Mjerenje razine šećera pomoću digitalnog refraktometra Hanna Instruments 96801

Međimorec, Lukrecija

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

MJERENJE RAZINE ŠEĆERA POMOĆU DIGITALNOG REFRAKTOMETRA HANNA INSTRUMENTS 96801

Završni rad

Lukrecija Međimorec

Osijek, 2022.

Sadržaj

1. UVOD

Mjerenje razine šećera sastavni je dio proizvodnje u vinarstvu, pivarstvu, prehrambenoj industriji i različitim područjima poljoprivrede te ga je važno kontinuirano provoditi tijekom cijelog procesa proizvodnje. Određivanje razine šećera u početnim stadijima proizvodnje pomaže odrediti povoljan trenutak za početak berbe ili žetve, dok u kasnijim stadijima olakšava praćenje kakvoće proizvoda. U vinarstvu i pivarstvu ključan je parametar i tijekom fermentacijskog procesa te se na osnovu razine šećera određuje kada će se fermentacija zaustaviti.

Za određivanje razine šećera u moštu koriste se moštne vage (moštomjeri), najčešće Oechsleov moštomjer i Baboova moštna vaga, koji pokazuju specifičnu težinu mošta, odnosno koliko kilograma šećera ima u 100 kilograma mošta. Oechselov moštomjer izmjerenu vrijednost prikazuje u stupnjevima Oechsela, dok Baboova moštna vaga koristi mjernu jedinicu Baum. Za mjerenje se također može koristiti i refraktometar, koji daje vrijednost u Brixima, odnosno pokazuje koliko težinskih dijelova sladora ima u sto dijelova voćnog soka. Digitalni refraktometar je optički instrument koji mjerenjem indeksa loma određuje relevantne parametre u proizvodnji vina te se može opisati kao pouzdan instrument idealan za brza mjerenja. Njegovim se korištenjem u velikoj mjeri otklanjaju nesigurnosti povezane sa standardnim mehaničkim refraktometrima. Unatoč brojnim prednostima korištenja digitalnog refaktrometra, pojavljuje se problem kontinuiranog praćenja izmjerenih vrijednosti razine šećera koje bi zahtijevalo redovito očitavanje vrijednosti sa zaslona instrumenta. Kako bi se otklonila potreba za čestim provjeravanjem izmjerenih vrijednosti, proces se može automatizirati korištenjem fiksirane digitalne kamere koja kontinuirano snima zaslon refraktometra, a čija se očitanja obrađuju u stvarnom vremenu i vrijednosti mjerenja određuju metodama prepoznavanja znakova.

Rad je razrađen u tri poglavlja. U prvom poglavlju dana je kratka povijest korištene metode kao i pregled postojećih rješenja koja se trenutno koriste. Kroz drugo su poglavlje opisani korišteni alati, odnosno fizički uređaji i programi korišteni u svrhu izrade rada, dok je u trećem poglavlju detaljno opisan proces rješavanja problema i interpretirani su dobiveni međurezultati te je objašnjeno i prokomentirano konačno rješenje. Također su predložena moguća unaprjeđenja korištenog algoritma.

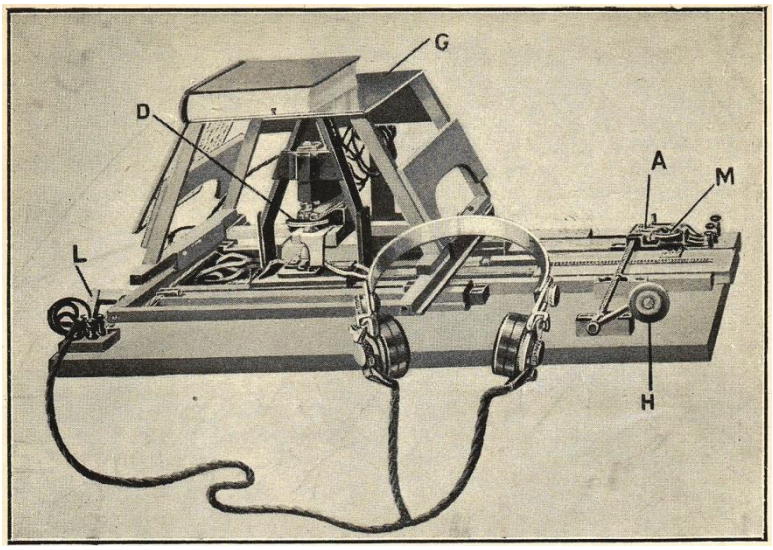
1.1. Zadatak završnog rada

U ovom radu je potrebno napraviti automatizirano mjerenje razine šećera s digitalnog refraktometra Hanna Instruments 96801. Uređaj prikazuje očitanje u obliku znakova na 7-segmentnom pokazniku. Iznad uređaja se nalazi kamera koja snima prikazane znamenke. Iz slike s kamere je potrebno prepoznati prikazane znakove primjenom metoda za prepoznavanje znakova (engl. Optical Character Recognition - OCR).

2. UPOTREBA ALGORITMA ZA PREPOZNAVANJE ZNAKOVA

Prepoznavanje znakova je proces koji obuhvaća dohvaćanje i obradu slike kako bi se ista pripremila za pretvorbu u tekstualni format, a sve kako bi se dalje mogle vršiti operacije nad informacijama sadržanim u dobivenom tekstu. Upotrebom OCR metoda uklanja se potreba za ručnim unosom teksta, odnosno prijepisom teksta sa slike, čime se štede brojni resursi, prije svega, ali ne isključivo, vrijeme.

Najranija pojava OCR-a seže čak u 1913. godinu kada je korišten u uređajima za asistenciju osobama slabog vida, na način da se uređajem , optofonom (slika 2.1.), prelazi preko pisanog teksta, tekst se prepoznaje i proizvode se zvukovi u skladu s prepoznatim slovom, kako bi slabovidna osoba znala što piše.[1] Znatan skok u popularnosti OCR algoritama započeo je devedesetih godina prošlog stoljeća, kada su se počeli koristiti za digitalizaciju povijesnih novina. Od tada su značajno modernizirani i poboljšani te primjenu pronalaze u raznim područjima, od digitalizacije tiskanih knjiga, preko prepoznavanja prometnih znakova u autonomnoj vožnji, do dohvaćanja bitnih informacija iz osobnih ili putnih dokumenata u zračnim lukama. Također se nastavljaju koristiti u asistenciji slabovidnim i slijepim osobama, posebice u obrazovanju i poslovnom svijetu, ali i za osobne potrebe.



Slika 2.1. Optofon, uređaj iz 1913. godine korišten za prepoznavanje napisanih znakova i pretvorbu prepoznatog teksta u niz zvučnih signala.

Dva su osnovna tipa algoritama za prepoznavanje znakova: algoritam zasnovan na prepoznavanju uzoraka i algoritam zasnovan na prepoznavanju značajki. Ukoliko algoritam radi pomoću prepoznavanja uzoraka, predaju mu se primjeri teksta u raznim oblicima kako bi ih algoritam usporedio i odredio sličnosti, odnosno razlike, između pojedinih znakova te na kraju odredio koji je koji znak na osnovu detektiranog uzorka. Ako s druge strane radi pomoću prepoznavanja uzoraka, algoritmu se prije predaje znakova definiraju značajke koje pojedini znak ima, primjerice broj i položaj vertikalnih, horizontalnih ili nakošenih linija, udaljenost i kut između njih te njihova duljina. Algoritam tada provjerava koje od definiranih značajki predani znak ima i na osnovu zadovoljenih značajki određuje o kojem se znaku radi.[1] U ovom radu je korišten algoritam zasnovan na prepoznavanju značajki.

2.1. Postojeća rješenja

S porastom digitalizacije u svim područjima poslovnog svijeta porastao je i broj dostupnih programa i alata koji implementiraju algoritme prepoznavanja znakova. Najpoznatiji i najčešće korišteni takav alat jest Adobe Acrobat Pro DC. Ovaj se alat koristi prvenstveno za uređivanje datoteka u PDF (Portable Document Format) formatu tako da se prvo prepozna tekst koji se u datoteci nalazi, zatim se taj tekst pohrani u datoteku koju je moguće uređivati. U svrhu ovog rada nije ga moguće koristiti.

Moguće je pronaći i niz manje poznatih alata koji rade samo sa slikama i nekolicina je u svrhu rada isprobana. Zbog prirode slika korištenih u ovom radu, dostupni alati nisu pouzdani i ne daju točno rješenje. Većina dostupnih alata kreirana je sa svrhom prepoznavanja blokova teksta sa slike, a kako se u ovom radu koriste slike zaslona sa prikazom znamenki na 7-segmentnom pokazniku, jasno je da će teško dati dobro rješenje.

3. OPIS ALATA KORIŠTENIH ZA DOHVAĆANJE I DALJNJU OBRADU SLIKE

Uređaj za mjerenje sastoji se od digitalnog refraktometra Hanna Instruments 96801 koji prikazuje očitanja u Brixima, digitalne kamere ESP32, LED osvjetljenja te 3D printanog kućišta u kojem su fiksirani digitalni refraktometar, osvjetljenje i kamera, što osigurava snimke kamere konstantnih dimenzija i konstantnog osvjetljenja. Konstantni uvjeti snimanja olakšavaju daljnju obradu slike.

3.1. Digitalni refraktometar Hanna Instruments 96801

Digitalni refraktometar je optički uređaj koji pretvara indeks loma voćnog soka u koncentraciju saharoze u jedinicama težinskih postotaka, postotak Brix (% Brix), a može se pronaći i izraz stupanj Brixa (°Brix). Kako je većina šećera u voćnom soku zapravo fruktoza i glukoza, očitavanje se ponekad naziva i prividni Brix.

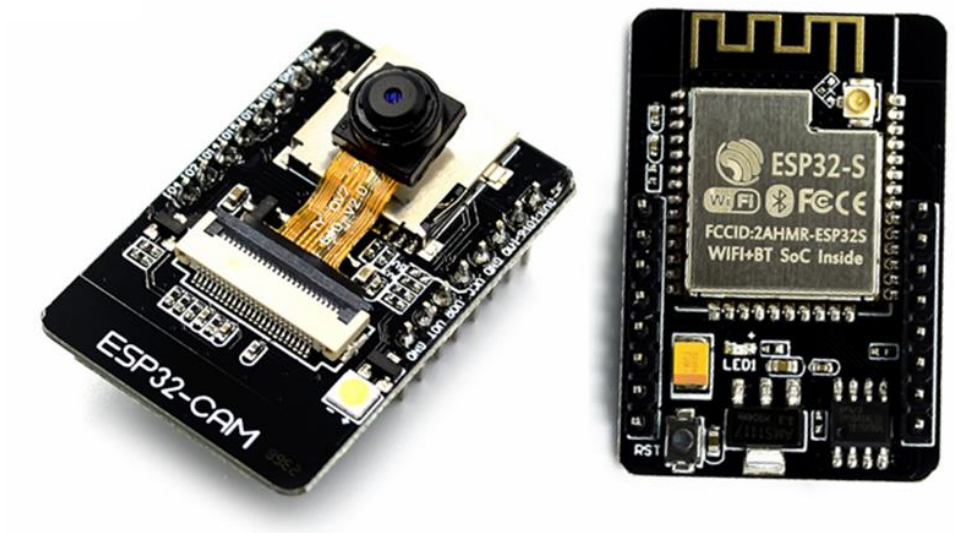
Slika 3.1. prikazuje refraktometar korišten u ovom radu koji prikazuje očitanja u rasponu 0 do 50% Brix, s točnošću ±2% Brix. Instrument ima automatsku korekciju temperature, tako da je vrijednost očitana na zaslonu u skladu sa stvarnim vrijednostima te ju nije potrebno korigirati s obzirom na razliku u temperaturi.[2]

Slika 3.2. Digitalni refraktometar Hanna Instruments 96801

3.2. ESP32-CAM

ESP32-S-CAM je uređaj s mikrokontrolerom, utorom za SD karticu i kamerom razlučivosti 800 x 600 px, koji pripada IoT (Internet of Things) uređajima, što znači da spajanjem preko Wi-Fi tehnologije može komunicirati s drugim uređajima i kontinuirano slati podatke. U ovom radu se koristi za snimanje zaslona refraktometra i za slanje snimki na daljnju obradu. Snimka zaslona se šalje na zahtjev korisnika.[3]

Slika 3.2. prikazuje ESP32-S-CAM modul sa svim navedenim komponentama.



Slika 3.3. ESP32-S-CAM modul

3.3. Python

Python je programski jezik visoke razine i opće namjene koji sve češće primjenu pronalazi u znanstvenim i tehničkim područjima. Koristi se u svrhu izrade aplikacija, softvera, analizu i vizualizaciju podataka, ali i u svrhu automatizacije te strojnog učenja.[4]

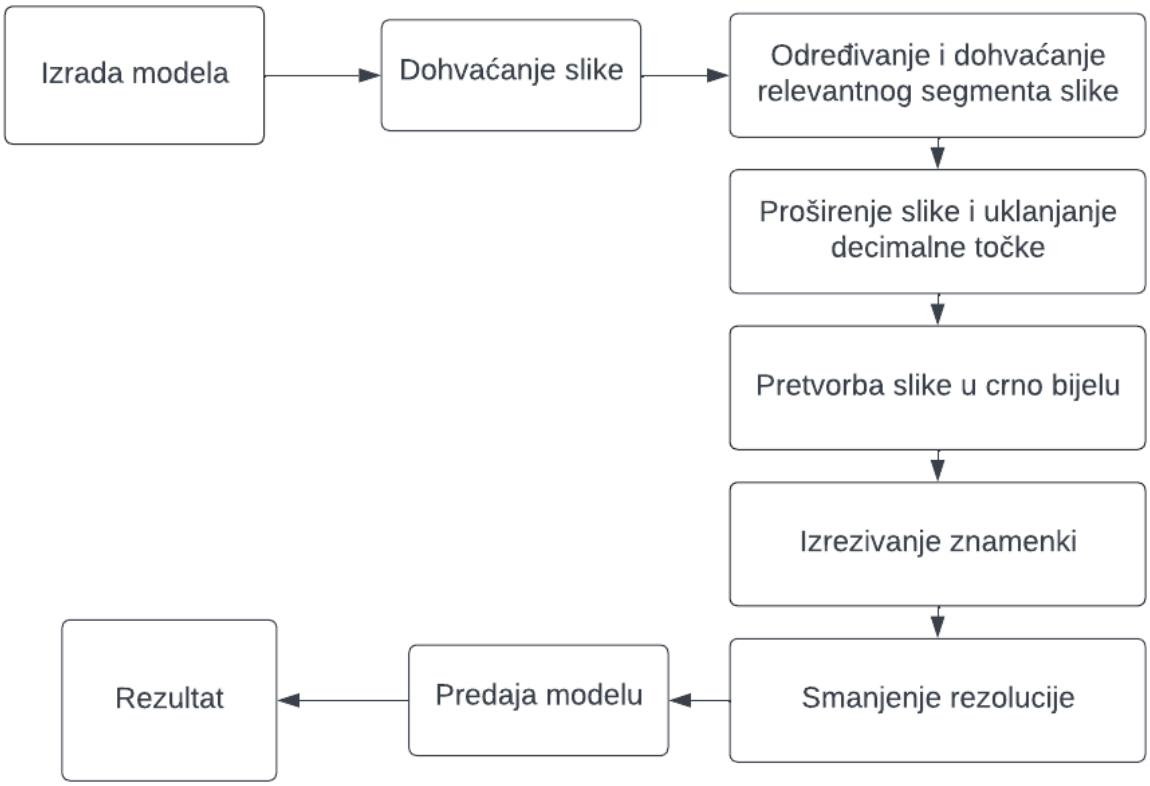
U ovom radu se koristi kao alternativa MATLAB-u za područje obrade slike i strojnog učenja. Prednost Pythona je u tome što je besplatan, lak za implementirati i sadrži mnoštvo biblioteka koje se koriste u strojnom učenju, a u području obrade slike približno je jednakih performansi kao MATLAB, iako se obradu oslanja na vanjske pakete.[5] Ukoliko se obrada slike koristi u kombinaciji sa strojnim učenjem, ili ako se obrada slike obavlja u stvarnom vremenu,

Python ipak pokazuje bolje performanse. Također ga je moguće pokrenuti u gotovo svim razvojnim okruženjima, što ga čini daleko pristupačnijim i fleksibilnijim.[6]

4. RAZVOJ I IMPLEMENTACIJA

U ovom poglavlju opisan je proces obrade primljene slike s kamere, na kojoj je prikazan zaslon digitalnog refraktometra s izmjerenom vrijednošću razine šećera u analiziranoj tekućini. Slika 4.1. prikazuje dijagram tijeka razvoja rješenja, odnosno grafički prikaz algoritma koji se koristi u svrhu analize aplikacija i procesa, a koristi se u projektiranju i izradi dokumentacije. Dijagram tijeka pomaže opisati događaje unutar razvoja i implementacije te olakšava razumijevanje cijelog procesa.

Kako je vidljivo u dijagramu tijeka, proces razvoja i implementacije može se okvirno podijeliti na 4 dijela: izrada modela, dohvaćanje slike, obrada slike i rezultat, a tim je redoslijedom obrazloženo rješenje zadatka završnog rada kroz 4. poglavlje.



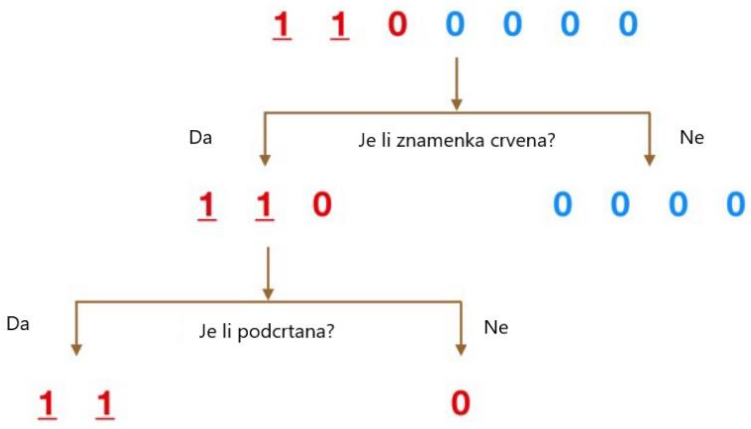
Slika 4.1. Dijagram tijeka

4.1. Izrada modela

Prije početka obrade slike potrebno je izraditi model koji će prepoznati koje su vrijednosti prikazane znamenkama i dati njihove vrijednosti u tekstualnom obliku. U tu se svrhu koristi skup podataka koji se sastoji od računalno generiranih slika, odnosno prikaza, 7-segmentnih znamenki. Skup podataka generiran je po uzoru na MNIST bazu podataka.

MNIST (engl. *Modified National Institute of Standards and Technology*) je baza podataka koja sadrži rukom napisane znakove dimenzija 28 x 28 px, koji se koriste u treniranju modela za prepoznavanje rukom napisanog teksta. Bazu je moguće implementirati direktno u Python kodu koristeći PyTorch radni okvir i ugrađene funkcije iz istog. Skupovi koji se dohvaćaju iz baze su u pravilu velikih dimenzija pa tako pojedini skup može sadržavati i do 60 tisuća znamenki. Budući da se u ovom radu ne koriste rukom napisani znakovi, ovaj se radni okvir ne koristi, kao niti baza koja se u njemu nalazi. Kako je korišten skup (u programskom kodu 4.1.1. 'sevensegdataset.npy') ipak generiran po uzoru na MNIST bazu, slike iz skupa su također dimenzija 28 x 28 piksela, što je važno za obradu slike, a koraci u obradi slike korišteni u ovom radu mogu se primijeniti i ako se koristi neki od MNIST skupova podataka.[7]

Nad korištenim se skupom koristi model nasumične šume (engl. *random forest*) koji se temelji na izgradnji stabla odluke (engl. *decision tree*). Pod izgradnjom stabla podrazumijeva se segmentiranje prediktorskog prostora u jednostavnije regije uz pomoć pravila dijeljenja, odnosno postavljanjem niza pitanja moguće je elemente dobivenog skupa podijeliti u kategorije ovisno o njihovim karakteristikama te doći do određenog zaključka. Da bi stablo odluke davalo pouzdane rezultate, regije se ne smiju preklapati. Slika 4.1.1. prikazuje primjer jednostavnog stabla odluke, a ista se logika primjenjuje i nad složenijim skupovima podataka.



Slika 4.1.1. Stablo odluke

Nasumična (slučajna) šuma sastoji se od velikog broja zasebnih stabala odluke. Svako stablo će nad određenim skupom dati svoju predikciju, a ona koja je najzastupljenija uzet će se kao konačna predikcija modela. Budući da su stabla individualna, odnosno ne koreliraju, konačna predikcija točnija je od predikcije svakog zasebnog stabla, čime na model slabo utječu greške pojedinih stabala, uz pretpostavku da neće sva stabla griješiti u istom trenutku i u istom smjeru.[8]

Da bi se trenirao model, prvo je potrebno učitati skup podataka i izvući značajke (eng. *features*). U ovom radu iz skupa se izvlače značajke i oznake (u kodu označene s X i Y), pri čemu su značajke opisni atributi, a oznake ono što model predviđa, konkretno znamenke koje je potrebno prepoznati sa zaslona (0-9).

Kreirane značajke i oznake ubacuju se u klasifikator nasumične šume, kako je prikazano u programskom kodu 4.1.1., pri čemu je parametar n_estmators broj stabala odluke koje klasifikator koristi.

```
features = np.load("sevensegdataset.npy")
X = features[:, :-1]
Y = features[:, -1]
clf = RandomForestClassifier(n_estimators = 10)
clf=clf.fit(X,Y)
```

Programski kod 4.1.1. Izrada modela nasumične šume

Model nasumične šume moguće je samostalno kreirati, ali se u ovom radu koristi gotov model. Budući da su modeli sadržani u radnim okvirima optimizirani, a radni opseg korištene opreme je ograničen, izvođenje programa koristeći gotov model znatno je brže.

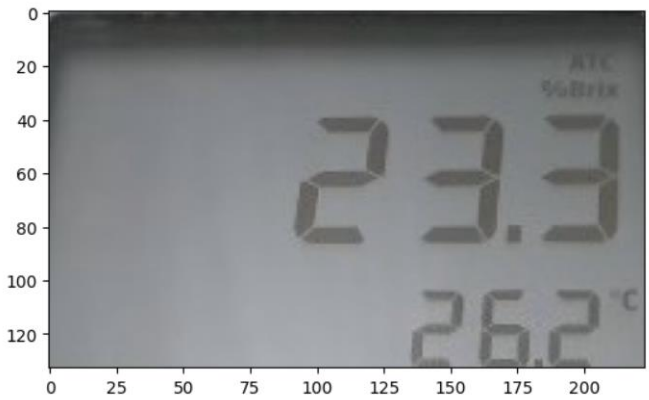
4.2. Dohvaćanje slike

Kako bi se dohvatila slika korisnik mora zatražiti njeno snimanje, nakon čega se ona šalje na obradu. Dobivenu sliku je potrebno učitati u Python kod, kako je prikazano programskim kodom 4.2.1., koristeći funkciju imread() iz OpenCV (cv2) radnog okvira. OpenCV radni okvir koristi se u svrhe računalnog vida, strojnog učenja i obrade slike kako bi se prepoznali i identificirali objekti, lica, rukom napisani znakovi i slično. Također se koristi u svrhu prepoznavanja uzoraka i detekciju anomalija, ukoliko se iste pojave, te prebrojavanje prepoznatih objekata. Moguće ga je kombinirati s ostalim radnim okvirima i na taj način dodatno proširiti područje u kojem se primjenjuje.[9]

```
starting_image = cv2.imread("6.jpg")
plt.imshow(starting_image)
```

Programski kod 4.2.1. Učitavanje primljene slike

Učitana slika prikazana je slikom 4.2.1., a ista je dobivena korištenjem funkcije imshow(), kako je prikazano u programskom kodu 4.2.1.



Slika 5. Rezultat pozivanja funkcije imshow(), snimka zaslona refraktometra

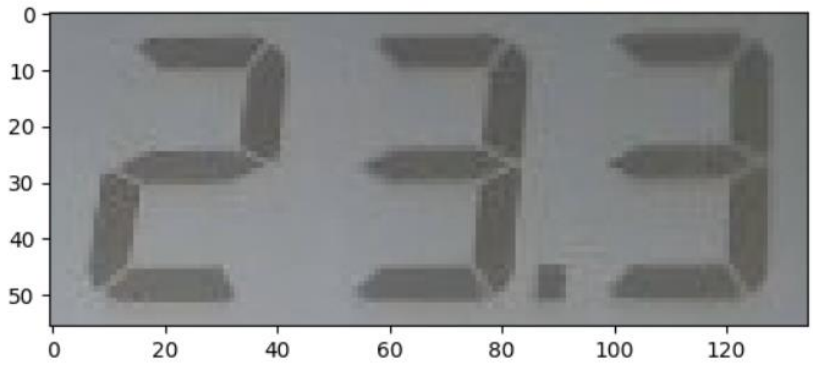
4.3. Obrada slike

Učitanu sliku potrebno je analizirati i pripremiti za daljnju obradu. Pod digitalnom obradom slike podrazumijeva se analiza digitalne slike i manipulacije nad istom koristeći računalo.[10] Obrada slike smatra se završenom kada je krajnja dobivena slika takva da ju je moguće dalje koristiti bez dodatnih izmjena i manipulacija. Analiza se provodi kroz više koraka. Prvo je potrebno odrediti dimenzije slike i je li ona u boji ili crno bijela, zatim odrediti koje je područje slike relevantno za obradu i odrezati dio slike koji nije relevantan, na način prikazan programskim kodom 4.3.1. Budući da zaslon prikazuje izmjerenu razinu šećera i izmjerenu temperaturu, potrebno je iz slike izdvojiti samo područje koje prikazuje izmjerenu razinu šećera. Kako je kamera fiksirana i slike su uvijek istih dimenzija, relevantno područje određuje se jednom i isto je za svaku učitanu sliku, dakle svaka se učitana slika izrezuje na isti način, jednostavnim pokretanjem prikazanog programskog koda.

```
print(starting_image.shape)
#dimenzije su 133 x 223 px
x=35
w=91
y=85
h=220
cropped_image = starting_image[x:w, y:h]
plt.imshow(cropped_image)
```

Programski kod 4.3.1. Određivanje i dohvaćanje relevantnog područja

Slika 4.3.1. prikazuje rezultat pozivanja funkcije imshow() nakon dohvaćanja relevantnog područja. Dimenzije nakon izrezivanja su 56 x 135 px, pri čemu je visina slike ciljano 56 piksela, kako kasnije ne bi bilo potrebe za mijenjanjem te dimenzije.

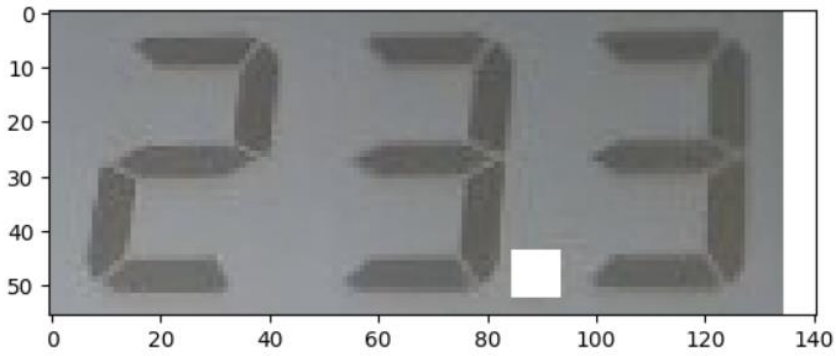


Slika 6.3.1. Rezultat pozivanja funkcije imshow() nakon izrezivanja

Sliku je kasnije u procesu potrebno izrezati tako da se dobiju tri slike takve da svaka sadrži jednu znamenku, a svaku je sliku zatim potrebno smanjiti na dimenzije 28 x 28 px kako bi se mogle ubaciti u model. Kako je slika širine 135 px, potrebno ju je proširiti dodatnim pikselima kako bi se nakon izrezivanja krajnja desna znamenka nalazila u sredini slike. Također, potrebno je ukloniti decimalnu točku, kako ista ne bi nakon izrezivanja bunila model. Slika se proširuje do 141 px, a decimalna točka prekriva kvadratom sačinjenog od bijelih piksela. Navedeno je napravljeno kako je prikazano programskim kodom 4.3.2., a slika 4.3.2. prikazuje rezultat pozivanja funkcije imshow() nakon napravljenih promjena.

```
cropped_image[44:53, 85:94]=[255,255,255]
W = [255,255,255]
Image_wide = cv2.copyMakeBorder(cropped_image, 0,0,0,6, cv2.BORDER_CONSTANT,
value=W)
plt.imshow(image_wide)
```

Programski kod 4.3.2. Proširenje slike i uklanjanje decimalne točke



Slika 4.3.2. Rezultat pozivanja funkcije imshow() nakon izmjena

Kako je vidljivo sa slike 4.3.2. u gornjem desnom kutu je sjena koju je potrebno smanjiti kako ne bi unosila grešku u rezultat. Sjena se smanjuje snižavanjem svjetline cijele slike, kako je prikazano programskim kodom 4.3.3.

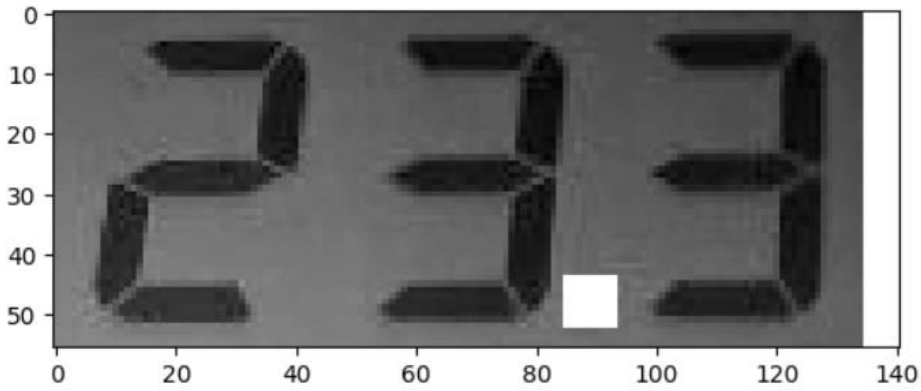
```
def increase_brightness(img, value=30):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)

    lim = 255 - value
    v[v > lim] = 255
    v[v <= lim] += value

    final_hsv = cv2.merge((h, s, v))
    img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    return img


image_light=increase_brightness(image_wide)
plt.imshow(image_light)
```

Programski kod 4.3.2. Smanjenje svjetline slike

Nakon smanjenja svjetline sliku je potrebno pretvoriti u crno bijelu. Prvi dio programskog koda 4.3.3. prikazuje proces pretvorbe slike u crno bijelu koristeći funkciju cvtColor() s COLOR_RGB2GRAY argumentom iz OpenCV radnog okvira. Sliku je potrebno prvo pretvoriti u nijanse sive jer se time slika kasnije jednostavnije binarizira, odnosno brže se i efikasnije uspoređuju pikseli s definiranom granicom. Navedena funkcija kao rezultat vraća željenu modifikaciju prikazanu slikom 4.3.3.

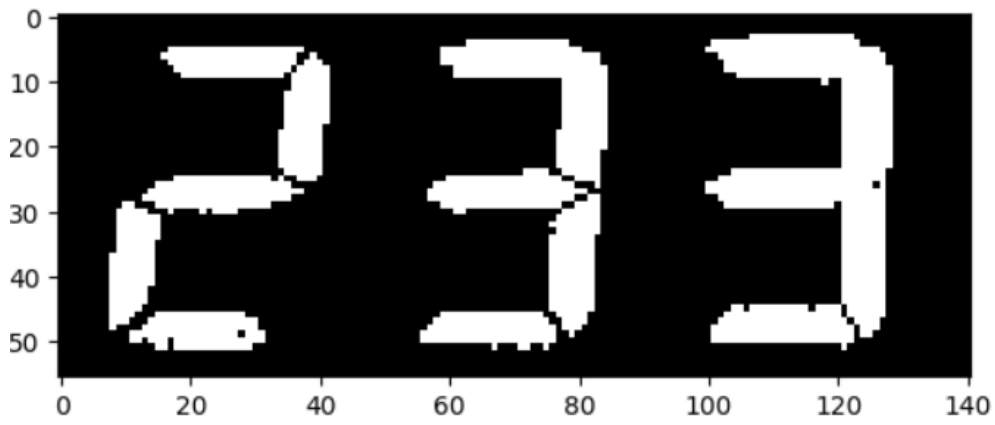
```
image_gray = cv2.cvtColor(image_light, cv2.COLOR_RGB2GRAY)
plt.imshow(image_gray, cmap='gray')
split = filters.threshold_otsu(image_gray)
print(split)
split = 143
image_numbers = np.array((image_gray < split)*255, dtype = np.uint8)
plt.imshow(image_numbers, cmap='gray')
```

Programski kod 4.3.3. Proces pretvorbe slike u crno bijelu



Slika 4.3.3. Izlaz nakon pretvorbe slike u crno bijelu

Nakon pretvorbe slike u crno bijelu potrebno je sliku pretvoriti u binarnu, a za to je potrebno odrediti granicu koja se uzima u obzir kod određivanja treba li se piksel pretvoriti u crni ili bijeli. U tu se svrhu koristi funkcija threshold_otsu() koja daje okviran iznos granice, koju je moguće modificirati ako nije u potpunosti točna, što u ovom slučaju nije.[11] Granica dobivena funkcijom je 193, što je ipak previsoka granica te ju je u kodu potrebno promijeniti. Prolaskom kroz sliku pikseli se uspoređuju s definiranom granicom i ako je vrijednost piksela manja od granice, on se postavlja u 0, odnosno u crni, a ako je veća onda se piksel postavlja u bijeli, odnosno u 1. Binarizirana slika prikazana je slikom 4.3.4.



Slika 4.3.4. Binarizirana slika

Da bi se dobile zasebne znamenke sliku je potrebno izrezati. Pokretanjem programskog koda 4.3.4. slika se izrezuje na tri dijela dimenzija 56 x 56 px.

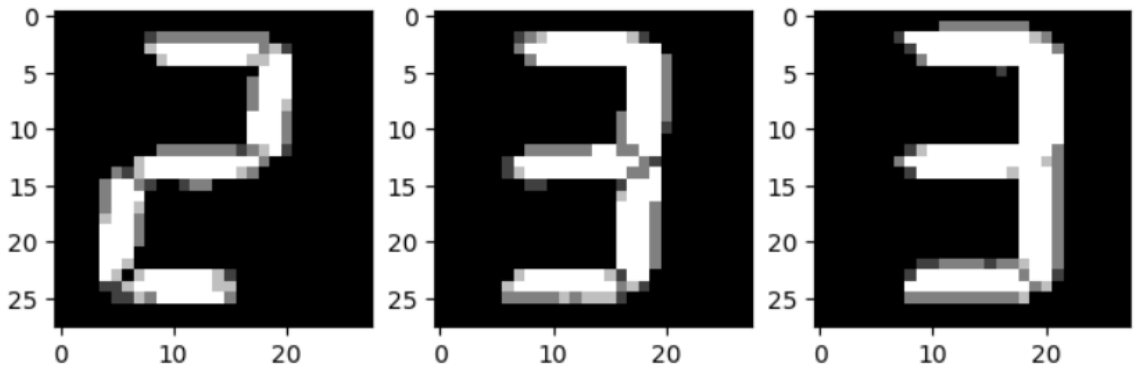
```
d1 = image_numbers[:, :56]
d2 = image_numbers[:, 44:100]
d3 = image_numbers[:, 85:]
```

Programski kod 4.3.4. Izrezivanje znamenki

Nakon izrezivanja, a prije predaje slika modelu, slike je potrebno smanjiti na dimenzije 28 x 28 px pokretanjem programskog koda 4.3.5., što za rezultat daje izlaz prikazan slikom 4.3.5. Smanjenje dimenzija slike provodi se korištenjem funkcije resize() koja kao parametre prima sliku koja se želi smanjiti te faktore skaliranja po vertikalnoj i horizontalnoj osi, koji su u ovom slučaju jednaki i iznose 0.5 jer se slika svodi na polovicu početne rezolucije.

```
d1=cv2.resize(d1, (0,0), fx=0.5, fy=0.5)
d2=cv2.resize(d2, (0,0), fx=0.5, fy=0.5)
d3=cv2.resize(d3, (0,0), fx=0.5, fy=0.5)

digits=[d1,d2,d3]
fig=plt.figure(figsize=(8,8))
for i, d in enumerate(digits):
    fig.add_subplot(1,3, i+1)
    plt.imshow(d, cmap='gray')
plt.show()
```

Programski kod 4.3.5. Smanjenje dimenzija slika



Slika 4.3.5. Slike znamenki smanjene rezolucije

Nakon što su slike znamenki svedene na potrebnu rezoluciju, iste se mogu predati modelu na predviđanje, odnosno prepoznavanje. Slike se spremljene u polju predaju modelu kako je prikazano programskim kodom 4.3.6.

```
X=np.array(digits).reshape((3,-1))
array=clf.predict(X)
```

Programski kod 4.3.6. Predaja polja slika modelu

Predanom polju je potrebno promijeniti oblik koristeći funkciju reshape() koja prima proizvoljan broj parametara po kojima preoblikuje polje. U ovom slučaju polju se predaju vrijednosti 3 i -1. Vrijednost 3 govori funkciji da će polje sadržavati 3 reda, budući da imamo tri znamenke, a vrijednost -1 da pri predaji polja funkciji nije poznato koliko elemenata svaki red treba sadržavati te je na taj način naznačeno da funkcija sama treba odrediti koliko će elemenata smjestiti u svaki red.

4.4. Konačni rezultat

Konačni rezultat ispisan je u obliku polja koje sadrži tri elementa koje je potrebno pri ispisu povezati u pregledniji format. Izabran je format niza znakova. Budući da je vrijednost prikazana na zaslonu refraktometra uvijek takva da ima tri znamenke i da je uvijek jedna od tri znamenke iza decimalne točke, u ovom koraku se također dodaje decimalna točka. Kroz rad je korišten primjer u kojem je cjelobrojni dio vrijednosti dvoznamenkast i prisutne su tri planirane znamenke, ali kako to ne mora biti tako potrebno je definirati slučaj kada je cjelobrojni dio izmjerene vrijednosti jednoznamenkast. U tom se slučaju provjerava slika dimenzije 28 x 28 piksela koja predstavlja prvu znamenku na način da se prolazi kroz sliku i provjerava postoji li neki piksel koji nije crn. Pikseli koji nisu crni se broje i ukoliko nema izbrojenog piksela koji nije crn korištena funkcija CountNonZero vraća 0, a ukoliko takvi pikseli postoje funkcija vraća njihov broj. [12] Ukoliko funkcija vrati broj veći od nule, smatra se da postoji znamenka i ispisuju se sva tri elementa polja, a ako funkcija vrati nulu smatra se da je izmjerena vrijednost jednoznamenkasta i prvi element polja se ne ispisuje. Navedeno je prikazano programskim kodom 4.4.1.

```
if cv2.countNonZero(d1)==0:
    print(str(array[1]) + '.' + str(array[2]))
else:
    print(str(array[0]) + str(array[1]) + '.' + str(array[2]))
```

Programski kod 4.4.1. Provjera prisutnosti prve znamenke i ispis rezultata

Konačno rješenje je ispisano kao 23.3, što je točna izmjerena vrijednost razine šećera.

Brzina izvođenja programa je zadovoljavajuća, što je postignuto maksimalnim korištenjem predefiniranih funkcija budući da su iste optimizirane i izvršavaju se brže od korisnički definiranih. Izvođenje je moguće dodatno poboljšati otklanjanjem smetnji kao što je sjena na način da se položaj osvjetljenja izmjeni, čime bi se otklonila potreba za promjenom svjetline slike i samim time bi izvođenje programa bilo znatno brže.

5. ZAKLJUČAK

Porastom obujma potrebnih mjerenja u industriji raste i potreba za automatizacijom u svim područjima u kojima ju je moguće implementirati. Kako je sadržaj šećera u vinarstvu jedan od parametara koji u svim koracima proizvodnje diktira koje je mjere potrebno dalje poduzeti, mjerenje istog neophodno je vršiti kontinuirano i uz minimalne pogreške. Da bi se postiglo navedeno, odnosno točna i pravovremena mjerenja, proces je moguće automatizirati. Uz automatizaciju, osim navedenog, smanjuju se i troškovi do kojih dolazi u slučaju ljudske pogreške.

Obrada slike svoju primjenu pronalazi u sve više područja tehnologije te je sastavni dio algoritama strojnog učenja u medicini, autonomnoj vožnji i ostalim područjima robotike. Također predstavlja podlogu za algoritme prepoznavanja znakova, od kojih je jedan korišten ovom radu. Algoritmi prepoznavanja znakova mogu se koristiti u procesima digitalizacije, čime se smanjuje potreba za ručnom pretvorbom datoteka u digitalni oblik, a od velike je pomoći i ljudima smanjenih sposobnosti vida, budući da se prepoznati tekst može pretvoriti u zvučne znakove.

U ovom radu je cilj bio objediniti gore navedeno. Bilo je potrebno automatizirati proces mjerenja razine šećera korištenjem algoritma za prepoznavanje znakova, tako da se slika obradi brzo i efikasno i da konačan rezultat bude ispravan. Napisani algoritam korišten je na znamenkama koje su oblika 7-segmentnog pokaznika, ali budući da su slike korištene u kreiranju modela takve da odgovaraju zahtjevima MNIST baze podataka, algoritam je moguće primijeniti i na skupove podataka iz navedene baze, čime je primjena algoritma proširena i na rukom napisane znakove.

6. LITERATURA

[1] „What Is Optical Character Recognition (OCR)?“, IBM Cloud https://www.ibm.com/cloud/blog/optical-character-recognition

[2] Online katalog proizvoda, Hanna Instruments https://www.hannainst.hr/digital-refractometer-for-measurement-of-sugar-in-wine-hi96814-product

[3] „ESP32-CAM: Machine Vision Tips, Camera Guides and Projects“, ArduCam https://www.arducam.com/esp32-machine-vision-learning-guide/

[4] Python dokumentacija https://www.python.org/about/apps/

[5] „Understanding The Key Differences Between Python and MATLAB“, Analytics Insight https://www.analyticsinsight.net/understanding-key-differences-python-matlab/#:~:text=In%20engineering%2C%20Python%20also%20helps,processing%20relies%20on%20external%20packages.

[6] „MATLAB vs. Python“, Javatpoint https://www.javatpoint.com/matlab-vs-python

[7] „MNIST“, PapersWithCode https://paperswithcode.com/dataset/mnist

[8] „Understanding Random Forest“, Towards Dana Science https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[9] „OpenCV Overview“, GeeksForGeeks https://www.geeksforgeeks.org/opencv-overview/

[10] „Image Processing“, ScienceDirect https://www.sciencedirect.com/topics/engineering/image-processing

[11] „Otsu's Thresholding with OpenCV“, LearnOpenCV https://learnopencv.com/otsu-thresholding-with-opencv/

[12] NumPy dokumentacija https://numpy.org/doc/stable/reference/generated/numpy.count_nonzero.html

SAŽETAK

Cilj rada je razviti algoritam za prepoznavanje znakova sedam segmentnog pokaznika sa zaslona digitalnog refraktometra. Slika se dohvaća pomoću kamere koja ju šalje putem WiFi veze. Obradom slike dolazi se do parametara koji se predaju modelu koji pomoću modela nasumične šume prepoznaje o kojim se znakovima radi. Rješenje je napravljeno u programskom jeziku Python, čime je osigurano da se program može pokrenuti u više razvojnih okruženja te je time program i dostupniji.

Ključne riječi: automatizacija mjerenja, obrada slike, prepoznavanje znakova, Python, stablo odluke, nasumična šuma

ABSTRACT

MEASURING SUGAR LEVELS USING DIGITAL REFRACTOMETER HANNA INSTRUMENTS 96801

The goal of this work is to develop an algorithm for character recognition to be used with a seven-segment display of a digital refractometer. An image is acquired using a digital camera that sends it using a wireless connection. Using digital image processing the algorithm obtains parameters which are then passed to a random forest machine learning model which recognizes the displayed characters. The solution was written using Python programming language, which ensures the program can be run in multiple IDEs, making the program widely available.

Keywords: measurement automatization, image processing, Optical Character Recognition (OCR), Python, decision tree, random forest model