Mobilna aplikacija za određivanje fonta teksta sa slike

Silađi, Emil

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

MOBILNA APLIKACIJA ZA ODREĐIVANJE FONTA TEKSTA SA SLIKE

Diplomski rad

Emil Silađi

Osijek, 2016.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 08.12.2016.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Emil Silađi
Studij, smjer:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D 754 R, 13.10.2014.
OIB studenta:	48868521154
Mentor:	Doc.dr.sc. Emmanuel Karlo Nyarko
Sumentor:	
Predsjednik Povjerenstva:	Doc.dr.sc. Damir Filko
Član Povjerenstva:	Doc.dr.sc. Ratko Grbić
Naslov diplomskog rada:	Mobilna aplikacija za određivanje fonta teksta sa slike
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Izraditi mobilnu aplikaciju koja će odrediti font teksta sa slike snimljenom mobilnim uređajem. (Sumentor: Darjan Bogdan, Mono d.o.o. Bihaćka 1d, 31000 Osijek)
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 Postignuti rezultati u odnosu na složenost zadatka: 3 Jasnoća pismenog izražavanja: 3 Razina samostalnosti: 3
Datum prijedloga ocjene mentora:	08.12.2016.
Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis: Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 19.12.2016.

Ime i prezime studenta:	Emil Silađi
Studij:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D 754 R, 13.10.2014.
Ephorus podudaranje [%]:	1%

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna aplikacija za određivanje fonta teksta sa slike**

izrađen pod vodstvom mentora Doc.dr.sc. Emmanuel Karlo Nyarko

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.

Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Različiti fontovi se nalaze svuda oko nas. Oni su jedno od glavnih dizajnerskih obilježja bilo kakvih tekstualnih poruka, logotipa i izražaja. Odabir pravog fonta ima veliki utjecaj na krajnji utjecaj poruke, bila ona logo kompanije ili internet stranica. Među tisućama različitih fontova, izazov je odabrati onaj pravi ili saznati ime fonta kojeg smo vidjeli. Za tu svrhu postoje internet stranice koje nude prepoznavanje fonta sa slike ili odabira fonta sličnih karakteristika. Cilj diplomskog rada je prebaciti sličnu funkcionalnost na mobilni uređaj. Mobilni uređaji su danas široko rasprostranjeni i omogućavaju brzo korištenje čime bi se korisniku omogućilo brzo raspoznavanje fonta bez prebacivanja slike na računalo i slanja na različite servise.

Diplomski se rad bavi prepoznavanjem fonta teksta sa slike preko mobilne aplikacije. Korisnik aplikacije ima mogućnost uslikati ili odabrati već postojeću sliku neke scene. Koristeći program otvorenog koda za proces optičkog prepoznavanja znakova (engl. *Optical Character Recognition – OCR*) pronašli bi se znakovi na toj sceni u vidu slova i brojki. Nakon toga bi se poznati znakovi sa slike usporedili sa znakovima različitih fontova i odredio najsličniji. Algoritam za klasifikaciju se ostvaruje pomoću *k*NN (engl. *k-nearest neighbor)* algoritma za klasifikaciju. Algoritam bi kao izlaz davao sortiranu listu fontova najsličnijih tekstu sa slike i prikazali korisniku unutar mobilne aplikacije.

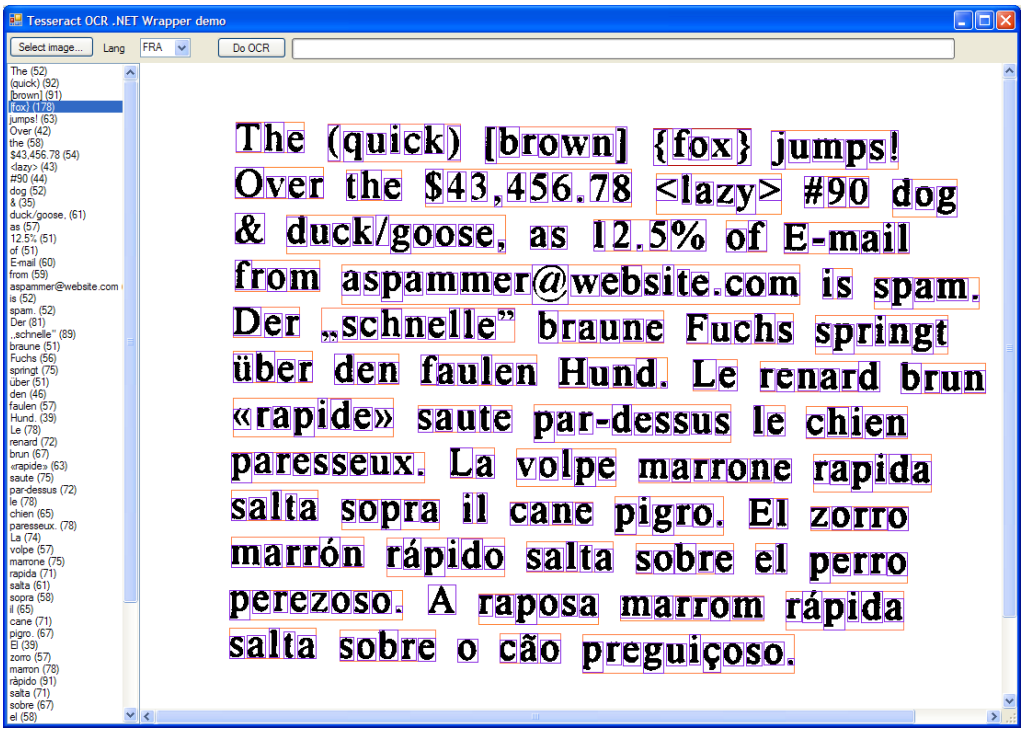
2. OPTIČKI ČITAČ ZNAKOVA – PREPOZNAVANJE TEKSTA SA SLIKE

Optički čitač znakova ili proces optičkog prepoznavanja znakova uključuje određeni računalni program kojemu je cilj prebaciti sliku teksta koji se nalazi na papiru u tekst prepoznatljiv računalu kojeg je zatim moguće prikazati ili urediti.

2.1. Alati za prepoznavanje teksta slike

Alat korišten za prepoznavanje teksta sa slike je Tessnet2 [1]. Tessnet2 je program otvorenog koda napravljen u programskom jeziku C# koji koristi Tesseract za optičko prepoznavanje znakova. Tesseract je program za optičko prepoznavanje znakova (engl. *Optical Character Recognition – OCR*) razvijen za razne operacijske sustave originalno od strane Hewlett Packard-a te sponzoriran i razvijan od strane Google-a od 2006. godine. To je besplatni program otvorenog izvornog koda (engl. *Open source*) izdan pod Apache licencom. Smatran je jednim od najtočnijih alata otvorenog koda za prepoznavanje teksta sa slike [2].

Originalno razvijen u programskom jeziku C i kasnije prilagođen za C++ Tesseract ne posjeduje grafičko korisničko sučelje i pogodan je za korištenje kao *backend* program, dok je Tessnet razvijen s grafičkim sučeljem. Što se tiče same slike kao ulaza u program, postoje neka ograničenja poput osjetljivosti na rezoluciju: visina teksta bi trebala biti najmanje 20 *pixela*, rotaciju ili nagib slike je potrebno ispraviti i spore promjene u osvjetljenju moraju biti ispravljene prije korištenja *softver-a* kako bi se dobili najbolji rezultati [3]. Tessnet2 sučelje je prikazano na slici 2.1.



Slika 2.1 Tessnet2 sučelje

S obzirom da u radu nije potrebno grafičko sučelje za optičko prepoznavanje znakova a Tessnet2 je alat otvorenog koda, iskorišteni su samo neki dijelovi programa koji pozivaju funkcionalnost samog alata Tesseract.

```
tessnet2.Tesseract ocr = new tessnet2.Tesseract();

ocr.SetVariable("tessedit_char_whitelist","ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop
      qrstuvwxyz0123456789");
```

Programski kod iznad prikazuje pokretanje Tesseract instance preko tessnet2 biblioteke i postavljanje dopuštenih znakova. Dopušteni znakovi uključuju mala i velika pisana slova te brojke te označavaju znakove koje korisnik želi prepoznati.

2.2. Rezultati prepoznavanja teksta

Nakon što se tekst prepozna koristeći Tesseract, na izlazu se dobiju sva otkrivena slova, odnosno njihove koordinate, veličina kvadrata koji opisuje svako slovo i samo slovo koje je prepoznato. Primjer izlaza prikazan je u programskom kodu ispod:

```
The (42)
T : {X=16,Y=47,Width=41,Height=48}
h : {X=57,Y=43,Width=37,Height=52}
e : {X=95,Y=57,Width=33,Height=38}
spectacle (66)
s : {X=148,Y=60,Width=27,Height=38}
p : {X=176,Y=60,Width=37,Height=53}
e : {X=215,Y=59,Width=32,Height=38}
c : {X=249,Y=60,Width=31,Height=38}
t : {X=282,Y=52,Width=24,Height=46}
a : {X=306,Y=60,Width=32,Height=38}
c : {X=339,Y=60,Width=32,Height=38}
l : {X=373,Y=45,Width=16,Height=52}
e : {X=390,Y=61,Width=32,Height=37}
…
```

Ti podaci se koriste u daljnjem procesu prepoznavanja fonta. Prikaz dobivenih podataka je predstavljen na slici 2.2. Sama slika se ne generira u finalnoj fazi i služi samo kao primjer procesa prepoznavanja znakova.

Slika 2.2. Primjer detekcije riječi i slova koristeći Tesseract

2.3. Metode prepoznavanja teksta sa slike

Što se tiče algoritma kojeg Tesseract koristi za prepoznavanje teksta sa slike, on se odvija u više koraka. Pretpostavlja se da je ulaz za samu obradu slike binarna slika s opcionalnim definiranim područjima u kojima se nalazi tekst. Sam proces je podijeljen u više faza [2].

Prvi korak je analiza spojenih komponenti na slici u kojoj se spremaju konture oblika. To je relativno zahtijevan računski postupak ali ima prednost što programu daje mogućnost prepoznavanja bijelog teksta na crnoj podlozi. U ovoj fazi te konture se grupiraju u skupine. Te skupine se organiziraju u linije slova. Te se linije i regije analiziraju kako bi se odredio stupanj nagiba teksta ili proporcionalni tekst. Linije slova se rastavljaju na riječi s obzirom na razmak između slova. Tekst s fiksnim nagibom se odmah rastavlja na slova koristeći poznati nagib. Proporcionalni tekst se rastavlja na riječi koristeći poznate razmake i nejasne razmake.

Druga faza započinje proces s dva koraka. U prvom prijelazu preko slike, pokušava se prepoznati svaku riječ, jednu za drugom. Svaka riječ koja zadovoljava se šalje adaptivnom klasifikatoru kao podatak za učenje. Sa svakom riječi koju primi kao riječ za učenje, adaptivni klasifikator ima sve veće mogućnosti prepoznati riječi kako se prolazi kroz tekst. Kako je moguće da je adaptivni klasifikator naučio nešto korisno prekasno, drugi korak je ponovni prelazak preko slike te se riječi koje nisu dovoljno dobro prepoznate, ponovno pokušaju prepoznati.

Završna faza rješava nejasne razmake koji su nađeni tijekom rastavljanja teksta na slova i provjerava alternativne hipoteze za x-visinu kako bi se pronašao tekst male veličine [2].

3. ODREĐIVANJE FONTA TEKSTA SA SLIKE

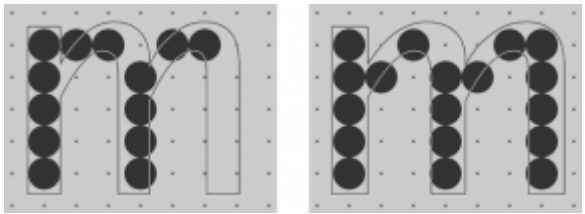
U ovom poglavlju bit će opisani sami fontovi i njihov način pohrane. Također, opisane će biti metode i algoritam za određivanje fonta koji se primjenjuju u mobilnoj aplikaciji, izrada podataka za učenje te sam proces određivanja fonta. Poglavlje će biti zaključeno s rezultatima testiranja algoritma na primjerima slika dobivenih preko mobilnog uređaja.

3.1. Fontovi i njihov format pohrane

Danas najrasprostranjeniji format za pohranu fontova je TrueType specifikacija koja se koristi na Microsoft Windows-ima i Apple Mac OS-u. TrueType fontovi su skalabilni, što znači da se znakovi poznati i kao glifovi mogu prikazati na ekranu pri bilo kojoj rezoluciji i veličini iako kvaliteta prikaza opada u ekstremnim slučajevima. TrueType font je binarna datoteka koja sadrži više tablica s direktorijem tablica na početku datoteke.

Svi fontovi sadrže glifove. TrueType fontovi opisuju svaki glif kao skup putanja. Putanja je zatvorena krivulja koja se specificira koristeći točke i matematičke formule. Na primjer, malo slovo „ i “ sadrži dvije putanje, jednu za točku i drugu za tijelo. Pri prikazu znakova, putanje se ispunjaju *pixelima* kako bi se kreirao oblik znaka. Skupina putanja se naziva obris. Jedan od načina na koji se glif može specificirati je referencirajući ostale glifove koji se kombiniraju kako bi se napravio novi složeni glif.

Osim matematičkih funkcija, u podacima fonta se mogu spremiti i dodatne instrukcije ili natuknice (engl. *Hints*) koje se izvode pri prikazu glifova na ekranu. Primjer natuknica se može vidjeti na slici 3.1.



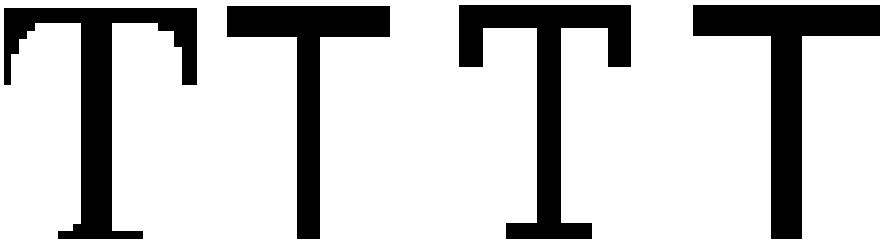
Slika 3.1. (lijevo) izobličen prikaz slova „m“; (desno) Popravljanje prikaza slova „m“ koristeći natuknice

Natuknice pomiču točke koje definiraju glif kako bi ih se bolje pozicioniralo u odnosu na mrežu na koju se crtaju glifovi. Razlika u kvaliteti prikaza između fontova sa i bez natuknica je velika, no kreiranje tih natuknica je složen postupak. Moguće je automatsko stvaranje natuknica, no rezultati u kvaliteti variraju ovisno o složenosti fonta.

Dodatne tablice se nalaze u binarnoj datoteci fonta. Neke od najznačajnijih tablica su: Tablica „cmap“ – pretvara znakove vanjskog kodiranja tipa Unicode u interne jedinstvene oznake glifova (engl. *ID*). Tablica se koristi kako bi se Unicode znakovi u rečenici preveli u jedinstvene oznake glifova i točno prikazali pripadajuće glifove na ekranu ili printeru. Tablice „hmtx“, „hdmx“, „OS/2“ i ostale – predstavljaju metrike potrebne za točno međusobno postavljanje glifova poput razmaka između glifova, visine i sličnih podataka [4].

3.2. Metode određivanja fonta

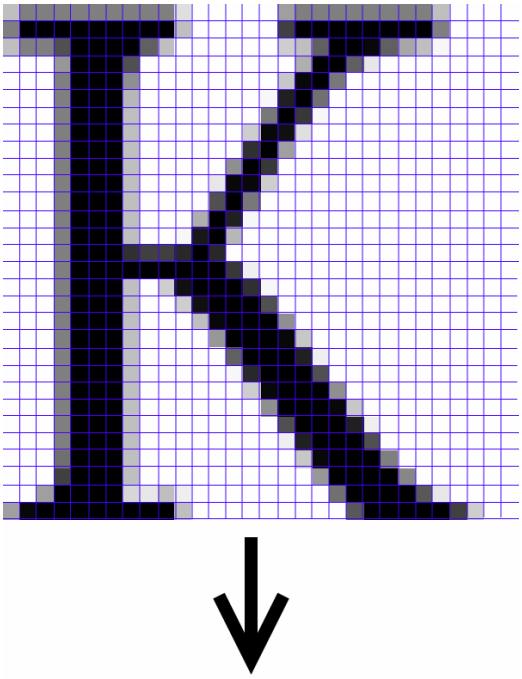
Problem koji se predstavlja pri određivanju fonta je prepoznavanje uzoraka. Ako je ulaz slika određenih dimenzija, potrebno je prepoznati uzorak poznatog slova kako bi iz njega bilo moguće odrediti font. Primjer je prikazan na slici 3.2 gdje su nacrtana velika pisana slova „T“ na sva 4 primjera, s različitim fontom.



Slika 3.2 Primjer slova „T“ različitih fontova, slike veličine 30x30 *pixela*

Lako je uočiti o kojem se slovu radi i da su sva slična oblikom no postoje manje razlike uzrokovane specifičnim fontom. Cilj aplikacije je odrediti te razlike i prema tome prepoznati font određenog znaka. Strojno učenje je grana znanosti gdje se nastoji raznim algoritmima, poput neuronskih mreža, računalu omogućiti učenje bez direktnog programiranja. Takvi algoritmi koriste podatke kako bi „učili“ i napravili predviđanja. Problem prepoznavanja fonta spada pod granu strojnog učenja koje se naziva prepoznavanje uzoraka čiji je cilj pronaći određene uzorke i pravilnosti u podacima. Ako postoje poznati uzorci koje želimo pronaći i imamo njihove primjere, tada se može koristiti nadzirano učenje gdje su algoritmu predstavljeni ulazi i željeni

izlazi a cilj je da računalo odredi pravilo kako će doći do rezultata ovisno o ulazu. Jedna vrsta prepoznavanje uzoraka može se svrstati pod klasifikaciju. Zadaća je određenim ulazima dodijeliti željeni izlaz i tako svrstati ulazne podatke u skupine. Prepoznavanje uzoraka se može definirati kao klasifikacija podataka bazirana na znanju koje se već posjeduje ili statističkim informacijama dobivenim iz uzoraka ili njihovih prikaza [5]. Uzorci su predstavljeni kao vektori vrijednosti značajki. U našem slučaju vektor podataka predstavlja slika određenog slova a vrijednosti značajki su vrijednosti *pixela* slike prikazano primjerom na slici 3.3.



```
int[] imageArray = { 191, 127, 127, 127, 127, 127, 127, 127, 127, 127, 223, 255, …}
```

Slika 3.3 Vrijednosti slike prikazani kao brojevi u vektor kolekciji C# programskog jezika

Sam postupak prebacivanja slike u vektor kolekciju biti će objašnjen u sljedećim poglavljima. Kod prepoznavanja uzoraka postoji pitanje: koliko značajki je potrebno uzeti u obzir i koje su najbolje za klasifikaciju onoga što je potrebno. Problem je što je više značajki, veća je dimenzija prostora značajki što za posljedicu ima kompliciranje problema klasifikacije, povećanje vremena i resursa potrebnih za izračune kao i povećane zahtjeva za skladištem. Taj problem naziva se i „prokletstvo dimenzionalnosti“ [6]. On označava probleme koji nastaju kada ima stotine ili više tisuća dimenzija u prostoru značajki. Neki od problema su da povećanjem dimenzija raste i volumen prostora tako da se dostupni podaci prorjeđuju a samim time potrebno je više podataka

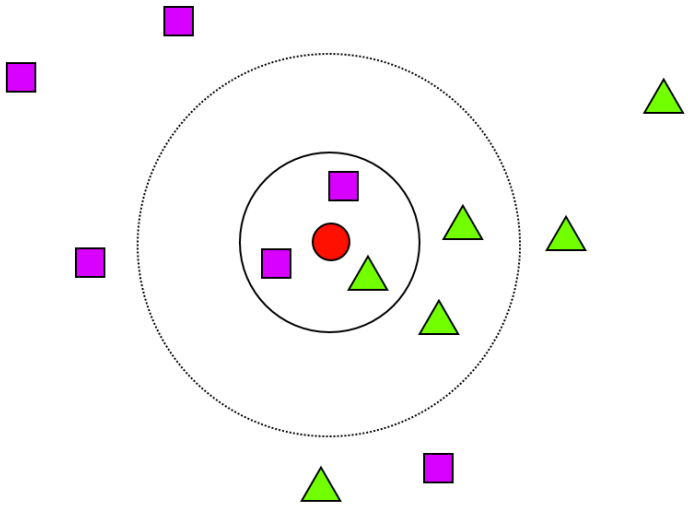
kako bi se dobili statistički značajni podaci. Također, pri većim dimenzijama grupe podataka su rjeđe i imaju manje sličnosti što otežava organizaciju podataka [6]. U radu se za učenje koriste slike veličine 30 X 30 *pixela* sa jednim kanalom za vrijednosti (engl. *grayscale*). Pri toj veličini dimenzija prostora značajki je 900, što je dovoljno veliko da se naiđe na prije spomenute probleme. U našem slučaju ne postoji problem rijetkosti podataka zbog toga što su korištene slike poznatih znakova s mogućim cjelobrojnim vrijednostima između 0 i 255 (8-bitni kanal). Iz tog razloga manjak podataka zbog povećih dimenzija ne predstavlja problem. Problem grupiranja podataka nije značajan zbog toga što na ulaz u algoritam dobivamo poznati znak a samim time i poznata važna područja, najveći utjecaj na točnost klasifikacije tu predstavlja šum i neželjeni podaci na slici.

Veći problem koji je prisutan je sama količina podataka s 900 dimenzija koje se moraju obraditi. Broj dimenzija se može smanjiti sa statističkim metodama poput metodom glavnih komponenata (engl. *Principal component analysis*, PCA) ili linearnom diskriminativnom analizom (engl. *Linear discriminant analysis*, LDA). Cilj je smanjiti broj dimenzija kako bi se smanjio broj podataka potrebnih za obradu i otkrili najbitniji podaci za klasifikaciju. Odlučeno je da takve metode neće biti korištene iz par razloga. Metoda poput PCA može otkriti strukturu podataka koja najbolje opisuje varijancu u podacima ali ne uzima u obzir željene klase (nenadzirano učenje) te se ne može garantirati da će pronađene najveće varijance u podacima biti dobra svojstva za željenu klasifikaciju. Također, male promjene u ulaznim podacima poput šuma na slici mogu uzrokovati velike promjene u podacima sa smanjenim dimenzijama koje se dobiju pomoću PCA [6]. Takvo ponašanje nije poželjno zato što je u radu aplikacije najčešći ulaz slika s fotoaparata čija kvaliteta nije najbolja i jako ovisi o kvaliteti senzora i o uvjetima u kojima je uslikana. Glavni razlog ne korištenja statističkih alata za smanjivanje dimenzija je jednostavnost operacija koje se koriste u algoritmu odabranome za određivanje fonta predstavljenome u sljedećem poglavlju. Čak i sa 900 dimenzija operacije su svedene na jednostavne operacije s matricama. Operacije se obavljaju više nego što je zadovoljavajuće brzo te ih je moguće i paralelizirati za brže izvođenje.

3.3. Algoritam *k* - najbližih susjeda (*k*NN – *k-nearest neighbors)*

Algoritam *k*NN (engl. *k-nearest neighbors*), *k* najbližih susjeda je jedan od najjednostavnijih algoritama strojnog učenja. Koristi se za klasifikaciju i za regresiju. Karakteristika što je algoritam bez parametarska metoda što znači da nije moguća statistička klasifikacija podataka,

odnosno nije poznat raspored vjerojatnosti podataka. Iz tog razloga algoritam *k*NN direktno kreira granice odluke bazirane na podacima za učenje. Algoritam *k*NN klasificira podatke bazirane na klasama *k* najbližih susjeda pri čemu koristi sličnosti između uzoraka za učenje i testnog uzorka. Klasa koja je dodjeljena novom (testnom) uzorku je jednaka većini klasa uzoraka za učenje koji su najbliži i brojem ograničenim s faktorom *k*. Iz tog razloga faktor *k* igra veliku ulogu u točnosti klasifikacije [5]. Primjer uloge faktora k prikazan je na slici 3.4



Slika 3.4 Primjer *k*NN klasifikacije ovisne o broju k

Ako je testni uzorak predstavljen krugom u sredini i $k = 3$, znači da se za klasifikaciju biraju 3 najbliža susjeda i u tom slučaju testni uzorak bi bio klasificiran kao pravokutnik. Ako povećamo broj *k* na $k = 5$ najbližih uzoraka, sada većinu imaju trokuti te bi novi uzorak bio klasificiran kao takav.

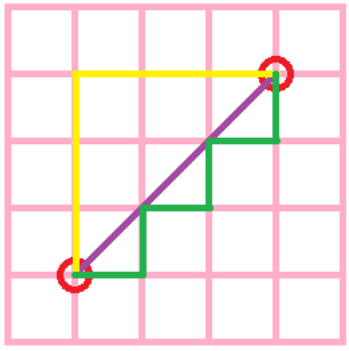
*K*NN algoritam se smatra *lijenim* algoritmom za učenje. Procesiranje podataka se odgađa sve do trenutka kada se ne zaprimi testni podatak koji je potrebno klasificirati. Prednosti lijenog učenja je intuitivnost, moguće ga je pratiti i analizirati i jednostavan je za implementaciju i paralelizaciju. Nedostatak algoritma je osjetljivost na lokalnu strukturu podataka [5]. Ulazni podaci su vektori u više dimenzionalnim značajkama prostora, u našem slučaju *grayscale* slike kako je objašnjeno u prijašnjem poglavlju. Faza učenja se sastoji od samo spremanja podataka koji služe za učenje i imaju poznate klase. U fazi klasifikacije, kada se zaprimi testni podatak, ovisno o broju *k*, dodjeljuje se klasa koju posjeduje većina susjeda (najsličniji uzorci). Kako bi se odredila sličnost podataka, potrebno je izračunati udaljenost između testnog i podataka za učenje. Jedan primjer je Euklidska udaljenost.

$$D_{ij}{}^2 = \sum_{k=1}^{n} (x_{ki} - x_{kj})^2 \tag{3-1}$$

Gdje je:

- D – udaljenost između bilo koje 2 točke i, j,
- n – broj dimenzija prostora
- x – vrijednosti točke

U našem je slučaju broj dimenzija jednak 900 (slika veličine 30 x 30) te imamo 900 operacija računanja razlike vrijednosti vektora i kvadriranja iste i na kraju operaciju sumiranja svih rezultata . Iz razloga što je prostor koji koristimo diskretan (vrijednosti slike između 0 i 255) i nije nam potrebna točna udaljenost iz razloga što će se pri mjerenju sličnosti koristiti samo omjeri udaljenosti, moguće je koristiti jednostavniju metodu naziva Manhattan udaljenost [7] (engl. *Manhattan distance, length*) koja zamjenjuje Euklidsku udaljenost. U Manhattan udaljenosti, udaljenost između bilo koje dvije točke određena je apsolutnom razlikom njihovih Kartezijskih koordinata. Primjer u slučaju dvije dimenzije dan je na slici 3.5.



Slika 3.5. Manhattan udaljenost u odnosu na Euklidsku udaljenost

Vidljivo je da po Manhattan udaljenosti postoji više puteva između dvije točke te je najkraća udaljenost jednaka 6. Direktna linija u Euklidskoj geometriji je jedinstven najkraći put i iznosi približno 4,24. Koristeći Manhattan geometriju dodatno je pojednostavnjen izračun te je formula predstavljena ispod.

$$D_{ij} = \sum_{k=1}^{n} \left|(x_{ki} - x_{kj})\right| \tag{3-2}$$

Vidljivo je da je nestala potreba za korijenom rezultata te je nastala jedna dodatna operacija, računanje apsolutnog iznos rezultata koju je računski moguće obaviti puno brže.

Za parametar *k* u *k*NN algoritmu upotrjebljen je broj 17. Nakon par ručnih testova sa različitim brojevima, 17 se pokazao kao najbolji za parametar *k* iz razloga što se dobiju bolji rezultati. Manjim brojem *k* najvjerojatniji font koji se dobije *k*NN algoritmom nije onaj font koji se testirao dok sa većim brojem *k* vjerojatnosti svih fontova se smanjuju kao i razlike između njih. Drugi način za određivanje optimalnog parametra *k* bi bila metoda unakrsne validacije (engl. *k*-fold cross-validation) gdje se testiranje provodi samo na podacima za učenje.

3.4. Generiranje podataka za učenje

Kako bi iskoristili *k*NN algoritam za određivanje fonta potrebno je imati podatke za učenje. U našem slučaju jedan podatak za učenje je vektor dobiven iz slike znaka jedinstvenog za svaki font, npr. slovo „a“. Zbog nekonzistentne kvalitete slike koju dobivamo od korisnika, kvaliteta ulaza može jako varirati i teško je obraditi ulaz na jedinstven način koji bi ujedno poboljšao i obuhvatio sve moguće varijacije ulaza. Jedna od opcija je kontrola poznatih varijabli odnosno podataka za učenje. Iz tog razloga, osim standardnih slika znakova za svaki font odlučeno je ubaciti dodatne podatke koji su varijacija standardnih slika. Svaka slika znaka je dodatno predstavljena sa 3 varijante prikazane na slici 3.6.

Slika 3.6 Znak „a“ fonta Raleway s lijeva na desno: Standardna slika, slika znaka napravljena s *anti aliasing-om*, slika obrađena Gausovim filterom (zamućenje), znak napravljen bez *hintinga*

Varijacije su na prvi pogled male no raspon varijacija predstavlja dodatno poboljšanje u točnosti predikcije fonta sa minimalnim povećanjem korištenja resursa pri raspoznavanju.

Za korištenje podataka za učenje potrebno ih je organizirati u jednu datoteku kako bi se mogli učitati kada je potrebno obraditi sliku s ulaza. Prvi korak je pokupiti sve podatke za učenje svih fontova, prebaciti ih u vektor i zapamtiti kojem fontu pripada koji znak. Prebacivanje u vektor se zasniva na prebacivanju slike u *grayscale* gdje postoji samo jedan kanal s vrijednostima *pixela* od 0 do 255. Te vrijednosti se zapisuju u vektor po stupcima i redovima slike. U našem slučaju dimenzije su 30 x 30 *pixela* te postoji 900 vrijednosti u svakom vektoru. Svaki znak predstavlja jednu klasu te kao parametre ima ime fonta kojem pripada te vektor vrijednosti. Znakovi su zatim organizirani u C# rječnik u kojem svaki ključ ima određenu vrijednost. U našem slučaju ključ je sami znak npr. „0“ ili „z“ a vrijednost je lista podataka za učenje za to slovo od svih fontova. Takav raspored omogućuje jednostavno pristupanje podacima za učenje. Ako se na ulazu nađe znak „1“, pristupom rječniku dobit će se sve vrijednosti znaka „1“ i njihov pripadajući font te je zatim jednostavno obaviti klasifikaciju *k*NN algoritmom. Sam rječnik sprema se u JSON (engl. *JavaScript Object Notation*) podatkovni format koristeći alat za serijalizaciju te se učitava iz datoteke deserijalizacijom. Primjer organizacije podataka prikazan je kodu ispod.

```
{
	"a":
	[
		{
		"array":[255,255,207,191,167,159,159,183,191,238,255,…],
		"font":"Inconsolata"
		},
		{
		"array":[255,255,255,255,255,255,255,255,255,255,…],
		"font":"Inconsolata"
		},
		…
	]
	"b": …
}
```

3.5. Postupak određivanja fonta

Kombinacijom prethodnih koraka dolazi se do potpunog algoritma za određivanje fonta sa slike. Koraci uključuju:

1. Obradu slike Tessnet alatom kojim dobivamo poziciju i veličinu svih željenih znakova na slici;
2. Učitavanje podataka za učenje u rječnik kao skup ključeva i pripadajućih vrijednosti deserijalizacijom JSON datoteke, objašnjeno u poglavlju 3.4.;
3. Iteraciju kroz skup znakova dobivenih s ulaza i pokretanje *k*NN algoritma za svaki znak gdje su podaci za učenje dobiveni iz rječnika uzimajući podatke za pripadajući znak;
4. Obrada i ispis krajnjeg rezultata gdje se svakom fontu pridruži vrijednost koja predstavlja vjerojatnost da je tekst sa slike baš taj font.

Algoritam *k*NN uključuje izračunavanje udaljenosti do svih podataka za učenje što su sve isti znakovi različitih fontova. Te udaljenosti se zatim sortiraju od najmanje do najveće i uzme se k prvih najmanjih rezultata. U rezultatu se ne koristi podatak same udaljenosti već se rezultat zasniva na principu glasovanja. Svaki predloženi font unutar *k* rezultata dobije jedan glas. Nakon obrade svih znakova potrebno je odrediti krajnju vjerojatnost da je predloženi font uistinu font sa slike. Rezultat predstavlja postotak koliko je svaki font dobio glasova i računa se formulom 3-3.

$$\mathrm{F} = \frac{\mathrm{G}}{k * \mathrm{N}} * 100\% \tag{3-3}$$

Gdje je:

- F – Rezultat obrade jednog fonta
- G – broj glasova za taj font dobiven od obrade svih znakova
- *k* – broj *k* za *k*NN algoritam, 17 u našem slučaju
- N – broj obrađenih znakova

Krajnji rezultat za jedan font je jednak broju ukupnih dobivenih glasova podijeljenih za umnoškom broja obrađenih znakova i parametra *k*. Zbroj vjerojatnosti predloženih fontova je uvijek 100% tako da taj postotak ne predstavlja sličnost predloženog fonta i fonta sa slike već postotak dobivenih glasova. Postupak je prikazan u kodu ispod.

```
// kDict predstavlja rječnik sa indexom vrijednosti koja se koristi kasnije za
// pronalaženje imena fonta i udaljenosti dobivene iz kNN algoritma

foreach (var element in kDict)
{
	// Pronalaženje imena trenutnog fonta u kDict
	var currentFont = trainList[element.Key].font;
	// Pronalaženje vrijednosti trenutnog fonta u kDict
	var currentValue = element.Value;


	if (UltimateResult.ContainsKey(currentFont))
		// Ako je font već u rječniku, dodaj jedan glas
		UltimateResult[currentFont] += 1;
	else
		// Ako font nije u rječniku, dodaj ga s jednim glasom
		UltimateResult.Add(currentFont, 1);
}


…


// Računanje i ispis krajnje vrijednosti za svaki predloženi font
foreach (var res in UltimateResult.OrderByDescending(key => key.Value))
{
	Console.WriteLine(res.Key + ": " + Math.Round(res.Value/(k*charCount),3)*100);
}
```

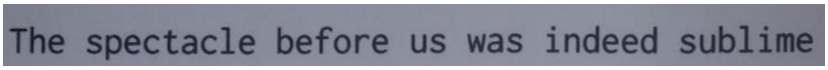
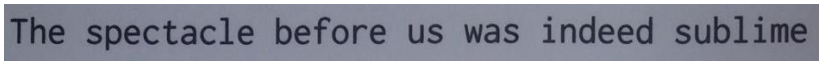
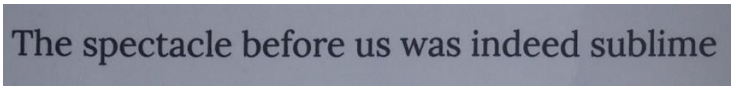
Primjer rezultata:

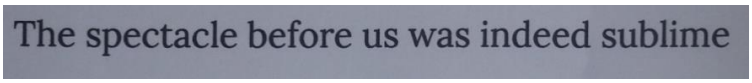
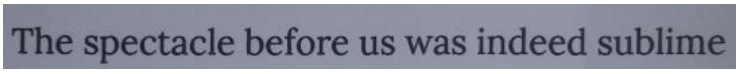
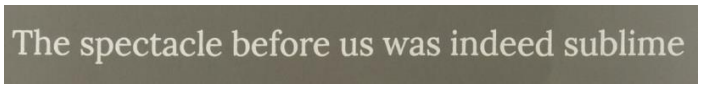
```
	Lora: 17
	Montserrat: 14.2
	Slabo_27px: 11.9
	Open_Sans: 11.6
	Roboto: 11.3
	Source_Sans_Pro: 10.8
	Inconsolata: 7.4
	Roboto_Condensed: 6.2
	Oswald: 5.9
	Raleway: 3.8
```

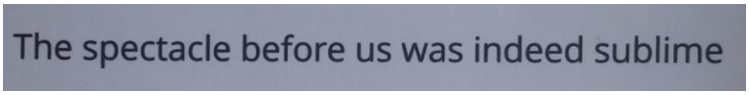
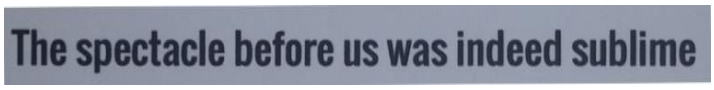
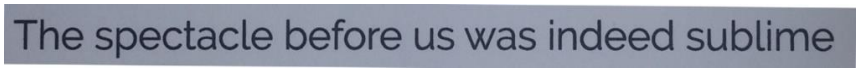
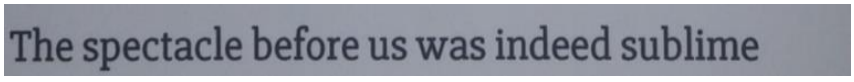
3.6. Testiranje algoritma

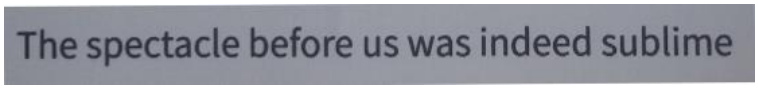
Za testiranje su iskorištene slike s mobitela. Slike uključuju tekst različitih fontova ali samo jedan font po slici. Neke slike imaju isti sadržaj samo su im promijenjene dimenzije kako bi se utvrdio utjecaj rezolucije na točan rezultat. Rezultati su prikazani u tablici ispod.

Tablica 3.1 Rezultati testiranja

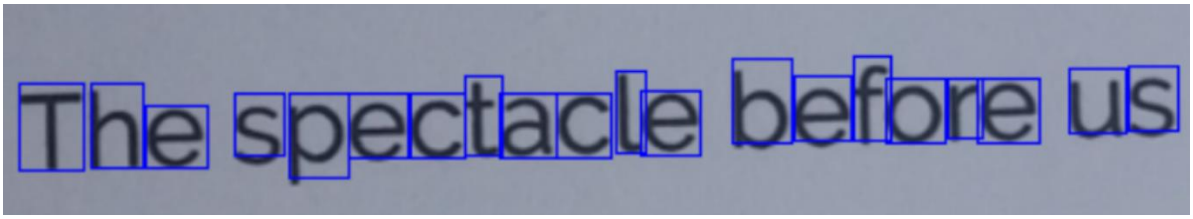
Test	**Dimenzije u *pixelima***	**Visina znakova u *px.***	**Font teksta**	**Rezultati**
				
1	742 x 55	17	Inconsolata	**Inconsolata: 15.7** Roboto: 15.4 Source_Sans_Pro: 14.5 Montserrat: 10.3 Open_Sans: 10.1 Roboto_Condensed: 9.6 Slabo_27px: 8.5 Oswald: 7.2 Lora: 5.7 Raleway: 2.9
				
2	1749 x 119	41	Inconsolata	**Inconsolata: 19.8** Roboto: 16.5 Source_Sans_Pro: 13.9 Open_Sans: 11.9 Montserrat: 9.3 Roboto_Condensed: 8.2 Slabo_27px: 6.5 Lora: 5.9 Oswald: 5.1 Raleway: 2.9
				
3	1342 x 150	35	Lora	**Lora: 17** Montserrat: 14.2 Slabo_27px: 11.9 Open_Sans: 11.6 Roboto: 11.3 Source_Sans_Pro: 10.8 Inconsolata: 7.4

				Roboto_Condensed: 6.2 Oswald: 5.9 Raleway: 3.8
				
4	889 x 88	24	Lora	**Lora: 16.3** Montserrat: 15.5 Slabo_27px: 12.6 Roboto: 11.1 Open_Sans: 10.9 Source_Sans_Pro: 10 Inconsolata: 6.5 Oswald: 6.2 Roboto_Condensed: 5.6 Raleway: 5.2
				
5	544 x 48	14	Lora	**Lora: 13.9** Montserrat: 13.4 Slabo_27px: 13.1 Roboto: 11.3 Source_Sans_Pro: 10.1 Open_Sans: 10 Oswald: 8.3 Roboto_Condensed: 7.8 Inconsolata: 6.9 Raleway: 5.2
				
6	1342 x 150	35	Lora	Oswald: 15.0 Roboto_Condensed: 13.2 Raleway: 12.6 Open_Sans: 10.5 Inconsolata: 10.1 Source_Sans_Pro: 10 Slabo_27px: 9.3 Roboto: 7.7 **Lora: 6.9** Montserrat: 4.7
				
7	961 x 85	23	Montserrat	**Montserrat: 22.2** Roboto: 15.7 Open_Sans: 12.4 Source_Sans_Pro: 11.3 Raleway: 7.2 Inconsolata: 6.7

				Slabo_27px: 6.4 Roboto_Condensed: 6.4 Lora: 6 Oswald: 5.7
				
8	898 x 105	24	Open Sans	**Open_Sans: 15.5** Roboto: 15.2 Source_Sans_Pro: 14.7 Inconsolata: 9.8 Roboto_Condensed: 9.5 Slabo_27px: 9 Montserrat: 7.5 Oswald: 7.2 Lora: 6.4 Raleway: 5.2
				
9	810 x 91	30	Oswald	**Oswald: 22.9** Roboto_Condensed: 22.1 Slabo_27px: 12.9 Source_Sans_Pro: 9.5 Roboto: 9.3 Open_Sans: 7.8 Inconsolata: 6.7 Lora: 3.8 Montserrat: 2.8 Raleway: 2.3
				
10	1317 x 97	34	Raleway	Open_Sans: 13.6 Montserrat: 13.4 Roboto: 12.7 Source_Sans_Pro: 12.6 **Raleway: 10.3** Inconsolata: 9.3 Roboto_Condensed: 8.2 Slabo_27px: 7 Lora: 6.5 Oswald: 6.4
				
11	912 x 76	25	Slabo 27px	**Slabo_27px: 18.6** Roboto_Condensed: 15 Oswald: 13.9 Source_Sans_Pro: 10.1

				Roboto: 9.6 Lora: 9.2 Open_Sans: 8.7 Montserrat: 6.9 Inconsolata: 6 Raleway: 2
				
12	558 x 59	16	Source Sans Pro	Roboto_Condensed: 18.1 Oswald: 14.9 **Source_Sans_Pro: 13.2** Roboto: 12.1 Slabo_27px: 11.1 Open_Sans: 9.6 Inconsolata: 5.9 Montserrat: 5.6 Lora: 5.6 Raleway: 3.9

Iz priloženih rezultata vidljivo je da kvaliteta slike ima dosta veliki utjecaj na klasifikaciju fonta. Što je veća rezolucija, često je veća sigurnost u pravi font te je razlika vjerojatnosti između prvog i ostalih fontova veća. U testu 6 traženi font je pri dnu rezultata, uzrok tome je što su na slici u testu 6 invertirane boje tako da je pozadina tamnija a tekst svjetliji. Iako je tekst prepoznat, trenutna implementacija algoritma *k*NN nije prilagođena za svjetliji tekst na tamnoj pozadini. Na testovima 10 i 12 pravi fontovi su na petom i trećem mjestu. U tim testovima vidljiv je utjecaj rotacije slike na kvalitetu klasifikacije. Primjer testa 10 prikazan je na slici 3.8.



Slika 3.8 Slika sa mobitela za test 10, font Raleway, uključeno označavanje znakova

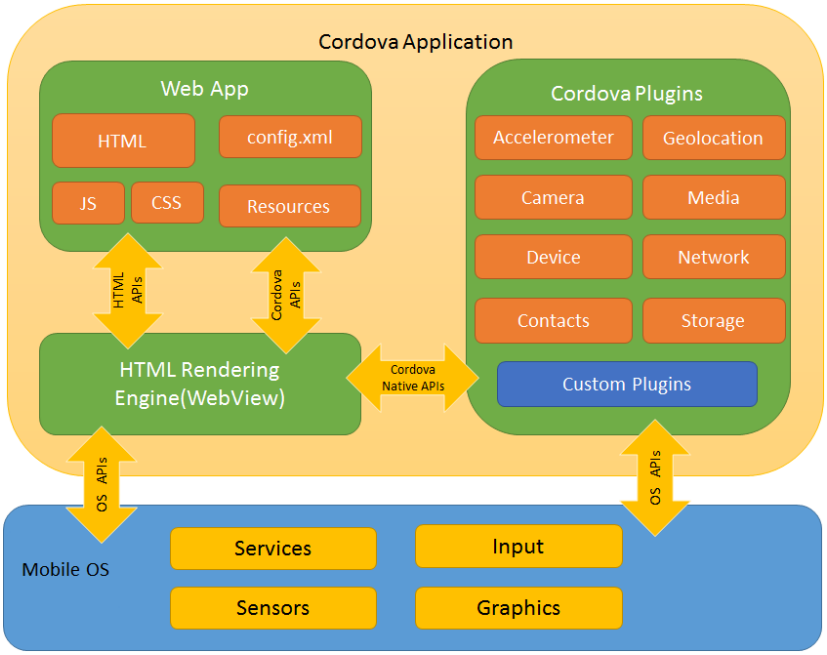
Slike su u tim slučajevima malo više rotirane te time tekst nije potpuno horizontalan. Posljedica toga je da, iako je tekst prepoznat, pozicija koju izbaci algoritam za optičko prepoznavanje znakova nije dobra, odnosno pomaknuta je, čime se ne dobiva potpuna slika znaka.

4. MOBILNA APLIKACIJA

U ovom poglavlju bit će predstavljena mobilna aplikacija, njena izrada i servisi potrebni za njeno ostvarivanje. Ukratko će biti opisano korisničko sučelje, a izgled će biti predstavljen slikama same aplikacije.

4.1. Izrada aplikacije

Mobilna aplikacija izrađena je pomoću Apache Cordova. Apache Cordova je *framework* otvorenog koda namijenjen izradi mobilnih aplikacija. Dopušta korištenje standardnih *web* tehnologija poput HTML5, CSS3 i JavaScript skriptnog jezika kako bi razvili aplikacije dostupne na većini mobilnih platformi. Aplikacije se izvode unutar *wrapper*-a na svakoj platformi i ovise o standardnim aplikacijskim programskim sučeljima (engl. application programming interface, API) kako bi pristupile mogućnostima svakog mobilnog uređaja poput senzora, podataka, statusa mreže, kameri i ostalima. Prednost Apache Cordove je mogućnost istovremenog razvoja za najpopularnije mobilne platforme bez potrebe ponovne implementacije aplikacije na novi programski jezik, korištenja novih alata i razvojne platforme. Ako već postoji web aplikacija, lakše ju je prebaciti i upakirati za distribuciju na raznim mobilnim trgovinama. Slika 4.1. prikazuje arhitekturu Apache Cordove na visokoj razini [8].



Slika 4.1. Arhitektura Apache Cordova *framework*-a na visokoj razini

Aplikacija je napravljena koristeći razvojno okruženje *Visual Studio 2015 Community* zajedno s *Tools for Apache Cordova*. Koristeći Cordovu u Visual Studio lakše je napraviti početne korake instalacije koji su većinom automatizirani. Također je jednostavnija instalacija i praćenje *plugin*-ova kojima se Cordova aplikaciji dodaju razne mogućnosti poput korištenja kamere, memorijskog sustava mobitela i slično.

4.2. Pozadinska logika aplikacije u oblaku

Za obradu slika i raspoznavanje fontova odabrana je opcija korištenja servisa u oblaku. Prednost korištenja servisa u oblaku je jedinstvenost pozadinske logike za sve korisnike, puno veća moć obrade podataka, mogućnost obrade veće količine fontova, spremanje potrebnih podataka na servisu a ne na mobitelu što štedi memorijski prostor mobitela korisnika i pisanje pozadinske logike u željenom programskom jeziku koji ne ovisi o mobilnoj platformi. Također, takvi servisi nude sve potrebne alate i mjerenja poput izrade i korištenja baza podataka, memorije, virtualnih strojeva i ostalih. Postoje i nedostaci kao što je neophodna veza s internetom, komunikacija između servisa u oblaku i mobilne aplikacije, potrebno je uložiti više vremena u izradu i takvi servisi nisu besplatni.

Kao servis u oblaku odabran je Microsoft Azure [9]. Azure podržava veliki izbor operacijskih sustava, programskih jezika, *framework*-a, alata, baza podataka i uređaja. Aplikacije je moguće izraditi sa JavaScript-om, Python-om, PHP-om, Javom i Node.js-om kao i .NET tehnologijama. To omogućava korištenje C# programskog jezika u kojem je izrađen aplikacija. Aplikacija za određivanje fonta koristi više alata koje nudi Azure servis. Koristi se: *Mobile App*, *Function App* i *Storage account*.

Mobile App je platforma kao servis (engl. *Platform as a Service, PaaS*) koja nudi razne mogućnosti za integraciju *web*-a i mobilnih aplikacija. Koristeći taj alat možemo povezati aplikaciju na korisničkom sučelju s aplikacijom u oblaku. U našem slučaju koristi se za pristup bazama podataka koje spremaju krajnji rezultat i među podatke. Glavni princip je da korisnik učitava sliku na račun za pohranu (engl. *Storage account*) za što mu treba *SAS query string* (engl. *Shared Access Signature*) koji služi kao autorizacija da se utvrdi da korisnik ima pravo mijenjati spremište. Kako bi korisnik dobio SAS, prvo se u mobilnoj aplikaciji napravi INSERT naredba u

bazu podataka koja služi kao među spremnik, vidljivo u kodu ispod. Kod je pisan u JavaScript programskom jeziku koji je glavni jezik za Cordova aplikacije.

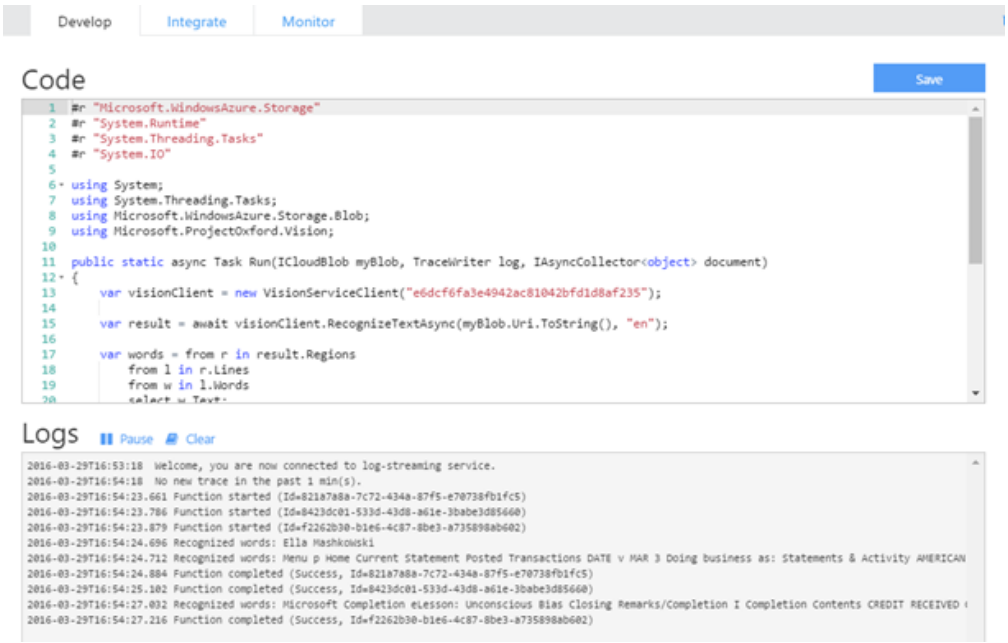
```
// Kreiranje reference na Azure Mobile Apps backend
client = new WindowsAzure.MobileServiceClient('https://fontfinder.azurewebsites.net');

// Kreiranje refence za tablice međuspremnika (todoitem) i krajnjeg rezultata
// (fontresultitem) koje se nalaze na Azure servisu
todoItemTable = client.getTable('todoitem');
fontfinderTable = client.getTable('fontresultitem');


…


// Kreiranje novog postupka za upload slike
var insertNewItemWithUpload = function (newItem, capturedFile) {
   // Pokretanje INSERT kako bi se dobio SAS
   todoItemTable.insert(newItem).then(function (item) {
       // Ako je SAS uspješno dobiven, započni sa upload-om slike
       if (item.sasQueryString !== undefined) {
           insertedItem = item;
           readImage(capturedFile);
       }
    }, handleError).then(handleError);
}
```

Na mobilnoj aplikaciji koja je pisana u Node.js-u potrebno je promjeniti rad INSERT funkcionalnosti u bazi podataka koja služi kao međuspremnik. To radimo kako bi iskoristili servis da kreiramo SAS koji je potreban korisniku za *upload*. Neke od potrebnih promjena prikazane su u kodu ispod.

```
// Uređivanje insert funkcionalnosti
table.insert(function (context) {
    console.info(context.item.resourceName);
    console.info(context.item.containerName);
…
// Omogućavanje write pristupa spremištu slika sljedećih 5 minuta
var sharedAccessPolicy = {
      AccessPolicy: {
        Permissions: azure.BlobUtilities.SharedAccessPermissions.WRITE,
```

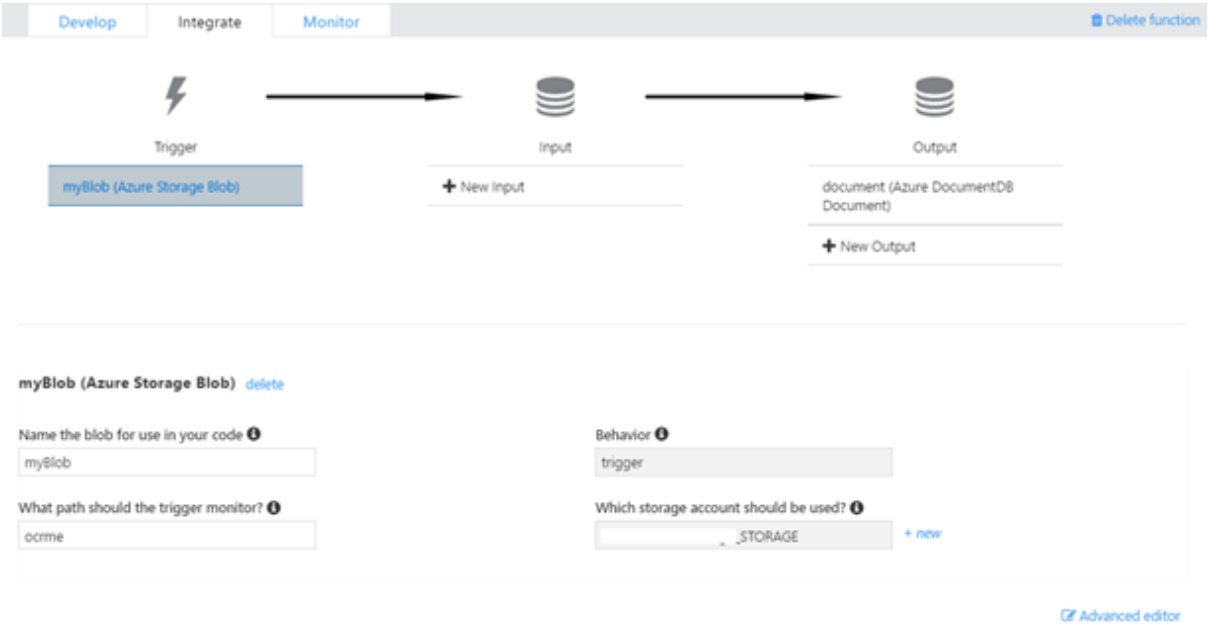
```
        Start: new Date(),
        Expiry: new Date(new Date().getTime() + 5 * 60 * 1000)
      },
};

// Generiranje SAS query string-a
var sasQueryUrl =
      blobService.generateSharedAccessSignature(context.item.containerName,
      context.item.resourceName, sharedAccessPolicy);


// Postavljanje stupca sasQueryString na bazi podataka koja služi kao međuspremnik
// Ako je SAS uspješno stvoren, korisnik sada ima mogućnost uploada slike preko SAS
// koji se čita iz baze podataka
context.item.sasQueryString = sasQueryUrl;
```

Kada korisnik dobije SAS, moguće ga je učitati i napraviti XML Http zahtjev kojim učitavamo sliku. Kada se slika nalazi na računu za pohranu, pokreće se funkcijska aplikacija. Funkcijska aplikacija (engl. *Function App*) je servis koji nudi računalnu moć na zahtjev (engl. *Compute-on-demand*) i koja ima razne okidače. Kada se pokrene okidač što mogu biti razni događaji, pokreće se funkcijska aplikacija odnosno računalni kod. Aplikacija se može programirati iz Internet preglednika. Primjer izgleda sučelja se nalazi na slici 4.2.



Slika 4.2. Programski kod i dnevnik događaja za Azure *Function App*

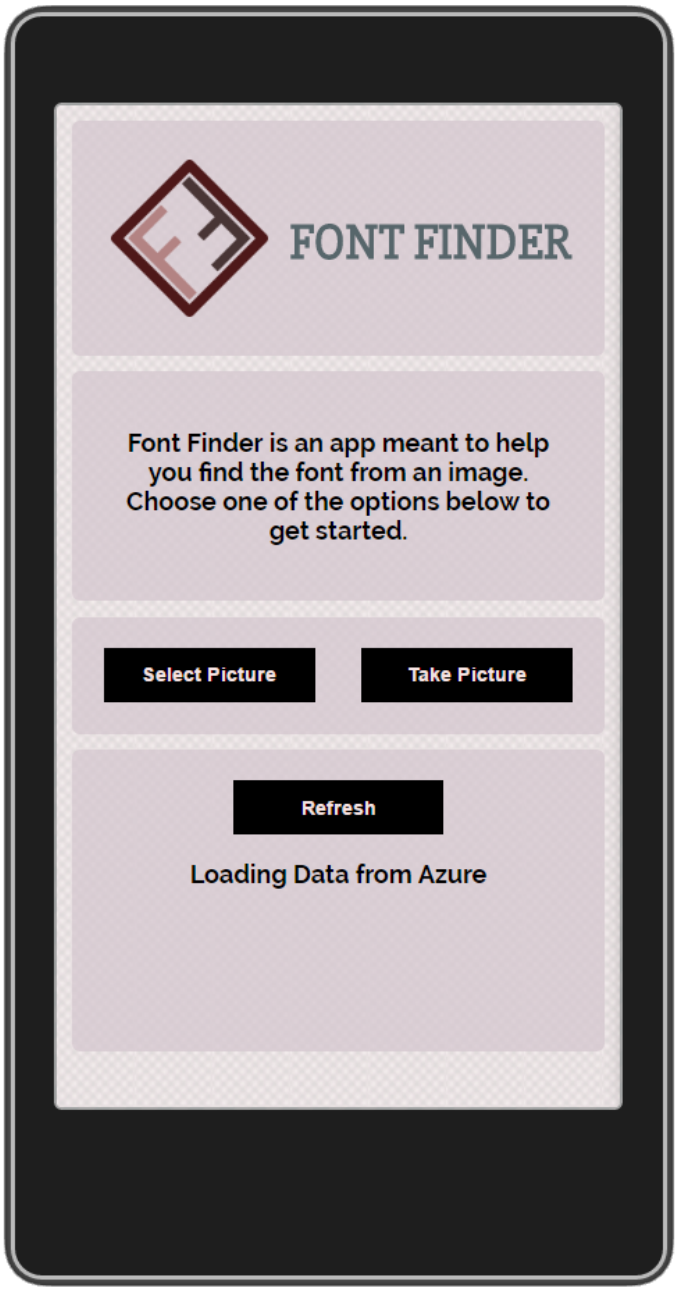
Funkcijska aplikacija kao okidač ima pojavljivanje slike u povezanom spremištu koju onda može iskoristiti. Kod koji se nalazi u funkcijskoj aplikaciji je algoritam za raspoznavanje fontova. Jedina razlika je što se na kraju funkcijske aplikacije rezultat ne ispisuje nego se sprema u bazu podataka krajnjeg rezultata. Kako korisnik ima pristup toj bazi podataka, može vidjeti svoje rezultate u korisničkom sučelju. Primjer postavljanja okidača može se vidjeti na slici 4.3.



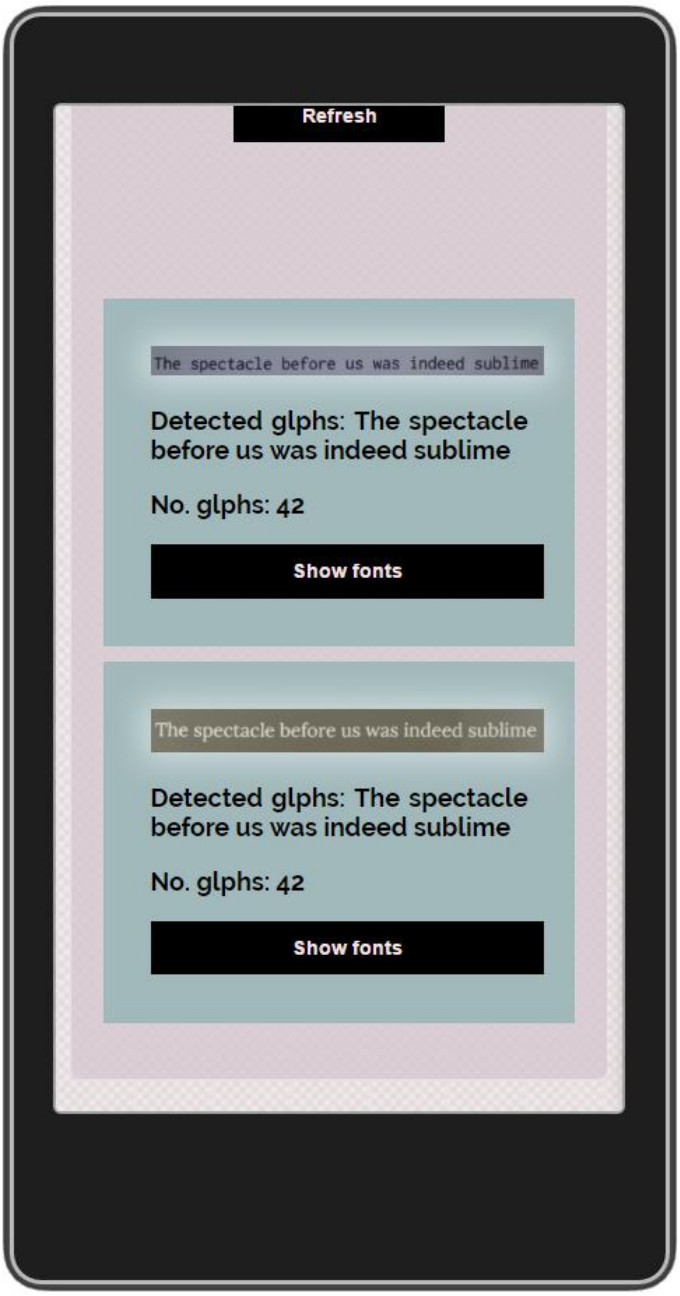
Slika 4.3. Primjer postavljanja okidača, ulaza i izlaza za Azure *Function App*

4.3. Korisničko sučelje i mogućnosti

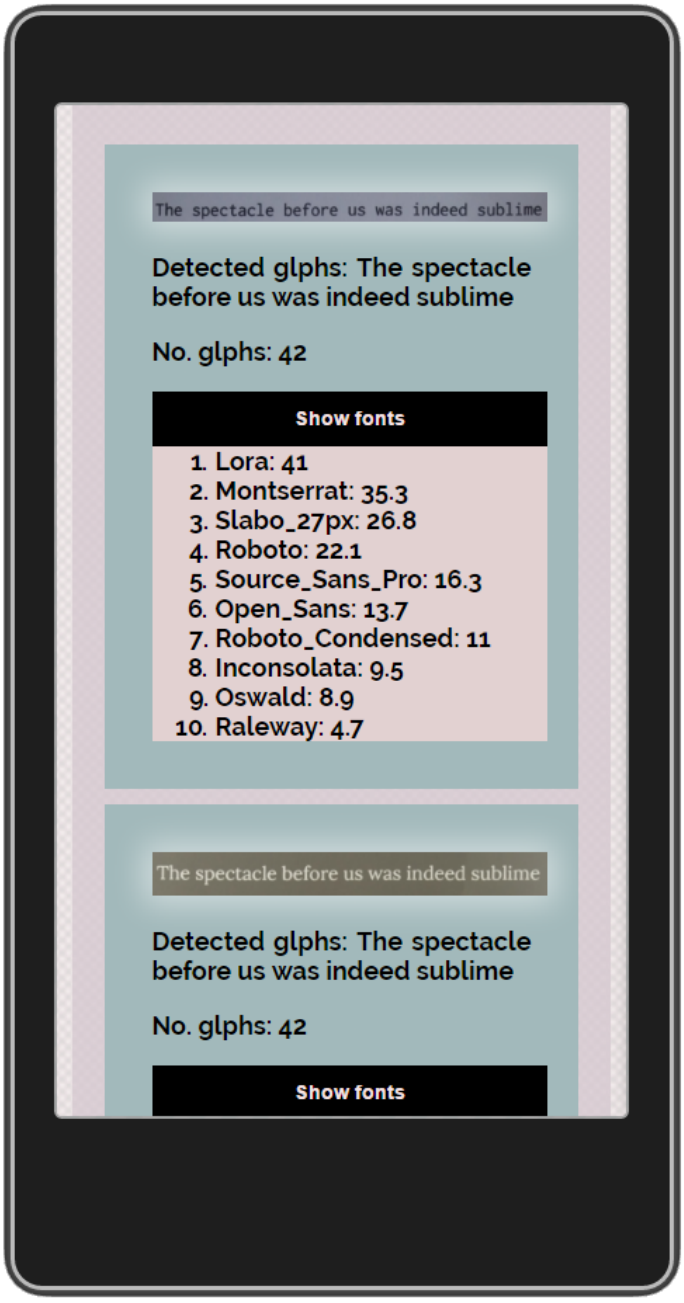
Aplikacija je dizajnirana s minimalističkim sučeljem kako bi se što lakše koristila. Postoji mogućnost slikanja koristeći ugrađenu kameru kao i mogućnost korištenja već postojeće slike na mobilnom uređaju. Odabirom slike na jedan od ta dva načina pokreće se proces traženja fonta nakon kojega se pojavljuju rezultati s najsličnijim fontovi sa pripadajućim postotkom sličnosti. Rezultati se pojavljuju nakon pritiska na gumb kako bi sučelje bilo što manje zatrpano. Također se pri rezultatu pojavljuje slika koja je obrađena. Proces traženja fonta i sam proces unutar mobilne aplikacije opisan je u prijašnjim poglavljima. Ispod se nalaze slike koje predstavljaju neke od izbornika i dizajn aplikacije.



Slika 4.4. Početni ekran

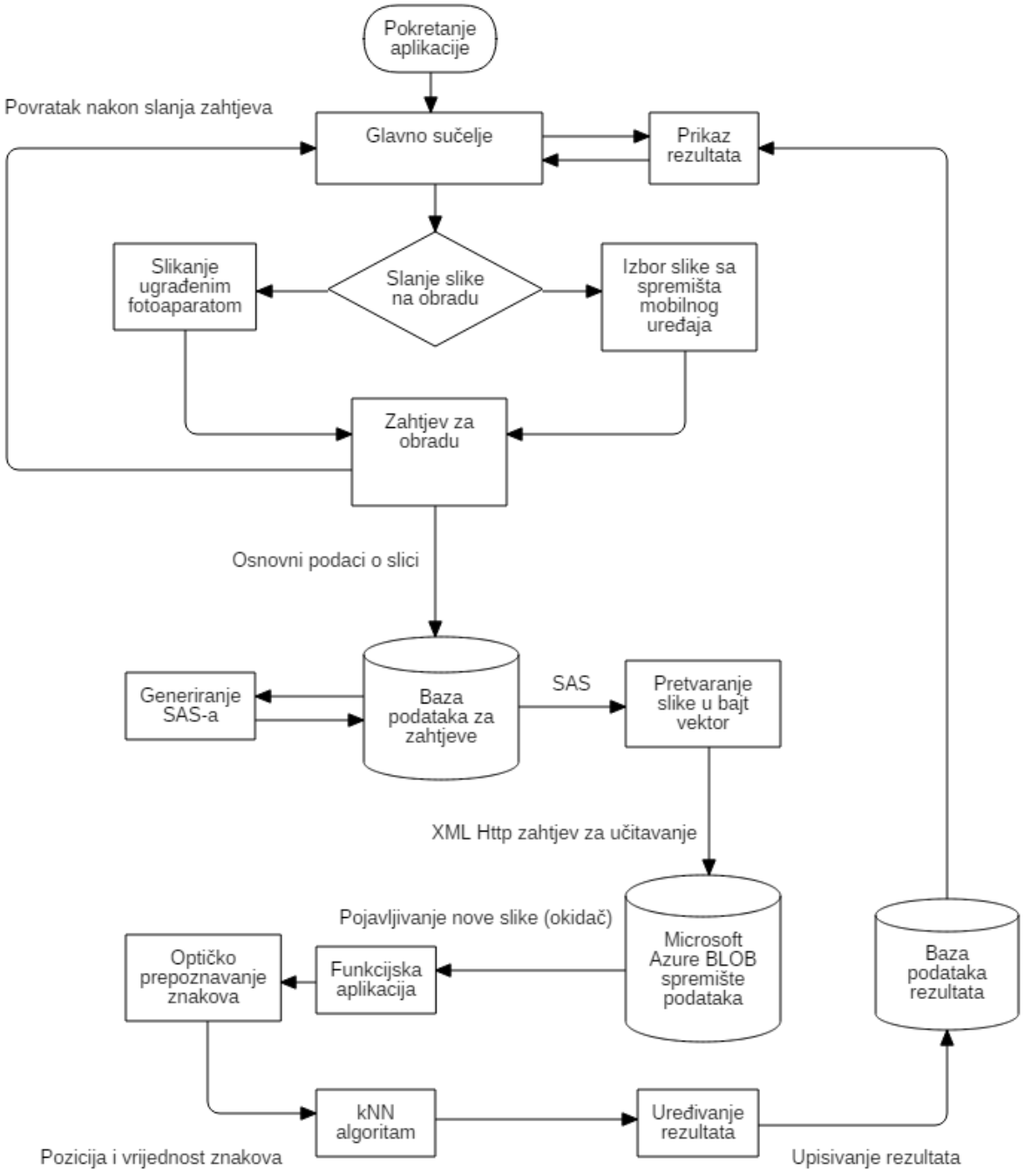


Slika 4.5. Rezultati obrade slike i traženja fonta



Slika 4.6. Prikaz rezultata

Nakon svih predstavljenih elemenata, na slici 4.7. nalazi se dijagram toka korištenja mobilne aplikacije sa pojednostavljenim prikazom toka procesa u pozadini.



Slika 4.7. Dijagram toka korištenja aplikacije

5. ZAKLJUČAK

U radu je prikazano prepoznavanje teksta sa slike koristeći alat otvorenog koda Tesseract te je opisan izlaz prilagođen daljnjim potrebama kao i sam algoritam prepoznavanja znakova. Opisani su načini pohrane i struktura fontova kao i metode za njihovo određivanje što uključuje obradu slike znakova i pretvaranje iste u prigodan format podataka. Opisan je rad i implementacija *k*NN (engl. *k-nearest neighbour*) algoritma što uključuje izradu podataka za učenje. Na kraju je sve zaokruženo sa cjelokupnim procesom određivanja fonta i s testovima izvršenim na sam proces kako bi se utvrdila valjanost. U zadnjem poglavlju rečeno je nešto o Apache Cordova *framework*-u u kojem je izrađena mobilna aplikacija te je objašnjen dio postupka izrade i korištenja Azure servisa kao *backend*-a. Na kraju je predstavljen dizajn mobilne aplikacije i priložene su slike same mobilne aplikacije.

LITERATURA

[1] Rémi Thomas, *Tessnet2 a .NET 2.0 Open Source OCR assembly using Tesseract engine*, http://www.pixel-technology.com/freeware/tessnet2/, 29.10.2016.

[2] R. Smith, *An Overview of the Tesseract OCR Engine*, Google Inc., http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf, 30.6.2016.

[3] Tesseract Frequently Asked Questions, https://github.com/tesseract-ocr/tesseract/wiki/FAQ, 30.6.2016.

[4] V. Gaultney, M. Hosken, A. Ward, *An Introduction to TrueType Fonts: A look inside the TTF format*, SIL International, 23.5.2003.

[5] M. Narasimha Murty, V. Susheela Devi, *Pattern Recognition, An Algorithmic Approach*, ISBN 978-0-85729-494-4, Springer, 2011.

[6] Geoff Dougherty, *Pattern Recognition and Classification, An Introduction*, ISBN 978-1-4614-5322-2, Springer, 2013

[7] Kevin Thompson, T. Dray, *Taxicab Angles and Trigonometry*, The Pi Mu Epsilon Journal, Worcester, MA. Vol. 11, No. 2 (Spring 2000), pp. 87-96, 1997.

[8] Apache Cordova pregled, https://cordova.apache.org/docs/en/latest/guide/overview/, 29.10.2016.

[9] Microsoft Azure pregled, https://azure.microsoft.com/en-us/overview/what-is-azure/, 29.10.2016.

SAŽETAK

Glavni zadatak diplomskog rada je odrediti font sa slike teksta preko mobilne aplikacije. Prvi korak je detekcija teksta sa slike što je ostvareno preko programa Tessnet koji obradom slike preko alata Tesseract daje lokaciju, veličinu i vrijednost pronađenog znaka. Nakon što se dobiju te potrebne informacije, slike slova ili brojki se koriste u *k*NN algoritmu za samu klasifikaciju fonta. Mobilna aplikacija koja koristi Apache Cordova *framework* prikazuje te rezultate korisniku koristeći Azure servise za obradu slike i kao *backend*.

Ključne riječi: mobilna aplikacija, obrada slike, optički čitač znakova, font, k najbližih susjeda, *k*NN, strojno učenje, klasifikacija

MOBILE APPLICATION FOR DETERMINING THE FONT OF THE TEXT FROM AN IMAGE

The main task of the thesis is to determine the font from an image using a mobile application. The first step in doing this is the detection of the text in an image which is accomplished using Tessnet. Tessnet using image processing via Tesseract provides the location, size and type of glyph found. With this information, the *k*-nearest neighbor algorithm is used to determine the most likely font. The mobile application uses Apache Cordova framework to show the data to the user while Azure services is used for image processing and as a backend to the mobile application.

Key words: mobile application, image processing, optical character recognition, OCR, font, k-nearest neighbor, *k*NN, machine learning, classification

ŽIVOTOPIS

Emil Silađi, rođen 5.2.1993. u Osijeku. Pohađao je 2. Jezičnu gimnaziju u Osijeku. Od 2011. do 2014. je pohađao i uspješno završio Sveučilišni preddiplomski studij smjera Računarstvo na Elektrotehničkom fakultetu Osijek. Na istom fakultetu od 2014. pohađa Sveučilišni diplomski studij, smjer Procesno Računarstvo. Autor je rada pod nazivom "*Video meteor detection filtering using soft computing methods*" objavljenog u *Proceedings of the International Meteor Conference*, Mistelbach, Austria, 2015.

PRILOZI

PRILOG 1 – Programski kod za mobilnu aplikaciju

PRILOG 2 – Programski kod korišten za realizaciju i testiranje *k*NN algoritma

PRILOG 3 – Programski kod korišten za generiranje podataka za učenje

PRILOG 4 – Programski kod koji se nalazi na Azure servisima i služi kao *backend* aplikacije