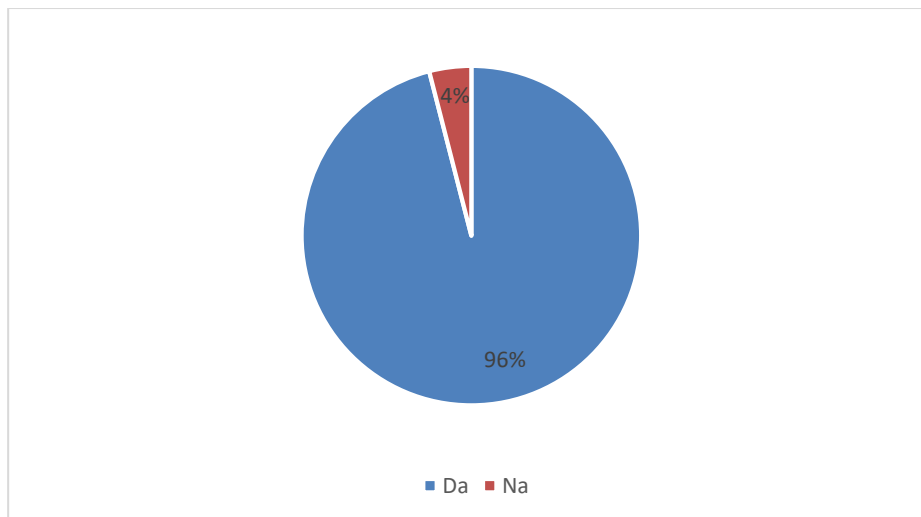


Graf 4: Zadovoljstvo s številom parkirnih mest

Iz grafa je razvidno, da je približno polovica anketirancev zadovoljna s številom mest za

GOO, pod drugo so anketiranci sporočili, da so zadovoljni z mesti za GOO, ampak bi jih bilo lahko tudi več.

Ali se vam zdi najin program uporaben?



Graf 5: Mnenje o uporabnosti programa

Anketirancem sva razložila najino idejo za program, ki bi lahko rešil problem parkiranja za GOO. Večina anketirancev, in sicer 96 % se strinja, da bi bil najin program uporaben. Pozitiven odziv naju je presenetil.

Pojasnite zakaj ste izbrali da/ne.

Anketirancem sva postavila tudi neobvezno vprašanje, saj naju zanima, zakaj se jim zdi program uporaben. Mnenje anketirancev je, da so pozitivne strani programa to, da je smiselna, uporabna rešitev, prihrani čas, dostop do parkirišča bi bil nemoten, zloraba bi bila manjša, mesto za parkiranje bi našli hitreje in da je na sploh super ideja. Izpostavili so tudi nekaj negativnih lastnosti, brez kartice ne bi mogli dostopati do parkirišča in nekdo, ki ni GOO, bi lahko parkiral tam s sposojeno kartico.

Imate morda kakšen drug problem, ki bi se dal rešiti na podoben način?

Tudi to vprašanje je neobvezno, najina želja je bila, da mogoče dobiva kakšno idejo za rešitev problema s pomočjo OCR. Na to vprašanje sva prejela manj odgovorov, kot na prejšnje in večina ni bila veljavnih. Ena ideja je bila to, da nadziramo, kdo parkira na mestih za GOO direktno.

8 OVREDNOTENJE HIPOTEZ

H1: Izdelek bo v veliko pomoč GOO.

Predvidevala sva, da se GOO srečujejo s problemom pomanjkanja parkirnih mest, zato bi jim najin program omogočil boljše pogoje za parkiranje in bi otežil zlorabo parkirnih mest za GOO.

Iz rezultatov ankete je razvidno, da si okoli 46 % GOO želi več parkirnih mest, 46 % anketirancev je tudi izjavilo, da nikoli ne parkirajo na mestih za GOO, misliva, da bi najin program zmanjšal ta odstotek. Pri vprašanju »Ali se vam zdi najin program uporaben?« je 96 % GOO odgovorilo z da, saj se jim zdi to dobra rešitev problema.

To hipotezo lahko potrdiva.

H2: Izdelek bo uporabljen na več mestih.

Primarni namen najinega programa je omogočanje GOO nemoteno parkiranje z uporabo zapornice, ampak sva ugotovila, da se lahko ta program prilagodi za druge namene. Na primer omogočanje parkiranja samo zaposlenim nekega podjetja, tako da preverja njihove osebne izkaznice in registrsko tablico.

To hipotezo lahko potrdiva.

H3: Občina bo bolj prijazna do GOO.

Že od same ideje naprej je bil eden izmed glavnih namenov še izboljšati prijaznost občine do GOO. A iz same ideje še ne moreva vedeti, če bi se to res zgodilo v praksi, saj bi morali med GOO po uporabi teh parkirišč izvesti novo raziskavo o zadovoljstvu v svoji občini.

Te hipoteze zato ne moreva potrditi niti zavreči.

H4: Posledica je manjša možnost zlorabe.

Enostavna možnost zlorabe je ena izmed največjih slabosti parkirišč za GOO. Zato sva pričakovala, da bo prav ta zloraba zanimala anketirance, kar se tudi vidi iz neobveznega vprašanja o uporabnosti programa.

Če primerjamo možnosti zlorabe, imajo današnja parkirišča bistveno večjo možnost zlorabe, saj ne uporabljajo nobenega preverjanja razen kartice, pri nama gre za večjo zaščito pred zlorabo, ni pa nemogoča.

A vseeno lahko to hipotezo potrdiva.

H5: Težko bo anketirati reprezentativno število GOO.

Najina skrb je bila to, da bo težko anketirati veliko število GOO in s tem pridobiti veljavne podatke za najino raziskavo.

Anketo sva posredovala vsem društvom za GOO v Sloveniji in anketo je rešilo le 75 GOO.

To hipotezo lahko zato potrdiva.

H6: Anketo bodo v večini izpolnili starejše GOO

Ker sva anketo posredovala društvom GOO, v katerih prevladujejo starejše osebe in tudi pričakovala sva, da bodo anketiranci starejši od 18 let, saj morajo imeti voziško dovoljenje.

Iz 2. vprašanja v anketi je razvidno, da so res sodelovale starejše osebe, saj so vsi anketiranci starejši od 40 let.

To hipotezo lahko zato potrdiva.

9 ZAKLJUČEK

OCR in njemu podobne tehnologije imajo že v sedanjem svetu večji vpliv, kot si ga večina ljudi predstavlja. Uporabljene so že v skoraj vse namene, a velikokrat razvijalci pozabijo, kako bi lahko njihova tehnologija vplivala prav na tiste, ki bi s svojimi idejami in izdelki najbolj pomagali in jim olajšali nekatere dejavnosti, ki jih zaradi svojih zdravstvenih težav ne morejo opravljati. Večino problemov se danes rešuje tudi s povezavo umetne inteligence, ki že tako široko polje znanja razširi še na večje področje.

Prihodnost te tehnologije pa je povezana prav z uporabo umetne inteligence in področja robotike, ki iz dneva v dan napreduje, z njim pa tudi uporaba »globokega učenja«. Novi OCR algoritmi bodo hitrejši, zmogljivejši in še bolj enostavni za uporabo. Takšni programi bodo lahko na več načinov olajšali življenje vsem, ki to potrebujejo. Pomagali bodo podjetjem in drugim uradnim ustanovam, ki tehnologijo uporabljajo za digitaliziranje dokumentov in drugih listin.

Iz vsega tega lahko ugotovimo, da ta tehnologija res spada pod tehnologije prihodnosti, ki bodo močno vplivale na podobo sveta že v času naših življenj in upava, da bodo vse te izboljšave pomagale čim večjemu številu ljudi.

10 LITERATURA

Wikipedia: Optical character recognition. Pridobljeno 17. feb. 2021. Dostopno na: https://en.wikipedia.org/wiki/Optical_character_recognition.

Python: about Python. Pridobljeno 18. feb. 2021. Dostopno na : <https://www.python.org/about/>

Python: Notes for professionals. Pridobljeno 18. feb 2021. Dostopno na: <https://books.goalkicker.com/PythonBook/>

Heroku: Cloud platform. Pridobljeno 20. mar. 2021. Dostopno na: <https://dashboard.heroku.com/apps>

Devdocs: Javascript. Pridobljeno 20. mar. 2021. Dostopno na: <https://devdocs.io/javascript/>

NodeJS: documentation. Pridobljeno 20.mar. 2021. Dostopno na: <https://nodejs.org/api/>

Boostrap: documentation. Pridobljeno 21. mar. 2021. Dostopno na: <https://getbootstrap.com/docs/4.1/getting-started/introduction/>

Vlad-Stefan Harbuz: from-scratch-2-ocr. Pridobljeno 22. mar. 2021. Dostopno na: <https://github.com/vladh/clumsycomputer/tree/master/from-scratch-2-ocr>.

Yann LeCun, Corinna Cortes, Christopher J.C. Burges: MNIST. Pridobljeno 22. mar. 2021. Dostopno na: <http://yann.lecun.com/exdb/mnist/>.

Alenka Krapež: Tehnologije znanja pri predmetu informatika, Statične metode. Pridobljeno 28. mar. 2021. Dostopno na: <http://old.gimvic.org/predmeti/informatika/gradiva/html-ji/statisticne.html>.

Samuel Hoffstaetter: pytesseract 0.3.7. Pridobljeno 1. apr. 2021. Dostopno na: <https://pypi.org/project/pytesseract/>.

Aswirth Raj: License Plate Recognition using Raspberry Pi and OpenCV. Pridobljeno 3. apr. 2021. Dostopno na: <https://circuitdigest.com/microcontroller-projects/license-plate-recognition-using-raspberry-pi-and-opencv>.

Priloga 1: Vprašalnik o parkiranju na parkirnih mestih za gibalno ovirane osebe

Pozdravljeni,

sva dijaka 4. letnika, izdelujeva raziskovalno nalogo z naslovom »OCR algoritem za pomoč gibalno oviranim osebam«. Namen najine raziskovalne naloge je izdelati program, ki bi z umetno inteligenco prepoznal podatke iz invalidske izkaznice in s tem omogočil prednost osebam, ki jo potrebujejo, na primer dostop do parkirišča, kjer lahko parkirajo samo gibalno ovirane osebe.

Anketa je namenjena hendikepiranim osebam in traja približno 5 minut. Sodelovanje v anketi je anonimno. Vljudno vas prosiva za sodelovanje, saj nama bodo pridobljeni podatki v veliko pomoč pri najinem delu.

Za sodelovanje se vam že vnaprej zahvaljujema.

VPRAŠALNIK

1. Kako pogosto parkirate na parkirnih mestih za gibalno ovirane osebe?

- a) Vedno.
- b) Zelo pogosto.
- c) Včasih.
- d) Nikoli.
- e) Drugo.

2. Ali ste zadovoljni s številom parkirnih mest za gibalno ovirane osebe v svoji okolici?

- a) Da.
- b) Ne.
- c) Drugo.

3. Ali se vam zdi najin program uporaben?

- a) Da.
- b) Ne.

4. Pojasnite, zakaj ste izbrali da/ne.

5. Imate morda kakšen drug problem, ki bi se dal rešiti na podoben način?

Hvala za sodelovanje. Odgovori nama bodo v veliko pomoč.

IZJAVA*

Mentor/-ica TINEJ PIRŠ v skladu z 20. členom Pravilnika o organizaciji mladinske raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje, zagotavljam, da je v raziskovalni nalogi z naslovom OCR ALGORITEM ZA FOTOGRAFIALNO OVRANJAM OSEBAH, katere avtor/-ica je DOMEN HRIBELNIK, TILLEN KOČEN

- besedilo v tiskani in elektronski obliki istovetno,
- pri raziskovanju uporabljeno gradivo navedeno v seznamu uporabljene literature,
- da je za objavo fotografij v nalogi pridobljeno avtorjevo dovoljenje in je hranjeno v šolskem arhivu,
- da sme Osrednja knjižnica Celje objaviti raziskovalno nalogo v polnem besedilu na knjižničnih portalih z navedbo, da je raziskovalna naloga nastala v okviru projekta Mladi za Celje,
- da je raziskovalno nalogo dovoljeno uporabiti za izobraževalne in raziskovalne namene s povzemanjem misli, idej, konceptov oziroma besedil iz naloge ob upoštevanju avtorstva in korektnem citiranju,
- da smo seznanjeni z razpisni pogoji projekta Mladi za Celje.

Celje, 13.5.2021



Podpis mentorja

Tinej Pirš

Podpis odgovorne osebe

[Signature]

*

POJASNILO

V skladu z 20. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje je potrebno podpisano izjavo mentorja (-ice) in odgovorne osebe šole vključiti v izvod za knjižnico, dovoljenje za objavo avtorja (-ice) fotografskega gradiva, katerega ni avtor (-ica) raziskovalne naloge, pa hrani šola v svojem arhivu.