UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET

RAZVOJ SOFTVERSKOG SISTEMA ZA PREPOZNAVANJE SAOBRAĆAJNIH ZNAKOVA PRIMENOM TEHNIKA MAŠINSKOG UČENJA

Diplomski rad

Mentor:
Doc. dr Dražen Drašković

Kandidat:
Danilo Lalić 2017/0501

Beograd, septembar 2021.

Sadržaj

1 Uvod

Tokom istorije ljudski rod je prošao kroz tri perioda razvoja: lovac-sakupljač, uzgajivač i period industrijskih revolucija. Ono što je zajedničko za ova razdoblja je način na koje je čovek koristio alat i mašine. Do skoro su alati i mašine bili pod našom punom kontrolom i nisu imali veliku mogućnost odlučivanja[1] (nož seče tamo gde ga ruka navodi, šporet greje onom temperaturom na koju je podešen, u ovom radu svaka reč je morala biti eksplicitno napisana…). Danas se čovečanstvo nalazi na pragu četvrte industrijske revolucije[2] u kojoj će najveću ulogu i značaj imati veštačka inteligencija, koja će omogućiti mnogo veći nivo autonomije mašina i alata. Time ćemo od primitivnih pomagala dobiti asistente i savetnike u radu. Jedan od mnogih primera u kojima je ovako nešto već sprovedeno je *Github copilot* koji na osnovu opisa funkcionalnosti koju programer želi da ostvari samostalno generiše odgovarajući kod.

Automobili će takođe u skoroj budućnosti postati mašine koje će u potpunosti same odlučivati (vožnja će postati autonomna). Da bi tako nešto bilo moguće potrebno je koristiti oblast veštačke inteligencije koja se naziva mašinsko učenje. Mašinsko učenje se razlikuje od klasičnog programiranja. Glavna razlika je u tome što programer ne piše pravila i parametre već ih algoritam samostalno određuje. To se postiže skupovima podataka koji se dostavljaju algoritmu, na osnovu njih se podešava model koji sadrži neophodne informacije za pravilno funkcionisanje programa. Mašinsko učenje se u velikoj meri primenjuje u kompjuterskom vidu.

U ovom radu je upoređeno par metoda za detekciju i klasifikaciju saobraćajnih znakova na slici. Takođe dat je i opis implementacije jednog od takvih sistema. Implementacija sistema koja će ovde biti opisana se može podeliti na deo za detekciju, koji se zasniva na Histogramima orijentisanih gradijenata (HOG) i Mašinama podržavajućih vektora (MPV), i na deo za klasifikaciju, koji je realizovan konvolucijskom neuronskom mrežom.

---

[1] Naravno da su i pre određene mašine mogle da vrše složene operacije i da donekle odlučuju samostalno. Primer za to su uređaji koji čine neku računarsku mrežu. Njima nije neophodno eksplicitno zadati kojom rutom bi trebalo da transport podataka ide, već na osnovu zagušenja i protočnosti mreže oni sami donose odluku kako da međusobno razmene podatke. I pored ovakvih primera, do skoro nismo videli autonomna vozila ili sisteme koji pružaju podršku za donošenje poslovnih odluka za koje bi bilo potrebno imati veliki tim stručnjaka (*IBM Watson*).

[2] Trenutno se među istoričarima vodi rasprava o tome da li smo trenutno u trećoj ili drugoj industrijskoj revoluciji, takođe postoji i teorija da se tačno razdoblje industrijalizacije ne može ustanoviti, pa zbog toga i ne treba pričati o bilo kakvoj revoluciji.

U drugom poglavlju biće dat teorijski pregled i opis tehnika i metoda koje će biti korišćene u konkretnoj implementaciji sa posebnim naglaskom na HOG i MPV. U trećem poglavlju će biti reči o konvolucijskim neuronskim mrežama. Biće opisani slojevi koji ovaj tip mreže čine jedinstvenim, način na koji se neuronske mreže obučavaju i takođe je dat osvrt na problem preobučavanja. U četvrtom poglavlju je opisana konkretna implementacija sistema, koju prati izvorni kod. U petom poglavlju se vrši predstavljanje konačnih performansi sistema i njihovo poređenja sa različitim radovima. Šesto poglavlje predstavlja zaključak u kome je dat osvrt na cilj i rezultate ovog rada.

2 Detekcija

U ovom poglavlju će biti opisane glavne karakteristike sistema za detekciju objekata na slici (saobraćajnih znakova) korišćenjem HOG-a i MPV-a. Pored toga biće opisani najvažniji algoritmi i navedeni razlozi za njihovu primenu. Takođe, ovde su objašnjeni parametri algoritama i navedene vrednosti koje bi ti parametri trebalo da poseduju.

2.1 Opis sistema

Raspoznavanje saobraćajnih znakova se vrši tako što prvo datoj slici pomoću HOG-a odredimo karakteristike (HOG karakteristike se kao podaci čuvaju u obliku niza; HOG je detaljno objašnjen u narednom poglavlju). Nakon toga se HOG karakteristike šalju MPV-u koji kao izlaz daje vrednost koja govori o tome da li se znak nalazi na slici ili ne. Uzimanje u obzir svakog piksela može dovesti do toga da problem postane jako kompleksan računski i memorijski. Zbog toga je izdvajanje svojstava od interesa pre same analize veoma bitan korak.

Slika 1. Tehnika „klizećeg prozora“. [7]

S obzirom da na slici može biti više znakova i da se oni mogu nalaziti na različitim mestima, naivno bi bilo da kompletnu sliku velike rezolucije propustimo kroz gore opisani sistem. Takav pristup bi onemogućio da kažemo koliko saobraćajnih znakova se nalazi na slici i na kojim mestima, sve što bi smo mogli da kažemo je da li se znak nalazi na slici ili ne. Takođe, na ovaj način rad MPV-a bi bio mnogo teži. Razlog za to je veći nivo kompleksnosti kome MPV treba da se prilagodi. Kompleksnost proizilazi iz toga što možemo imati neodređen broj znakova na slici koji mogu biti na bilo kom mestu, pa bi zbog toga bilo jako teško istrenirati MPV koji bi imao dosta veliki nivo opštosti i preciznosti. Usled toga, neophodno je koristiti tehniku „klizećeg prozora“ (slika 1). Ideja kod gore navedene tehnike je da

u svakom koraku izdvojimo određeni segment slike i da analiziramo da li on sadrži znak ili ne. Veličina segmenta može biti proizvoljna i zavisi od toga koliku preciznost i performanse sistema želimo da postignemo. Manji segment će doprineti većoj preciznosti, jer će omogućiti prepoznavanje objekata manjih dimenzija, ali će dovesti do sporijeg izvršavanja, jer će povećati broj računarskih operacija (pošto su segmenti manji biće ih više, samim tim biće neophodno analizirati više segmenata). Takođe, bitno je voditi računa i o tome kojih dimenzija će objekat (znak) biti na slici. Ako je deo slike preveliki ili premali znak možda neće biti prepoznat iako se nalazi na slici.

Nakon što odredimo dimenzije potrebno je odrediti intenzitet pomeranja segmenta po slici. Pod intenzitetom se podrazumeva koliko će piksela segment biti pomeren po horizontali i vertikali u svakom novom koraku (odatle i naziv „klizeći prozor“). Kao i u prethodnom slučaju manji intenzitet će dovesti do veće preciznosti, jer će se smanjiti mogućnost da se objekat jednim delom nalazi u jednoj, a drugim u drugoj poziciji segmenta, ali će veće učestanosti pozicija segmenta rezultovati sporijom analizom slike.

Objekti se na slici mogu naći u različitim dimenzijama. Da bismo mogli da ih na pravilan način detektujemo potrebno je da koristimo piramidalno skaliranje. Nakon što se slika u potpunosti obradi njene dimenzije se smanjuju za određeni faktor nakon čega se slika ponovo obrađuje, samo što sada segment obuhvata veći deo slike.

Takođe, javlja se još jedan problem, a to je da jedan isti objekat na slici može biti detektovan više puta. Ovakav scenario je moguć usled malog intenziteta pomeranja prozora i usled piramidalnog skaliranja. Da ne bi bio prijavljen veliki broj objekata kada je reč samo o jednom koristi se algoritam NMS (*Non-maximum Suppression*).

2.2 Histogram orijentisanih gradijenata

Glavna ideja kod Histograma orijentisanih gradijenata je mogućnost da neki oblik bude detektovan na slici zahvaljujući intenzitetu gradijenta i konturama koje taj oblik proizvodi. Postoji puno sličnih pristupa (Histogrami orijentisanih ivica, kontekst oblika…). Ipak, ono što ovaj metod čini jedinstvenim i boljim je činjenica da se izračunavanja vrše nad takozvanim ćelijama, koje zapravo predstavljaju uniformne delove segmenta slike koji se analizira, i da se dodatno vrši blokovska normalizacija kontrasta nad dobijenim vrednostima iz ćelija.

2.2.1 Računanje gradijenata

Prvi korak ove metode je računanje gradijenata. Računanje se izvodi na taj način što se nad pikselima slike primeni maska diskretnih izvoda. Maska koja je u radu Dalala i Trigsa [1] dala najbolje rezultate je [-1 0 1]. Ona se primenjuje tako što se svaki njen element množi sa vrednošću odgovarajućeg piksela, nakon čega se te vrednosti sabiraju i dobija se rezultat po određenoj koordinati.

Maska se koristi po horizontali dok se njena transponovana verzija koristi po vertikali. Zatim je potrebno odrediti magnitudu i ugao gradijenta, što se postiže formulama (2.2.1) i (2.2.2) respektivno.

$$g = \sqrt{({g_x}^2 + {g_y}^2)} \tag{2.2.1}$$

$$\theta = arctan\frac{g_y}{g_x} \tag{2.2.2}$$

S obzirom da slike mogu biti u boji, pitanje je na koji način se u tom slučaju računa gradijent nekog piksela koji ima informacije o crvenoj, plavoj i zelenoj. Odgovor je veoma jednostavan: gradijent se računa za sva tri kanala i u obzir se za određeni piksel uzima gradijent sa najvećom magnitudom. Za masku možemo uzeti i neke druge vrednosti kao što su Sobelov operator ili necentralizovanu masku [-1 1]. Ipak, kao što je već rečeno najbolje rezultate daje [-1 0 1].

2.2.2 Kreiranje histograma

Naredni korak je kreiranje histograma. Svaki piksel unutar jedne ćelije sada nosi informacije o magnitudi i uglu gradijenta, dok svaka klasa histograma obuhvata određeni raspon uglova. Takođe, klase su uniformne (obuhvataju jednak raspon uglova). Na osnovu veličine ugla gradijenta piksela magnituda tog gradijenta se dodaje određenoj klasi. Ovaj postupak se ponavlja za svaki piksel u datoj ćeliji.

Broj klasa je proizvoljan, mada je eksperimentalno dokazano da performanse deskriptora rastu do 9 klasa, nakon čega se povećavanjem broja klasa ne dobijaju značajno bolje performanse [1]. Takođe, možemo izabrati da li želimo da imamo ceo krug (0^o – 360^o), takozvani označeni, ili polukrug (0^o – 180^o), takozvani neoznačeni. Kod neoznačenog se vrednosti raspoređuju na taj način što se prvih 180 stepeni rasporedi uobičajeno, nakon čega se preostalih 180 stepeni posmatraju kao da počinju od nula i raspoređuju u iste klase kao prethodni. Naravno histogram celog kruga ima dva puta vise klasa od histograma polukruga. Ovde se pokazalo da polukrug zapravo daje bolje rezultate od celog kruga.

2.2.3 Normalizacija

Usled različitog osvetljenja na slici, gradijenti slike koja prikazuje istu scenu mogu biti različiti. Na primer, ako bismo sve vrednosti piksela neke slike pomnožili sa određenim faktorom dobili bismo svetliju sliku, ali bi takođe i gradijenti bili veći, samim tim i histogrami bi imali različite vrednosti u odnosu na početnu sliku. Način na koji ovo možemo izbeći je normalizacijom vektora (histograma). Uzmimo za primer dva vektora: [20 20 20] i [100 100 100]. Iako naizgled potpuno različiti ova dva vektora mogu predstavljati delove histograma dveju slika koje prikazuju istu scenu, ali pod različitim osvetljenjem. Ako bismo elemente oba vektora kvadrirali, pa zatim sabrali i korenovali, a nakon toga

vektore podelili tim rezultatima, dobili bismo sledeće vrednosti: [0.577 0.577 0.577] i [0.577 0.577 0.577].

Prikazani način normalizacije vektora je L2 norma. Pored ovog načina možemo koristiti i druge, na primer L1 normu ili L2-hys (L2 norma praćena odsecanjem vrednosti na neku zadatu maksimalnu vrednost). Opet pokazano je da je L2 norma najbolja [1]. Gore opisana normalizacija se sprovodi nad blokovima koji predstavljaju grupisane ćelije. Postoji dva osnovna načina grupisanja ćelija: pravougaono ili kružno. U radu Dalala i Trigsa [1] pokazano je da je najuspešniji način grupisanja pravougaoni sa 3x3 ćelije koje imaju 6x6 piksela. Takođe je poželjno da se blokovi međusobno preklapaju. Na taj način jedna ćelija veći broj puta doprinosi normalizaciji gradijenata.

2.2.4 Dobijanje finalnog vektora

Nakon što se veličina bloka i broj ćelija koje će se preklapati kod susednih blokova odredi započinje se proces normalizacije. Histogram svake ćelije se posmatra kao vektor i histogrami svih ćelija u jednom bloku se spajaju u jedan veći vektor koji se normalizuje. Nakon što se obradi jedan blok prelazi se na sledeći. Svi novodobijeni vektori se zatim spajaju u krajnji vektor koji predstavlja konačni rezultat metode Histograma orijentisanih gradijenata.

2.3 Mašina podržavajućih vektora

Korak koji sledi nakon izračunavanja karakteristika date slike je određivanje da li se na slici nalazi traženi objekat ili ne. Možemo pokušati da ovaj problem rešimo klasičnim programiranjem. Takav program bi bio neverovatno kompleksan i ne bi bio dovoljno robustan. Bolji način bi bio da iskoristimo neki od algoritama mašinskog učenja. U implementaciji za prepoznavanje saobraćajnih znakova, koja će kasnije biti detaljno objašnjena, iskorišćena je mašina podržavajućih vektora (MPV). MPV je veoma pogodna za ovakav problem jer je inicijalno dizajnirana da rešava diskriminatorne probleme, a tek kasnije je generalizovana za rešavanje problema sa više od 2 klase.

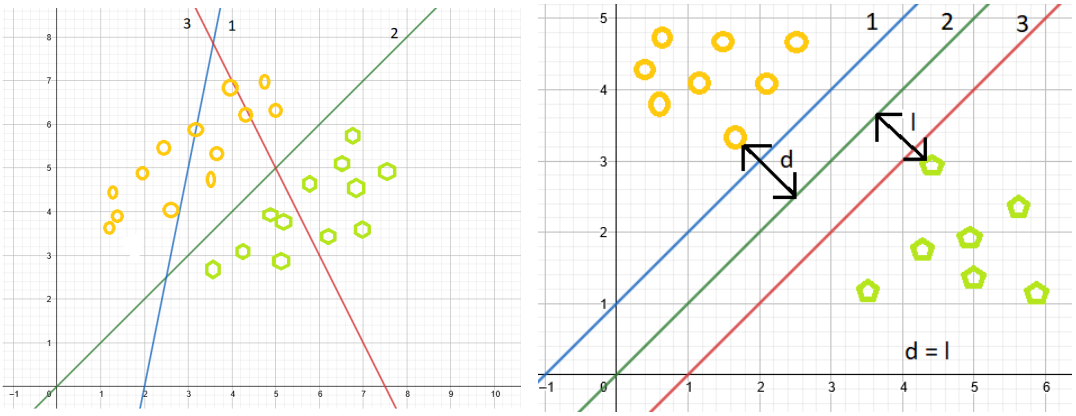
MPV spada u skup algoritama nadgledanog učenja. Osnovna karakteristika ovakvih algoritama je ta što su skupovi podataka označeni. Propuštanjem ulaznih podataka algoritam podešava vrednosti parametara da bi se vrednosti na izlazu poklopile sa datim oznakama. Kada je potrebno odrediti kojoj klasi pripada neki ulazni podatak, i to u slučaju kada imamo samo dve klase, MPV teži pronalaženju hiperravni koja vrši preciznu podelu između dve klase i nalazi se na što većem razmaku od svih uzoraka.

Da bismo istrenirali ovakav model potrebno je prvo da skup podataka podelimo na pozitivne i negativne uzorke, koje možemo označiti sa vrednostima nula i jedan. Zatim će, u odnosu na ulazne podatke i njihove oznake, algoritam pronaći ranije pomenutu hiperravan koja razdvaja ova dva tipa. Nakon toga je moguće odrediti, u zavisnosti na kojoj strani hiperravni se podatak nalazi, kojoj klasi

pripada podatak koji se nalazi van skupa podataka za treniranje. Uz informaciju o klasi podatka moguće je dobiti i informaciju o stepenu poverenja. To je moguće ostvariti izračunavanjem udaljenosti uzorka od hiperravni. Ako je dužina velika onda je i stepen poverenja veći, u suprotnom stepen poverenja opada.

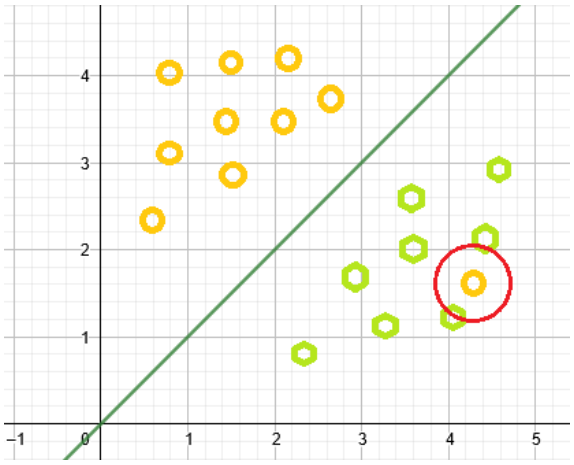
Neke od prednosti MPV-a su mogućnost rešavanja nelinearnih problema, efikasnost kada je broj dimenzija veći od broja uzoraka za treniranje, efikasno korišćenje memorije… Iako danas postoje algoritmi koji su bolji od MPV-a, navedene vrline i jednostavnost su razlog zbog koga ovaj algoritam i dan danas ima veliku primenu. Problemi za koje se koristi su ponekada dosta kompleksni.

Ostalo je još objasniti kako se određuje hiperravan. Radi jednostavnosti koristićemo dvodimenzionalni prostor, hiperravan će zapravo biti prava linija i imaćemo samo dve klase. Počnimo od najjednostavnijeg primera koji je prikazan na levoj strani slike broj 2. Tu možemo videti tri prave koje presecaju ravan na kojoj su izdvojeni uzorci dveju klasa. Intuitivno je lako zaključiti da bi ispravan izbor bila prava broj 2, jer jedino ona razdvaja elemente dva skupa. Ovim je ispunjen prvi zahtev, a to je da hiperravan razdvaja uzorke.



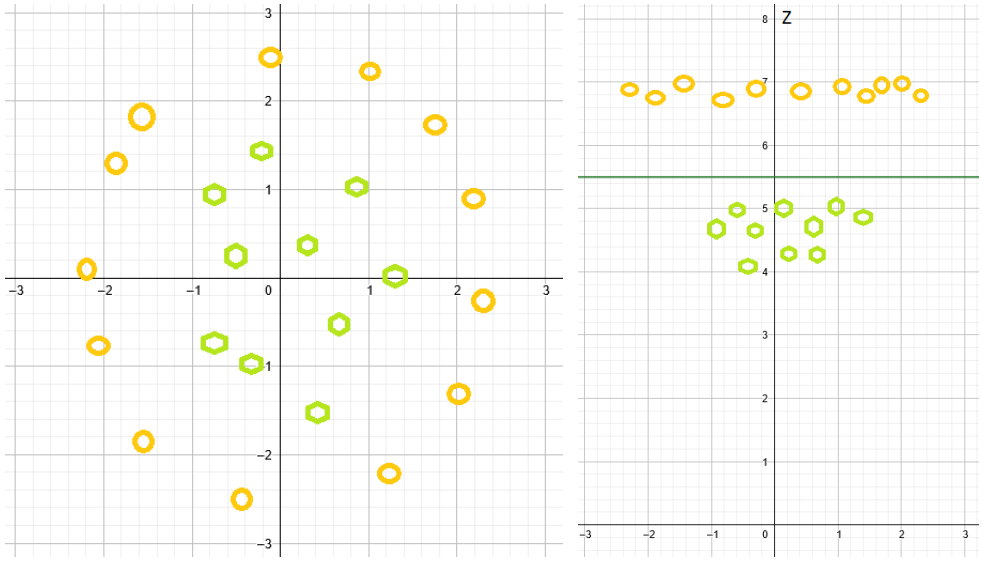
Slika 2. Prava broj 2 razdvaja uzorke na pravi način.

Na desnoj strani slike broj 2 imamo sličnu situaciju. I ovde je intuitivno da prava broj 2 bude izabrana, ali potrebno je ipak pojasniti način na koji je to izvršeno. Kao što je već rečeno ovde tražimo pravu koja je najudaljenija od uzoraka. Udaljenost se određuje tako što se posmatraju samo dva najbliža uzorka i traži se takav položaj prave da duži koje su normalne na nju i koje je povezuju sa uzorcima budu što veće. Takođe, potrebno je da te duži budu iste dužine. Ovim je ispunjen drugi zahtev, a to je da je hiperravan maksimalno udaljena. Ipak, u praksi je redak slučaj da ne dolazi do mešanja podataka, pa prema tome na slici 3 možemo videti da je element jedne „zalutao“ među elemente druge klase. MPV poseduje mehanizam zahvaljujući kome ovakve izuzetke ne razmatra, tako da se prava koja razdvaja dva skupa ponovo nalazi na pravom mestu.



Slika 3. MPV poseduje mehanizam kojim je u mogućnosti da zanemari izuzetke.

Na primeru slike 4 se može videti na koji način možemo iskoristiti MPV da rešimo nelinearni problem. Ovde ponovo imamo dvodimenzionalnu ravan, ali ovoga puta ne možemo razdvojiti uzorke pravom. Da bismo mogli da napravimo podelu potrebno je da dodamo još jednu dimenziju. Koordinate za novu dimenziju računamo po sledećoj formuli: $z = x^2 + y^2$. Na ovaj način dobijamo linearni problem koji možemo rešiti uz pomoć ravni. Dodatnu dimenziju ne moramo ručno dodavati ili definisati, već MPV poseduje mehanizam zvani „trik jezgra“ koji dati posao obavlja za nas. Jezgro je funkcija koja kao ulaz prima podatke određenih dimenzija i dodaje im nove dimenzije da bi omogućila njihovu klasifikaciju.



Slika 4. MPV prostoru u kome se podaci nalaze dodaje nove dimenzije da bi se problem sveo na linearni.

2.4 Piramidalno skaliranje

Da bismo prepoznali objekte različite veličine potrebno je da sliku analiziramo sa klizećim prozorima različitih veličina. Tu nastaje problem zbog toga što MPV ne može da podrži podatke različitih dimenzija. Iako bi bilo moguće promeniti veličinu dela slike koja je obuhvaćena segmentom ili napraviti više MPV-ova koji bi prihvatali podatke različitih veličina, najefikasniji način je da promenimo veličinu kompletne slike i omogućimo da prozor u novom prolazu obuhvati veći deo slike.

Za tako nešto se obično koristi „piramida slika“ (slika 5). „Piramida slika“ je skup slika koje su nastale iterativnim smanjivanjem ili povećavanjem dimenzija iste slike. Korišćenjem ove tehnike nismo u mogućnosti da preciziramo visinu i širinu koju će svaka nova slika u nizu imati, već samo možemo da podesimo faktor za koliko će slika biti umanjena ili uvećana. Razlog za korišćenje ove tehnike je taj što se umanjuje šum koji nastaje usled gubitka informacija prilikom promena dimenzija slike. Postoje dve vrste piramida: Gausova i Laplasova. U ovom radu će biti korišćena i objašnjena Gausova piramida.

Slika 5. „Piramida slika“. [8]

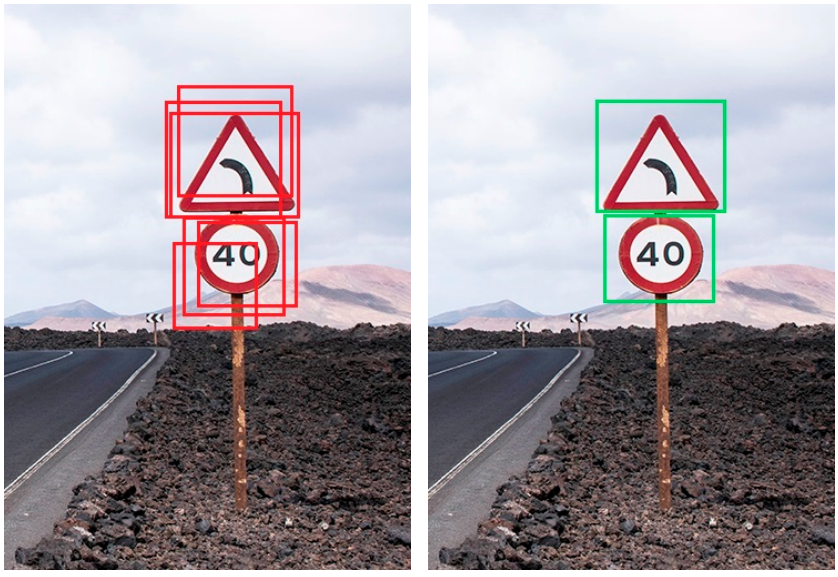
Prvo je potrebno da nad slikom primenimo Gausov filter, koji je zapravo sledeći izraz:

$$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} * \frac{1}{16}$$

Ovim svaki piksel dobija srednju vrednost ostalih piksela koji se nalaze pored njega. Dalje samo preostaje da uklonimo određeni broj redova i kolona u zavisnosti od preciziranog faktora. Ovim putem dobijamo sledeću sliku u nizu. Iteriranjem prethodnih koraka dobijamo celu piramidu. Ovako možemo da smanjimo sliku, ali na koji način bismo mogli da sliku piramidalno povećamo? U tom slučaju potrebno je prvo dodati nove redove i kolone koji su popunjeni nulama. Zatim na tako dobijenu sliku ponovo primenjujemo gore navedeni filter, ali ovog puta pomnožen sa 4. Na taj način vršimo aproksimaciju piksela koji nedostaju (novih redova i kolona koji su popunjeni nulama). Opet, ponavljanjem prethodnih koraka dobijamo kompletnu piramidu.

2.5 Algoritam NMS

Ovaj algoritam kao ulaz prihvata koordinate levih gornjih uglova delova slike koji su markirani kao delovi koji poseduju traženi objekat[3]. Prvi korak je određivanje procenta preklapanja segmenata među sobom. Nakon toga se uzima prvi segment u nizu i posmatra se njegov procenat preklapanja sa drugim segmentima. Ako je taj nivo veći ili jednak u odnosu na parametar, koji je moguće podesiti kao ulaz algoritma, segment sa kojim se vrši poređenje se proglašava redundantnim i izbacuje iz niza. Nakon što prođemo kroz ceo niz uzimamo sledeći element (ako postoji) i vršimo poređenja sa elementima koji se nalaze nakon njega (ne vršimo poređenje sa elementima pre njega jer smo to već uradili).



Slika 6. Rezultati pre i posle primene NMS algoritma. [9]

[3] Ovo je moguće ako su veličine segmenata uvek iste; u suprotnom bi bilo potrebno proslediti i dimenzije ili koordinate desnog donjeg ugla.

Rečeno je da se za upoređivanje uzima prvi segment u nizu, a da se ostali upoređuju sa njim i onda izbacuju iz niza. Ovo znači da način na koji je niz sortiran ima uticaj na to kako će finalni niz izgledati. Kao kriterijum sortiranja možemo koristiti procenat poverenja da se u datom delu nalazi objekat. Ako nemamo takav podatak onda možemo na primer sortirati niz po desnom donjem uglu ili ga uopšte ne sortirati.

3 Klasifikacija

Nakon detekcije saobraćajnog znaka potrebno je odrediti kog tipa je taj znak, odnosno kojoj klasi pripada. Za ovaj zadatak koristićemo konvolucijske neuronske mreže, koje pripadaju skupu algoritama dubokog učenja. Nakon opisa slojeva konvolucijske neuronske mreže, predstavljen je problem preobučavanja i predstavljeni su algoritmi za obučavanje neuronskih mreža.

3.1 Duboko učenje i konvolucijske neuronske mreže

Ovaj tip algoritama je u poslednjoj deceniji privukao veliku pažnju zbog njegove velike primene u kompjuterskom vidu i obradi glasova. Među tim algoritmima se izdvajaju konvolucijske neuronske mreže koje predstavljaju jedan od najboljih algoritama za kompjuterski vid. One su naziv dobile po konvolucijskom sloju. Ovaj sloj sprovodi konvoluciju[4], operaciju koja je široko primenjena u obradi slike. Glavni element ove operacije su konvolucijski filteri koji imaju ulogu da registruju određena svojstva na slici. Upravo su oni ti koji doprinose efikasnosti ovih mreža jer kada bismo jednostavno sliku (koja je zapravo matrica piksela) razvili u vektor ne bismo mogli da ostvarimo preciznost i tačnost koju dobijamo zahvaljujući ovim filterima. Konvolucijska neuronska mreža se može podeliti na više slojeva koji imaju različite funkcije:

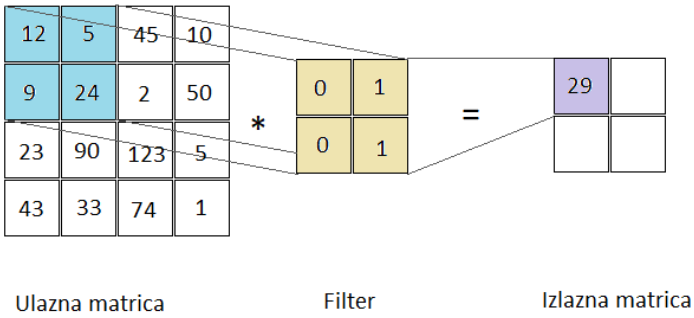
- Konvolucijski sloj
- Sloj sažimanja
- Sloj aktivacione funkcije
- Potpuno povezan sloj

3.2 Konvolucijski sloj

Ovaj sloj sadrži nekoliko filtera. Svaki od tih filtera služi pronalaženju određenog svojstva na slici. Da bismo razumeli na koji način se ovi filteri primenjuju uzećemo za primer matricu veličine 4*4 i filter veličine 2*2 (slika 7). Ideja je slična kao i kod tehnike „klizećeg prozora“; filter pomeramo za određeni broj piksela po vertikali i horizontali dok ne obradimo celu sliku. Ono što je novo je operacija koja se izvršava. Kada filteru dodelimo određeni broj piksela slike koji mu dimenziono odgovaraju

[4] Mnogi eksperti iz matematike bi se pobunili zbog ovakvog naziva jer ovo baš i nije konvolucija (jedna od stvari koje bi bilo potrebno odraditi da bi ovo zaista bila konvolucija je rotiranje konvolucionog filtera). Ipak, s obzirom da je mašinsko učenje veoma mlada disciplina i da se terminologija nije još učvrstila, držaćemo se ovog naziva koji je opšteprihvaćen.

vršimo operaciju skalarnog proizvoda vektora. Novodobijena vrednost postaje element matrice koja je rezultat konvolucije.



Slika 7. Primena filtera u konvolucijskom sloju.

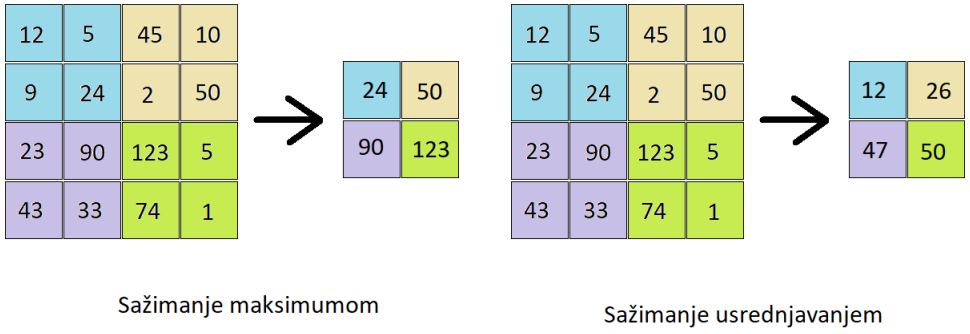
S obzirom da kompleksne neuronske mreže mogu imati i po nekoliko hiljada ovakvih filtera pitanje koje se postavlja je kako da znamo koja svojstva da tražimo na slikama i da li moramo svaki filter pojedinačno da definišemo. Najznačajnija stvar kod konvolucijskog sloja je ta što omogućava da filteri sami nauče koje vrednosti treba da poseduju. Ovo omogućava da se neuronska mreža prilagodi ne samo običnim ivicama koje mogu biti vertikalne ili horizontalne već da pronađe svojstva koja su specifična za dati problem. O algoritmima koji omogućavaju proces učenja će biti više reči u nastavku. U praksi obično imamo više ovakvih slojeva. Kod takve konfiguracije filteri prvih slojeva obično prepoznaju jednostavne šablone kao što su linije i boje, dok će filteri u narednim slojevima prepoznavati kompleksnije strukture kao što su oblici i senke.

Ono što predstavlja problem je što smo nakon primene filtera dobili manju matricu (slika 7). Ovo je ponekad poželjno, ali kada je reč o obradi slike ovo može biti jako loše jer previše brzo gubimo informacije. Da bismo ovo izbegli potrebno je da povećamo veličinu početne slike dodavanjem neutralnih piksela (piksela koji imaju vrednost nula) sa svih strana.

3.3 Sloj sažimanja

U ovom sloju se vršu odabiranje. Reč je o operaciji koja smanjuje veličinu podataka čuvanjem onih od značaja i odbacivanjem ostalih. Time ovaj sloj smanjuje količinu operacija izračunavanja koje je potrebno izvršiti. Takođe, ovim se favorizuju dominantna svojstva koja nisu pod uticajem rotacije ili položaja objekta na slici.

Kao i u konvolucijskom sloju imamo klizeći prozor koji se primenjuje nad matricom koja je izlaz konvolucijskog sloja. Postoje dve najčešće operacije koje se primenjuju, a to su sažimanje maksimumom i sažimanje usrednjavanjem (slika 8). Sažimanje maksimumom bira element u prozoru sa najvećom vrednošću i taj element postaje element nove matrice koja je izlaz ovog sloja. Kod sažimanja usrednjavanjem novi element je srednja vrednost svih elemenata koji su obuhvaćeni prozorom.



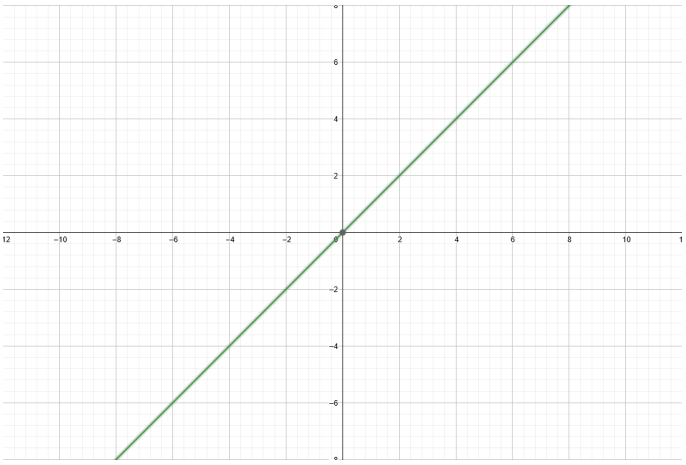
Slika 8. Sažimanje maksimumom i sažimanje usrednjavanjem.

3.4 Sloj aktivacione funkcije

Aktivaciona (još se naziva i prenosna) funkcija ima ogroman uticaj na to kako će teći obučavanje neuronske mreže. Njena uloga je da odredi na koji način će ulazni podaci i težine grana biti mapirani. Postoji više aktivacionih funkcija od kojih su najpoznatije: linearna, sigmoidna, hiperbolički tangens, ReLU (*Rectified Linear Unit*), *Leaky ReLU*… Zajednička karakteristika svih aktivacionih funkcija je njihova diferencijabilnost. Ovo je jako bitna osobina koja omogućava treniranje.

3.4.1 Linearna aktivaciona funkcija

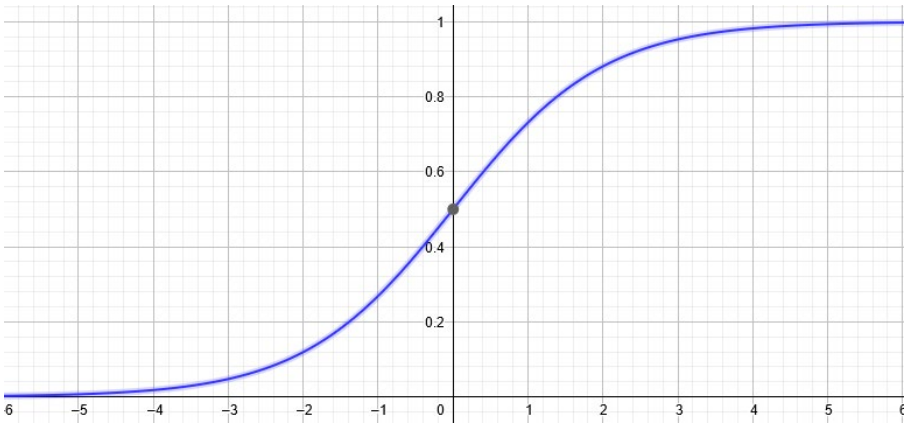
Ova funkcija, kao što joj i samo ime kaže, može se predstaviti pravom linijom (slika 9). Korisna je kada je potrebno rešiti neki linearni problem, ali ne može se iskoristi kod nelinearnih problema. Česta zabluda početnika je da se dodavanjem dodatnih slojeva može prevazići ovaj nedostatak, ali svaka višeslojna neuronska mreža čiji neuroni imaju linearnu funkciju se može svesti na mrežu koja ima jedan sloj linearnih neurona, tako da dodavanjem dodatnih slojeva ne dolazimo do rešenje.



Slika 9. Grafik linearne aktivacione funkcije.

3.4.2 Sigmoidna aktivaciona funkcija

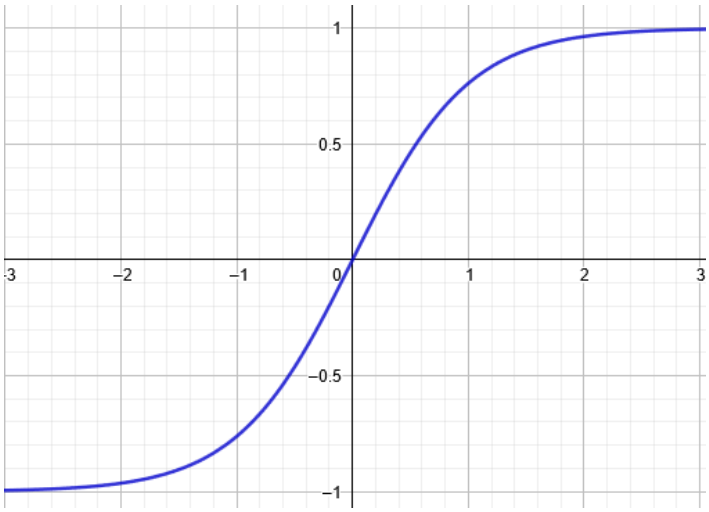
Ovo je nelinearna funkcija čiji je izlaz između 0 i 1 (slika 10). Upravo zbog toga je pogodna za korišćenje na mestima na kojima se traži verovatnoća kao rezultat. Problem kod ovakve funkcije je taj što se kod jako malih ili veliki vrednosti kriva zaravnjuje (dolazi do saturacije). Zbog ovoga izvod postaje jako mali usled čega može doći do stagniranja obučavanja.



Slika 10. Grafik sigmoidne aktivacione funkcije.

3.4.3 Hiperbolički tangens aktivaciona funkcija

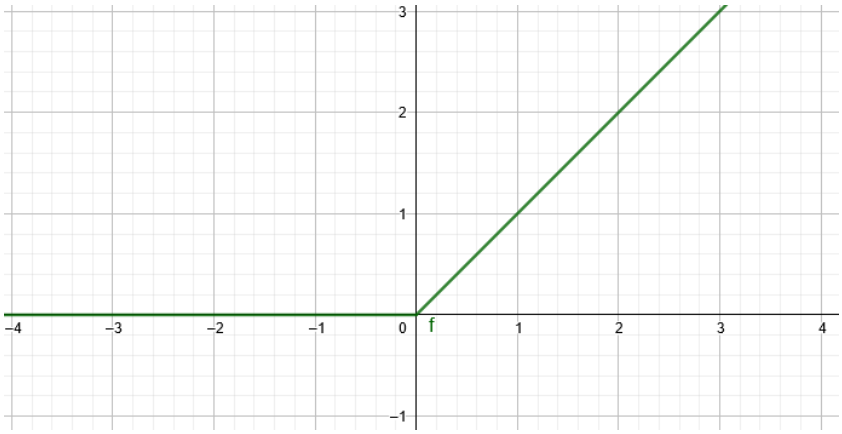
Hiperbolički tangens je veoma sličan sigmoidu. Jedina razlika i prednost je u tome što kod ove funkcije negativne vrednosti bivaju mapirane na negativne vrednosti (slika 11). Ipak, problem stagnacije je i ovde prisutan.



Slika 11. Grafik aktivacione funkcije hiperboličkog tangensa.

3.4.4 *ReLU* aktivaciona funkcija

ReLU (slika 12) delimično rešava problem stagnacije jer je njen izvod kao i sama funkcija monoton. Ovde se pak javlja problem sa negativnim vrednostima koje bivaju „zakucane“ na nulu. Posledica ovoga u praksi je da oko 20% neurona biva „umrtvljeno“ (izvod je 0 tako da algoritam za obučavanje ne može ništa da učini).



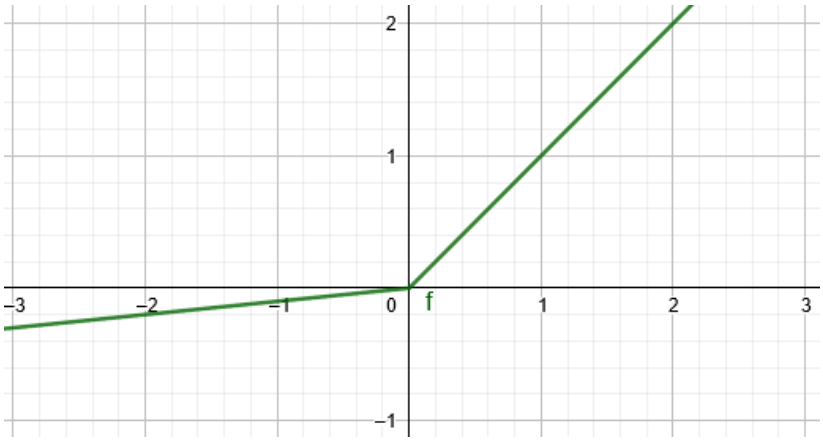
Slika 12. Grafik ReLU aktivacione funkcije.

Ovo je inače najkorišćenija funkcija kod konvolucijskih neuronskih mreža.

3.4.5 *Leaky ReLU* aktivaciona funkcija

Ovde je problem „mrtvih“ neurona rešen s tim što funkcija $f(x) = 0$ postaje $f(x) = a * x$ za $x < 0$, gde a ima malu vrednost kao što je na primer 0.05 (slika 13). Ipak, ne treba uvek koristiti ovu funkciju

jer ono što je navedeno kao negativna karakteristika ReLU-a je ponekada prednost prilikom obučavanja. Odnosno, ponekada i želimo da imamo mrtve neurone jer na taj način se donekle kontroliše preobučavanje, a takođe mrtvi neuroni proizvode i veštački šum.



Slika 13. Grafik Leaky ReLU aktivacione funkcije.

3.5 Potpuno povezan sloj

Ovaj sloj je dobio ime po tome što su svi neuroni međusobno povezani (potpuno povezani). Njegova uloga je da svojstva koja su izdvojena u prethodnim slojevima nauči, kroz podešavanje vrednosti grana između neurona, i da ih klasifikuje. Upravo iz ovog razloga se potpuno povezan sloj uvek postavlja pred sam kraj. Naravno, pošto su na izlazu iz sloja sažimanja svojstva predstavljena matrično potrebno ih je, pre nego što se pošalju prevesti u vektore i te vektore spojiti.

3.6 *Softmax* sloj

Softmax funkcija se koristi kada je potrebno da vrednosti na izlazu iz mreže budu predstavljene u vidu raspodele verovatnoća između određenih klasa. Kao takva ova funkcija se koristi u poslednjem sloju i pruža mnogo bolju preglednost i korisnost rezultata. Ona predstavlja generalizaciju logičke regresije koja može da obradi samo dve klase i koristi sigmoidnu funkciju koja je veoma slična samoj *Softmax* funkciji. Formula ove funkcije je (3.6.1), gde je K broj elemenata u izlaznom vektoru, a z vrednosti su elementi.

$$\sigma(\vec{z}) = \frac{e^{zi}}{\sum_{j=1}^{K} z_j} \tag{3.6.1}$$

3.7 Sloj odbacivanja (engl. *Dropout layer*) i preobučavanje

Preobučavanje je pojava koja se javlja kod mašinskog učenja kada dati model podesi svoje parametre tačno prema skupu podataka za treniranje. Kada se ovo dogodi model nije u mogućnosti da na pravilan način postupa sa podacima koji nisu u skupu za treniranje. Na ovaj način se gubi poenta algoritma za mašinsko učenje jer je kao što je već navedeno glavna prednost ovih algoritama nad algoritmima klasičnog programiranja u tome što mogu da se sa velikom uspešnošću primene nad podacima sa kojima nisu imali prilike da rade.

Do ovog fenomena dolazi kada model koji se obučava počne da „uči“ šumove i nepravilnosti koje nisu od značaja, a nalaze se u skupu podataka za obučavanje. Uzrok ovome može biti mali ili loše izabran skup podataka (ako neki model želimo da osposobimo da razume govor nekog jezika, potrebno je da prikupimo snimke govora više ljudi, a ne samo jedne osobe), predugo treniranje, prevelika kompleksnost modela za date podatke i mnogi drugi.

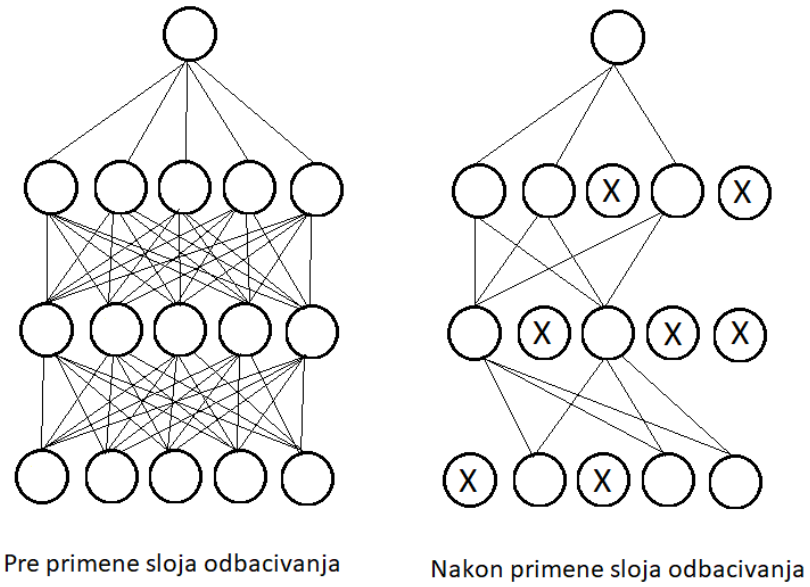
Postoje mnoge preventive koje se koriste da bi se sprečila gore navedena situacija. Neke od najvažnijih su:

- Rano zaustavljanje: Ideja kod ove metode je da se treniranje zaustavi tačno pre nego što degradacija („učenje“ šumova i nepravilnosti) modela počne. Da bi se ovo postiglo potrebno je početni skup podeliti na test i trening skup, a zatim nakon svake iteracije obučavanja proveriti nivo obučenosti nad podskupom podataka iz test i trening skupa (izvršiti evaluaciju modela). Ako u nekoliko uzastopnih iteracija evaluacija nad test skupom daje lošije rezultate od one nad trening skupom obučavanje treba zaustaviti. Naravno tokom ovog procesa obučavanje se vrši isključivo nad trening skupom.
- Korišćenje većeg skupa podataka: Jedno od pitanja koje se postavlja kod preobučavanje je zašto uopšte koristimo kompleksne modele koji se preobučavaju i zašto trening traje tako dugo. Zar ne bi bilo bolje da koristimo jednostavne modele koji neće biti savršeni, ali se zato barem neće preobučavati? S obzirom na to da je većina algoritmi mašinskog učenja inspirisana ljudskom i životinjskom biologijom odgovor ćemo potražiti upravo kod samih ljudi. Kao jedan dobar i pomalo mračan primer nam mogu poslužiti instituti za nezbrinutu decu i to oni koji svoj posao obavljaju nesavesno. Naime, deca u ovakvim ustanovama dobijaju jako lošu negu i brigu (skup podataka nad kojima se „obučavaju“ je dosta limitiran i loš). Usled toga ona nemaju mogućnost da pokažu svoj pravi potencijal i u nekim slučajevima dolazi i do retardacije. Mi, takoreći, ne želimo da naši modeli veštačke inteligencije postanu „retardirani“ i zbog toga je potrebno da im obezbedimo širok i dobar skup podataka da bi oni bili u mogućnosti da pokažu svoj pravi potencijal. Ono što mi

zaista želimo su upravo kompleksni modeli koji obrađuju kompleksne strukture, ali ta kompleksnost postaje mana kada nemamo dovoljno dobre podatke.

- Korišćenje veštački proširenih podataka: Prikupljanje podataka je skup i težak posao. Zbog toga se nekada pribegava veštačkom proširivanju podataka. Ovo se izvodi tako što se od već postojećih podataka kreiraju novi namernim dodavanje šumova i drugih deformiteta. Na primer, od jedne slike sa saobraćajnim znakom je moguće napraviti više njih rotacijom, izoštravanje ili kvarenjem fokusa originalne slike.

Ovaj problem se često javlja kod dubokih neuronskih mreža, koje poseduju veliki broj slojeva sa puno neurona. Da bismo rešili ovaj problem najbolje bi bilo da istreniramo nekoliko nezavisnih modela nad čijim izlazima bismo na kraju izvršili aproksimaciju i na taj način dobili veoma robusne sisteme. Ti modeli bi trebalo da se razlikuju po svojim arhitekturama i bilo bi poželjno da budu trenirani nad različitim podskupovima podataka. Ovako nešto je sasvim izvodljivo i pokazuje se kao dobra praksa kada je reč o malim modelima, ipak kod većih modela ovakav metod postaje veoma skup i računarski neizvodljiv.



Slika 14. Sloj odbacivanja.

Sloj odbacivanja je tehnika koja rešava problem preobučavanja i pruža gore navedenu mogućnost aproksimacije izlaza više različitih mreža. U terminu „sloj odbacivanja“ odbacivanje se odnosi na neurone iz sloja za koji je sloj odbacivanja vezan. Ti neuroni se privremeno gase zajedno sa njihovim

ulaznim i izlaznim granama (slika 14). Parametar koji određuje da li će neki neuron biti ugašen ili ne je predstavljen kao verovatnoća[5]. On obično uzima vrednosti u rasponu od 0,2 do 0,5.

Sloj odbacivanja se može postaviti nakon bilo kog sloja (izuzev poslednjeg, izlaznog sloja). Obično se koristi nakon potpuno povezanog i konvolucijskog sloja, mada se ponekada koristi i nad ulaznim slojem. Tada ovaj sloj doprinosi izgradnji veštačkog šuma.

Nakon što se treniranje ovakve mreže završi težine njenih grana su veće nego što bi trebalo da budu (posledica odbacivanja pojedinih neurona). Da bismo ovo nadomestili potrebno je da težine izlaznih grana svih neurona pomnožimo sa verovatnoćom zadržavanja tog neurona. Na ovaj način smo uspeli da izvršimo aproksimaciju svih mreža koje su nastale tokom obučavanja i da tu aproksimaciju predstavimo kao jednu mrežu. Sloj odbacivanja se naravno koristi samo tokom treniranja, dok se tokom vršenja predviđanja ovaj sloj zanemaruje.

3.8 Obučavanje i algoritam propagacije unazad

Neuronsku mrežu možemo posmatrati kao funkciju koja za ulaz ima podatke nad kojima želimo da je obučimo, težine grana, bajase (engl. *bias*) neurona… Izlaz ove funkcije će biti greška koju je neuronska mreža napravila na svom izlazu i računaće se kao $e = \frac{1}{n} * \sum_{i=1}^{n}(y_i - v_i)$, gde je n broj neurona u poslednjem sloju, y vrednosti na izlazu iz svakog od tih neurona i v je očekivani izlaz. Cilj treninga je da pronađemo za koje vrednosti parametara dobijamo minimum ove funkcije.

Prvo što nam pada na pamet je da izračunamo u kojoj tački je gradijent ove funkcije jednak nuli (ovakvih tačaka može biti više pa bismo među njima morali da pronađemo onu koja daje najmanju vrednost). Ovo je moguće izvesti kod najjednostavnijih modela, ali kada se posmatraju neuronske mreže koje treba da posluže u nekoj realnoj primeni ovaj način je računarski prezahtevan. Zbog toga se skoro uvek koristi metod gradijentalnog spusta.

Ideja kod ove metode je da na početku izaberemo nasumičnu tačku (vrednosti težina grana i bajasa neurona su nasumične). Nakon toga u toj tački izračunamo gradijent. Gradijent možemo posmatrati kao vektor, pa prema tome gradijent će biti usmeren ka tačkama koje zapravo povećavaju vrednost funkcije. Zato je potrebno da vrednosti vektora kojim je gradijent predstavljen pomnožimo sa -1. Zatim početni ulazni vektor treba sabrati sa ovim vektorom nakon čega se dobija nova tačka koja je bliža minimumu.

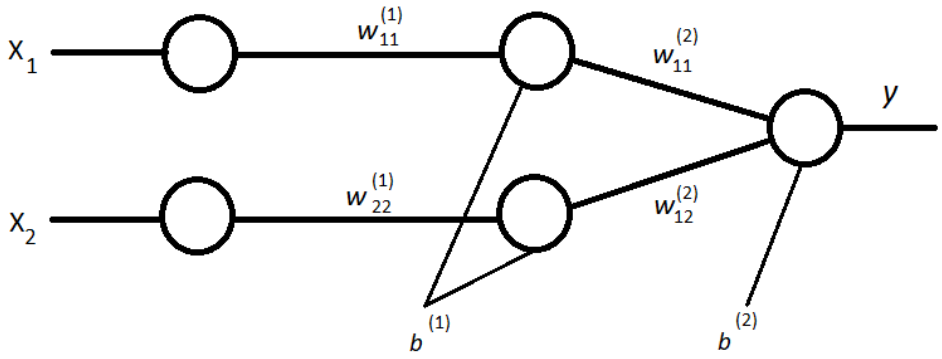
Jedan od jako bitnih parametara u ovom procesu je brzina učenja. To je vrednost koja se množi sa vektorom gradijenta pre nego što isti biva sabrat sa vektorom koji predstavlja trenutnu tačku. Drugim

[5] U nekim implementacijama se ovaj parametar tumači kao verovatnoća da neuron ostane u mreži (ne bude ugašen), dok se u drugim tumači kao verovatnoća da neuron bude odbačen. U *TensorFlow* biblioteci, na primer, imamo drugi slučaj.

rečima, zahvaljujući tom parametru možemo kontrolisati koliko brzo ćemo se „spuštati“ niz našu funkciju.

Algoritam propagacije unazad se zasniva na upravo pomenutoj tehnici. Ime proizilazi iz toga što ovaj algoritam propagira vrednost greške unazad kroz mrežu koristeći pravilo lanca. Da bismo razumeli kako ovaj algoritam funkcioniše poslužićemo se jednostavnim primerom mreže koju možemo videti na slici 15. Recimo da želimo da podesimo težinu grane $w_{11}{}^{(2)}$. Da bismo to izveli potrebno je da izračunamo njen gradijent. Neka je $y = \sigma^{(2)} = f(z_1{}^{(2)})$ izlaz aktivacione funkcije neurona u izlaznom sloju, gde se $z_1{}^{(2)}$ računa kao $z_1{}^{(2)} = \sum_{k=1}^{2}(w_{1k}{}^{(2)} * \sigma_k{}^{(1)}) + b^{(2)}$, gde su $w_{1k}{}^{(2)}$ težine grana, a $b_1{}^{(2)}$ bajas poslednjeg neurona. Sada gradijent ove grane možemo naći po formuli (3.8.1). Isti postupak možemo primeniti i na ostale grane. Na kraju, pronađeni gradijent je potrebno pomnožiti sa negativnom vrednošću brzine učenja i tu vrednost sabrati sa trenutnom vrednošću težine grane $w_{11}{}^{(2)}$.

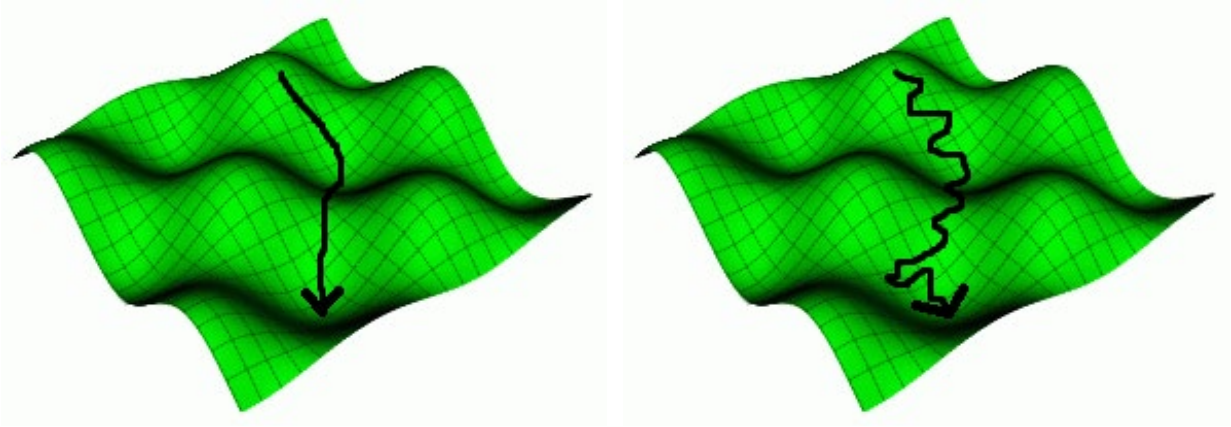
$$\frac{\partial e}{\partial\, w_{11}{}^{(2)}} = \frac{\partial e}{\partial \sigma^{(2)}} * \frac{\partial \sigma^{(2)}}{\partial z_1{}^{(2)}} * \frac{\partial z_1{}^{(2)}}{\partial w_{11}{}^{(2)}} \tag{3.8.1}$$



Slika 15. Neuronska mreža sa jednim skrivenim slojem

Postupak koji je prikazan u prethodnom primeru se ne izvodi nakon propuštanja jednog uzorka, već se taj proces dešava nakon što se propusti kompletan skup uzorka. Ipak, kada imamo milijarde parametara i veliki broj uzoraka tako nešto postaje memorijski neizvodljivo jer je neophodno pratiti ogroman broj vrednosti da bismo na kraju mogli da podesimo parametre. Takođe sam proces može jako dugo trajati jer je potrebno mnogo vremena da se odabere nova tačka. Zbog toga se umesto gradijentalnog spusta koristi stohastički gradijentalni spust. Ono po čemu se ove dve jako slične tehnike razlikuju je to što se kod stohastičkog gradijenta trening skup podeli na mnoštvo podskupova koji obično sadrži ne više od sto uzoraka. Nakon što se prvi podskup propusti kroz mrežu parametri se ažuriraju. Pošto se gradijent

računa nad podskupom „spuštanje“ niz funkciju greške neće biti idealno, ali će biti mnogo brže i računarski efikasnije.



Slika 16. Gradijentalni spust je dosta precizniji i ujednačeniji od stohastičkog gradijentalnog spusta, ali je drugi mnogo brzi i računarski efikasniji. [10]

4 Konkretni sistem

U nastavku će biti opisana implementacija sistema za prepoznavanje saobraćajnih znakova korišćenjem svih prethodno objašnjenih metoda i tehnika. U delu za detekciju će biti korišćeni HOG deskriptori i MPV, dok će za klasifikaciju znakova biti iskorišćena konvolucijska neuronska mreža. Cilj sistema je da u realnom vremenu prepoznaje znakove sa snimka.

4.1 Obučavanje komponenti sistema

Obučavanje sistema je urađeno u programskom jeziku *Python*. Razlog za to je taj što je ovaj jezik netipiziran što pruža mogućnost jako jednostavnog upravljanja podacima. Takođe ovaj jezik ima veliku podršku u različitim bibliotekama koje su od značaja.

Uzorci koji su iskorišćeni za treniranje su iz skupa zvanog *The German Traffic Sign Benchmark*. U tom skupu postoje dva tipa podataka. Prvi je namenjen za obučavanje sistema za detekciju znakova. On se sastoji od 900 slika koje predstavljaju scene sa puteva. Drugi tip podataka su slike samih saobraćajnih znakova koje su sa svih strana oivičene marginama i namenjene su za klasifikaciju. Oivičenost marginama nam daje mogućnost da problemu pristupimo iz ugla detektovanja oblika i ivica na slici. Upravo iz tog razloga ćemo se u ovoj implementaciji poslužiti samo drugim tipom podataka, dok će nam prvi služiti za testiranje.

4.1.1 Treniranje MPV-a

Na početku je neophodno izdvojiti sve slike saobraćajnih znakova (pozitivne uzorke) i pomoću HOG deskriptora izračunati njihova svojstva. To je izvršeno u sledećoj funkciji:

```
def calcHogTrafficSigns(rootpath):
    hog = cv2.HOGDescriptor((64, 64), (16, 16), (8, 8), (8, 8), 9)
    features = []

    for c in range(0,43):
        prefix = rootpath + '/' + format(c, '05d') + '/'
        gtFile = open(prefix + 'GT-' + format(c, '05d') + '.csv')
        gtReader = csv.reader(gtFile, delimiter=';')
        next(gtReader, None)

        for row in gtReader:
            resized_image = cv2.resize(cv2.imread(prefix + row[0]), (64, 64))
            fd = hog.compute(resized_image)
            features.append(np.ravel(fd).tolist())
        gtFile.close()

    return features
```

Ovde posebnu pažnju treba obratiti na funkciju *HOGDescriptor* iz biblioteke *OpenCV*[6]. Ona predstavlja konstruktor HOG deskriptora. Prvi parametar predstavlja veličinu slike koja treba da se obradi. Zatim sledi navođenje dimenzija bloka za normalizaciju. Ovde treba biti oprezan jer se ovde radi o broju piksela koje će jedan blok obuhvatiti, a ne o broju ćelija kako je to u nekim drugim implementacijama (jedna takva je *sklearn*-ova verzija ove funkcije). Nakon toga slede parametri za pomeraj bloka za normalizaciju i dimenzija ćelija. Dok poslednja vrednost predstavlja broj klasa histograma.

Slike znakova su date u različitim dimenzijama, pa im je nakon učitavanja neophodno promeniti dimenzije na 64*64 piksela, odnosno veličina slike se mora poklapati sa onom koju HOG deskriptor očekuje. To se radi pomoću funkcije *resize* koja kao prvi argument prihvata samu sliku, dok drugi argument predstavlja željene dimenzije. Nakon što je slika spremna za obradu treba je propustiti kroz metodu *compute* koja vraća vektor koji predstavlja HOG svojstva date slike. Konačna lista svojstava svih slika se šalje kao povratna vrednost i čuva u *CSV* (engl. *Comma-separated values*) formatu.

Za negativne uzorke se mogu uzeti bilo koje slike koje ne sadrže saobraćajni znak. Za treniranje ovog sistema su uzeti skupovi podataka koji sadrže slike sa biljkama, ljudima, infrastrukturnim objektima, nebom, oblacima… Ti podaci se obrađuju na identičan način kao i prethodno opisani pozitivni uzorci.

Ipak, ovo nije kraj obrade podataka. Sada je potrebno te podatke učitati i strukturno ih pripremiti za MPV. Prvi deo koda koji vrši ovo pripremanje je dat u nastavku:

```
def prepareTestAndTrain(sign_features, non_sign_features):
    features = np.concatenate((sign_features, non_sign_features), axis=0)
    labels = np.append(np.ones(len(sign_features)),\
                np.zeros(len(non_sign_features)))\
                    .reshape(len(sign_features) + len(non_sign_features), 1)

    data = np.hstack((features, labels))
    np.random.shuffle(data)

    percentage = 80
    partition = int(len(data) * percentage / 100)

    f_train, f_test = data[:partition, :-1], data[partition:, :-1]
    l_train, l_test = data[:partition, -1].ravel(),\
                      data[partition:, -1].ravel()
```

Ova funkcija na početku vrši označavanje pozitivnih uzoraka sa 1 i negativnih sa 0 praveći novi vektor zvani labels. Nakon toga je potrebno označene uzorke nasumično pomešati, tako da se nule i jedinice

[6] *OpenCV* je biblioteka koja je namenjena onima koji se bave računarskim vidom. Pruža veliki broj funkcija za manipulaciju slikom i videom, a takođe poseduje i gotove programe za neke od najpoznatijih problema u oblasti. Jedan od primera za tako nešto bi bio skup funkcija i klasa za prepoznavanje pešaka na slici koji se zasniva upravo na HOG-u i MPV-u. Ova biblioteka ima podršku za više jezika među kojima su nama od značaja *C++* i *Python*.

smenjuju. Mešanje redosleda pojavljivanja podataka u vektoru je jako bitan korak. Ako bismo ispustili da ovo uradimo onda bismo u trening skupu imali mnogo veći broj negativnih (ili pozitivnih zavisno od početnog rasporeda) uzoraka, dok bi u test skupu imali mnogo veći broj pozitivnih uzoraka. Ovakav disbalans bi doveo do toga da naš MPV ne bude obučen na pravi način, a takođe rezultati evaluacije nad test skupom ne bi bili precizni. Nakon mešanja vrši se podela na trening i test skupove, podaci se zatim prebacuju u njihove 32-bitne vrednosti i pripremljeni podaci se vraćaju:

```
    f_train = f_train.astype('float32')
    f_test = f_test.astype('float32')
    l_train = l_train.astype('int32')
    l_test = l_test.astype('int32')

    return (f_train, f_test, l_train, l_test)
```

Sada je neophodno kreirati sam objekat MPV-a. Funkcija jezgra koja je odabrana je linearna funkcija, koja je u poređenju sa nekim drugim funkcijama dosta jednostavna. Ipak, ta jednostavnost je upravo ono što nam je i neophodno. Zahvaljujući njoj moguće je na jako brz način obraditi jedan uzorak što nam je od velikog interesa jer bi ovaj sistem trebalo da radi u realnom vremenu. Takođe, nešto komplikovanije funkcije se u ovom slučaju nisu pokazale kao superiornije. Sledi kod za inicijalizaciju MPV-a:

```
def configLinearSvm(max_iter, precision, c):
    svm = cv2.ml.SVM_create()
    svm.setType(cv2.ml.SVM_C_SVC)
    svm.setC(c)
    svm.setKernel(cv2.ml.SVM_LINEAR)
    svm.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, max_iter, precision))

    return svm
```

Metoda *setType* postavlja tip MPV-a. Ovde je izabrano da to bude višeklasni (broj klasa je jednak ili veći od 2) C – MPV. Ovaj tip MPV-a koristi parametar *c* kojim balansira između veličine margine i ispravno klasifikovanih uzoraka. Za velike vrednosti ovog parametra margina će biti mala, ali će zato veći broj uzoraka u trening skupu biti ispravno klasifikovan. Male vrednosti će dovesti do toga da se optimizacija skoncentriše na traženje što veće margine po cenu pogrešno klasifikovanih podataka. Metoda *setKernel* postavlja funkciju jezgra, dok metoda *setTermCriteria* postavlja kriterijum za prekid obučavanja. Ovde je neophodno proslediti i maksimalan broj iteracija i maksimalnu preciznost, iako je kao kriterijum za prekid postavljen samo maksimalan broj iteracija.

Na kraju je još prikazan deo koda koji sve ovo uklapa u jednu celinu:

```
def trainSvm(svm):
    (f_train, f_test, l_train, l_test) = \
        prepareTestAndTrain(sign_features, non_sign_features)
    svm.train(f_train, cv2.ml.ROW_SAMPLE, l_train)

    retval, l_pred = svm.predict(f_test)

    print(classification_report(l_test, l_pred))

svm = configLinearSvm(100000, 1e-6, 0)
trainSvm(svm)
```

Metoda *predict* vrši evaluaciju nad test skupom. Nakon toga se predikcija upoređuje sa stvarnim vrednostima i ispisuje se izveštaj koji sadrži podatke o preciznosti i tačnosti sistema. Ovakav model je i dalje daleko od gotovog. Ako bismo ga testirali na nekoj slici dobili bismo ne tako dobre rezultate. Razlog za to leži u negativnim uzorcima. Naime, iako su negativi koje smo spremili daleko brojniji od pozitiva oni nisu dovoljni da bi sistem mogao da na dobra način i sa velikim procentom uspešnosti raspoznaje znakove. Zbog toga je neophodno da kroz MPV propustimo dodatne slike i da izdvojimo sve one koje MPV označi kao pozitivne. Iz tih podataka treba odabrati one koji su zapravo negativni i njih uvrstiti u skup negativa. Na ovaj način MPV je u mogućnosti da nam naznači sa kakvim strukturama na slici ima problema, a mi zatim možemo da izdvojimo te strukture i da ih dodatno uvrstimo u skup za treniranje. Da bismo prikupili što relevantnije podatke bilo bi dobro da ovu metodu primenimo nad podacima sa kojima očekujemo da se naš sistem susretne. Za to možemo koristiti slike ili sam video snimak koji će naš sistem analizirati.

U zavisnosti od toga da li se koristi samo slika ili video snimak programski kod koji vrši automatizaciju ovog procesa može biti različit, ali u oba slučaja je neophodno iskoristiti funkciju čija implemtacija je data na sledećoj strani. Ona prihvata sliku koju je potrebno obraditi, MPV, HOG deskriptor, listu svojstava u koje je potrebno sačuvati rezultate i da li želimo da sami delovi slike za koje MPV proceni da sadrže znakove budu sačuvani. Preostala dva parametra možemo iskoristiti u slučaju kada funkciju koristimo u okviru petlje koja obrađuje video snimak da bismo na pravilan način numerisali slike. Ovde možemo zapaziti da metoda *compute* ima još jedan dodatni parametar koji određuje za koliko će se segment koji obrađuje određeni deo slike pomeriti. Odnosno, mi kao parametar ovoj funkciji prosleđujemo sliku dimenzija 2048*1024 piksela. Kao što je već rečeno, da bismo ovu sliku mogli da obradimo potrebno je da koristimo tehniku „klizećeg prozora“. Prava moć metode *compute* se upravo ovde iskazuje jer ona ima ugrađenu tehniku „klizećeg prozora“ za koju lako možemo da definišemo intenzitet pomeranja segmenta. Sada kao povratnu vrednost te metode dobijamo vektor koji je izgrađen nadovezivanjem vektora koji su bili rezultat obrade svakog pojedinačnog segmenta. Zbog toga je potrebno kroz taj vektor iterirati sa korakom od 1764 elementa. Taj broj predstavlja veličinu jednog

vektora koji je nastao kao rezultat izračunavanja svojstava nad slikom od 64*64 sa do sada navedenim parametrima. Ako MPV naznači da neki od segmenata sadrži saobraćajni znak onda vektor nad kojim je izvršena procena treba sačuvati. Dodatno se računaju i koordinate dela slike od značaja da bi kasnije taj deo bio uokviren. Takođe, ako je parametar *save* postavljen na vrednost *True* deo slike se izdvaja i pamti kao posebna slika radi kasnije analize od strane čoveka i zaključivanja da li je došlo do tačne predikcije i da li svojstva koja su izdvojena treba uvrstiti u skup za treniranje.

```
def findTrafficSigns(image, svm, hog,\
                     features, save = False, round_num = 0, it = 0):
    fd = hog.compute(image, winStride=(16, 16))

    matches = []
    for i in range(0, len(fd), 1764):
        arr = fd[i:i + 1764]
        retval, pred = svm.predict(arr.T)
        if pred[0] == 1:
            x = int(((i / 1764) % 125) * 16)
            y = int(((i / 1764) // 125) * 16)
            matches.append((x, y))

            if save:
                window = image[y:y + 64, x:x + 64]
                cv2.imwrite("dataset/autogen/v_iter"\
                    + str(round_num) + "_" + str(it) + ".jpg", window)

            features.append(np.ravel(arr).tolist())

            it = it + 1


    for m in matches:
        cv2.rectangle(image, (m[0], m[1]),\
                      (m[0] + 64, m[1] + 64), (255, 0, 0), 2)

    return image, it
```

Nakon što su dodatni negativi izdvojeni potrebno je ponovo pokrenuti treniranje MPV-a. Ovaj niz koraka treba ponavljati sve dok se ne dođe do željenih performansi, odnosno, sve dok su rezultati nakon treniranja bolji od prethodnih. Tokom ovog procesa se nikako ne smeju brisati niti zamenjivati uzorci, već se samo vrši dodavanje.

4.1.2 Treniranje konvolucijske neuronske mreže

The German Traffic Sign Benchmark sadrži 43 klase saobraćajnih znakova. Sve slike su označene i skup dolazi sa gotovim kodom za čitanje slika i oznaka iz fajlova. Time je posao obrade podataka umnogome olakšan, ali je potrebno osim učitavanja podatke dodatno pripremiti.

Deo koda koji to obavlja je sledeći:

```
trainData, trainLabels = readTrainCNN(trainFolder)
testData, testLabels = readTestCNN(testFolder)

valData = testData[int(len(testData) * 0.6):len(testData)]
valLabels = testLabels[int(len(testLabels) * 0.6):len(testLabels)]

testData = testData[0:int(len(testData) * 0.6)]
testLabels = testLabels[0:int(len(testLabels) * 0.6)]

trainData = np.array(trainData)
testData = np.array(testData)

trainData = trainData.astype('float32') / 255.0
testData = testData.astype('float32') / 255.0

trainLabels = np_utils.to_categorical(trainLabels, 43)
testLabels = np_utils.to_categorical(testLabels, 43)
```

Kao i pre na početku imamo čitanje trening i test podataka. Ono što se ovde javlja kao novi pojam je validacioni skup podataka. Kao što je već napomenuto, prilikom treniranja nekog modela može doći do preobučavanja, da bismo preobučavanje mogli da prepoznamo i zaustavimo potrebno je da koristimo test skup koji sadrži podatke nad kojima model nije obučavan. Evaluacijom modela nad tim podacima dobijamo pravu sliku performansi. Ali ponekada može doći do preobučavanja modela i nad samim test skupom. Da bismo mogli da sa sigurnošću kažemo da će model raditi sa određenom tačnošću neophodno je da izvršimo testiranje nad validacionim skupom. Kao takav, validacioni skup ne bi trebalo koristi tokom bilo koje faze treninga, već samo na kraju kada želimo da se uverimo da su izveštaji ispravni i da damo konačni sud o samom modelu.

Prema tome test podaci se dele na validacioni i test skup. Podaci zatim treba da budu ubačeni u nampaj (engl. *numpy*) nizove koji omogućavaju njihovu laku i brzu obradu. Nakon toga svaka vrednost u tim nizovima koja predstavlja jačinu boje nekog piksela treba da bude pretvorena u tip *float32* i skalirana na vrednosti između 0 i 1. Ovakva normalizacija ulaznih podataka je veoma bitna jer se s njom postiže veća brzina obučavanja i veća brzina konvergencije ka minimumu funkcije greške. Takođe, izostavljanjem ovog koraka i slanjem podataka koji su međusobno vrednosno veoma različiti dovodi do toga da težine grana postaju velike, a takođe i problem postaje kompleksniji. Usled ovoga model postaje nestabilan i gubi robusnost i opštost.

Pošto će konvolucijska neuronska mreža na kraju izbaciti vektor od 43 broja (posmatramo 43 različite klase) potrebno je da struktura oznaka bude u istom formatu. Ono što dati skup podataka pruža su oznake koje predstavljaju skalarnu vrednost neke klase (na primer, za drugu klasu imamo upisan broj 2). Da bismo ove vrednosti pretvorili u vektore čiji su svi elementi 0 osim onog elementa čiji se redni broj poklapa sa datim skalarom i čija je vrednost 1 možemo iskoristi funkciju *to_categorical*. Za primer

možemo uzeti oznaku sa vrednošću 1. Ova oznaka će nakon primene prethodno navedene funkcije biti pretvorena u sledeći vektor: [0 1 0 0 … 0 0].

Struktura neuronske mreže je predstavljena klasom *LeNet5CF*. Kao što joj samo ime kaže ova mreža se bazira na *LeNet5* arhitekturi, koja je jedna od prvih konvolucijskih neuronskih mreža. Napravljena je sa ciljem prepoznavanja štampanih i pisanih karaktera. Najveća razlika između originalnog *LeNet5*-a i ove implementacije je ta što umesto trećeg konvolucijskog sloja, koji je trebalo da usledi nakon poravnanja izlaza drugog sloja sažimanja, imamo potpuno povezan sloj. Konstruktor klase *LeNet5CF* je sledeći:

```
def __init__(self):
        self.model = Sequential()
        self.model.add(Conv2D(24, 5, input_shape=(64, 64, 3),\
                        padding='same', activation='relu'))
        self.model.add(Dropout(0.65))
        self.model.add(MaxPool2D(pool_size=(2, 2)))
        self.model.add(Conv2D(64, 5, padding='same', activation='relu'))
        self.model.add(Dropout(0.65))
        self.model.add(MaxPool2D(pool_size=(2, 2)))
        self.model.add(Flatten())
        self.model.add(Dense(480, activation='relu'))
        self.model.add(Dropout(0.67))
        self.model.add(Dense(84, activation='relu'))
        self.model.add(Dropout(0.67))
        self.model.add(Dense(43, activation='softmax'))
```

Konstruktor *Sequential* gradi model koji će sadržati različite slojeva mreže jedne za drugim. Tako postavljeni ti slojevi će podsećati na strukturu steka. Dodavanje slojeva u sekvencijalni model funkcioniše nalik dodavanju elemenata u listu. Prvi sloj je konvolucijski. Ovaj sloj ima 24 filtera dimenzija 5*5 piksela, a pomeraj je 1 piksel. Da ovde ne bi došlo do sažimanja i gubitka podataka potrebno je parametar *padding* postaviti na vrednost *same*. Na ovaj način će ulaz biti proširen nulama, čime će dimenzije biti očuvane. Parametar *activation* precizira da li će vrednosti na izlazu biti obrađene od strane aktivacione funkcije ili ne i koja će to funkcija biti. Ovde je izabrano da to bude *ReLU*. Još je neophodno naznačiti kojih dimenzija će biti ulazni podaci postavljanjem vrednosti parametra *input_shape*. U ovom slučaju su to slike dimenzija 64*64 sa 3 kanala.

Zatim sledi sloj odbacivanja. Argument koji mu se prosleđuje predstavlja verovatnoću da neuroni u sloju za koji je vezan budu ugašeni. Ovde je prosleđena vrednost od 0,65 što je neuobičajeno veliko, ali u datom problemu se pokazalo kao efikasno.

Naredni sloj je sloj sažimanja. Sažimanje maksimumom se vrši nad kvadratima sa stranicama dužine 2 piksela sa pomerajem od takođe 2 piksela. Ovakvo sažimanje će na izlazu od prethodno pomenutih 64*64 dovesti do dimenzija 32*32.

Onda se ista tri sloja ponavljaju sa razlikom u tome što konvolucijski sloj sada ima 64 filtera. Sledi pretvaranje višedimenzionih podataka u jednodimenzioni vektor pomoću sloja *Flatten*. Ovaj sloj nije bio neophodan pošto bi naredni potpuno povezani sloj automatski poravnao podatke.

Prvi potpuno povezani sloj sadrži 480 neurona na čijim izlazima je *ReLU* aktivaciona funkcija. Drugi potpuno povezani sloj sadrži 84 neurona. Oba sloja imaju sloj odbacivanja sa verovatnoćom uklanjanja neurona od 0,67. Poslednji sloj je takođe potpuno povezan, ali je njegova aktivaciona funkcija *softmax.* Ovim je omogućeno da na izlazu imamo vektor raspodele verovatnoća po klasama.

Nakon što instanciramo model potrebno je izvršiti njegovo kompajliranje. To se izvodi na sledeći način:

```
def compile(self):
        self.model.compile(optimizer=SGD(lr=0.0005),\
                           loss="categorical_crossentropy",\
                           metrics=["accuracy"])
```

Optimizacija je podešena na stohastički gradijentalni spust sa brzinom učenja od $5 * 10^{-4}$. Funkcija greške je *categorical crossentropy* čija je formula $e = \sum_{i=1}^{n} y_i * \log x_i$, gde je y skalarna vrednost u izlaznom modelu, a x je očekivana vrednost i n je veličina izlaznog vektora. Od parametara koje želimo da pratimo tokom obučavanja i evaluacije, pored vrednosti funkcije greške, navodimo i tačnost modela.

Obučavanje ovog modela se vrši pozivom sledeće metode:

```
def train(self, trainData, trainLabels, epochs_num):
        return self.model.fit(trainData, trainLabels,\
                              batch_size=32, epochs=epochs_num, verbose=1)
```

Ovoj metodi se prosleđuju podaci i oznake za treniranje, kao i broj epoha. Kod funkcije *fit* se još precizira i veličina jednog podskupa koji će biti analiziran. Povratna vrednost je objekat istorije obučavanja koji sadrži brojne informacije o stanjima u kojima se model našao tokom treninga. Nakon treninga obavlja se evaluacija modela sledećom metodom:

```
def evaluate(self, testData, testLabels):
      (loss, accuracy) = self.model.evaluate(testData,\
                         testLabels, batch_size=32, verbose=1)
      print("accuracy: {:.2f}%".format(accuracy * 100))
```

Argumenti koji se prosleđuju su identični onima koji se prosleđuju metodi za treniranje, samo što ovde na kraju imamo dodatni ispis tačnosti na kraju.

4.2 Implementacija sistema

Implementacija je napisana u *C++* da bi izvršavanje sistema bilo što brže. Biblioteke *OpenCV* i *TensorFlow* pružaju podršku za ovaj jezik pa je u potpunosti moguće modele koji su trenirani u *Python*-u sačuvati i učitati pomoću *C++*-a. Čuvanje i učitavanje vrednosti se vrši sledećim kodom:

```
svm.save(filename)
cnn.model.save(foldername)
```

```
cv::Ptr<cv::ml::SVM> svm = cv::ml::SVM::load(SVM_MODEL_FILENAME);
cppflow::model cnn(CNN_FOLDERNAME);
```

Za učitavanje konvolucijske neuronske mreže se koristi biblioteka *CppFlow* koja predstavlja omotač biblioteke *TensorFlow C API.*

Nakon što su modeli spremni za prihvatanje podataka sledi instanciranje HOG deskriptora i učitavanje video snimka:

```
cv::HOGDescriptor hog(cv::Size(64, 64),
                  cv::Size(16, 16), cv::Size(8, 8), cv::Size(8, 8), 9);
cv::VideoCapture cap(VIDEO_FILENAME);

if (!cap.isOpened()) {
    std::cout << "Can't open the video." << std::endl;
}

cv::Mat frame;
cv::Mat resized;
```

Video snimak je predstavljen klasom *VideoCapture* koja pored mogućnosti da video snimak bude učitan iz datoteke pruža mogućnost da izvor video snimka bude kamera. Frejmovi su predstavljeni klasom *Mat* i ovde su potrebni sam frejm i njegova kopija sa podešenim dimenzijama.

Čitanje frejma iz video snimka je trivijalno i vrši se pomoću operatora >> u *while* petlji, kao što je prikazano u narednom delu koda:

```
cap >> frame;
while (!frame.empty()) {

    cv::resize(frame, resized, cv::Size(2048, 1024));

    find_traffic_signs(resized, svm, hog);

    cv::imshow("Video", resized);
    int key = cv::waitKey(1);
    if (key == 'q') {
        break;
    }
    cap >> frame;
}
```

Nakon što se dimenzije slike promene tako da budu prilagođene ostalim komponentama, slika se prosleđuje na analizu funkciji *find_traffic_signs*. Nakon što se slika obradi i na njoj označe znakovi sledi prikazivanje te slike i provera da li korisnik želi da završi sa radom.

Funkcija *find_traffic_signs* ima istu ulogu kao i njena istoimena funkcija u delu za treniranje (videti 4.1.1). Ipak, ove dve funkcije nisu potpuno identične. Na početku se računaju svojstva slike i vrše predikcije:

```
std::vector<float> fd;
hog.compute(image, fd, cv::Size(16, 16));

std::vector<std::pair<int, int>> matches;

for (int i = 0; i < fd.size(); i += 1764) {

    float result =
            svm->predict(std::vector<float>(fd.begin() + i,
                                                fd.begin() + i + 1764));

    if (result == 1) {
        int x = ((i / 1764) % 125) * 16;
        int y = ((i / 1764) / 125) * 16;
        matches.push_back(std::make_pair(x, y));
    }
}
```

Koordinate levog gornjeg ugla dela slike koji sadrži saobraćajni znak se čuva u promenljivoj *matches* koja je tipa *vector*. Nakon ove obrade se u pomenutom nizu mogu naći segmenti slike koji ukazuju na isti objekat (videti 2.1 i 2.5). Zbog toga se ovaj niz kao argument prosleđuje funkciji *non_maximum_suppression* koja predstavlja implementaciju NMS algoritma:

```
auto boxes = non_maximum_suppression(matches, 0.5);
```

Nakon toga potrebno je u petlji svaki od preostalih segmenata propustiti kroz model neuronske mreže i izvršiti predikciju:

```
cv::Mat segment = image(cv::Range(box.second, box.second + 64),
                            cv::Range(box.first, box.first + 64));
segment.convertTo(segment, CV_32F, 1.0 / 255.0);
cv::Mat flat = segment.reshape(1, segment.total() * 3);

std::vector<float> image_data(64 * 64 * 3);
image_data = segment.isContinuous() ? flat : flat.clone();

cppflow::tensor input(image_data, { 1, 64, 64, 3 });
auto output = model(input);
int signClass = cppflow::arg_max(output, 1)
```

Na kraju se na osnovu rezultata obrade informacije iscrtavaju na slici. Mesto na kome je znak detektovan se oivičava kvadratom, dok se broj klase ispisuje. Funkcija *non_maximum_suppression* na početku inicijalizuje neophodne strukture i sortira koordinate svih segmenata prema *y* koordinati donjeg desnog ugla tog segmenta:

```
std::vector<std::pair<int, int>> boxes;
const int matches_len = matches.size();
bool* suppressed = new bool[matches_len]();
std::sort(matches.begin(), matches.end(), comp_coord);
```

Poslednji argument funkcije *sort* je funkcija *comp_coord* na osnovu koje se vrši sortiranje koordinata. Njena implementacija je jako trivijalna pa ona ovde neće biti prikazana. *Suppressed* je niz čiji su elementi postavljeni na *true* ako je segment uvršćen u finalni niz (*boxes*) ili ako je izbačen iz niza, u protivnom njegovi elementi imaju vrednost *false*. Zatim imamo deo koda koji vrši raspoređivanje i odbacivanje:

```
for (int i = 0; i < matches_len; ++i) {
    if (!suppressed[i]) {
        suppressed[i] = true;
        boxes.push_back(matches[i]);

        for (int j = i + 1; j < matches_len; ++j) {
            if (!suppressed[j]
                   && calc_overlap_area(matches[i], matches[j])
                         >= threshold) {
                suppressed[j] = true;
            }
        }
    }
}
```

Algoritam NMS je već detaljno opisan (videti 2.5) pa ovde još ostaje da se spomene uloga funkcije *calc_overlap_area*. Ona kao argumente prihvata koordinate dva segmenta i vrši izračunavanje procenta preklapanja ta dva segmenta. Procenat preklapanja se računa tako što se izračuna površina preklapanja dva segmenta nakon čega se ta vrednost podeli sa površinom jednog segmenta koja je 64*64 piksela. Procenat preklapanja se izražava kao vrednosti između 0 i 1, pa prema tome funkcija *calc_overlap_area* vraća realan broj. Načini na koje se ova funkcija može implementirati su mnogobrojni i svi su relativno trivijalni, pa ovde njena implementacija neće biti prikazana.

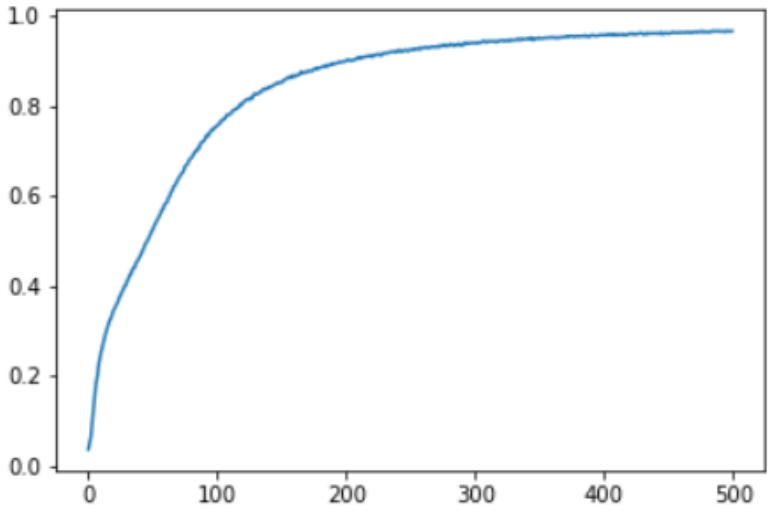
5 Performanse sistema i diskusija

U ovom poglavlju će biti napravljen osvrt na rezultate prethodno opisanog sistema. Nakon toga će sistem biti upoređen sa druga dva sistema. Prvi od ta dva je skoro identičan ovde prikazanom sistemu, dok se u drugom koriste novije tehnike koje daju dosta bolje rezultate.

5.1 Performanse sistema

Sistem koji je ovde opisan je postigao tačnost od 96,66% u delu za detekciju i 95,93% u delu za klasifikaciju. Evaluacija tačnosti u delu za detekciju je izvršena nad 300 slika iz skupa *The German Traffic Sign Benchmark* posebno namenjenih za testiranje detekcije. U delu za klasifikaciju evaluacija je odrađena nad validacionim skupom.

Tabela 1 prikazuje kako je proces podešavanja parametara tekao. Na početku je problem predstavljalo jako sporo obučavanje, pa se zbog toga nije moglo doći do konvergencije. U narednim koracima je brzina učenja povećana, a veličina beča (engl. *batch*) smanjena. To je dovelo do preobučavanja. Taj problem je rešen povećavanjem odbacivanja neurona. Zapaziti da poslednji model za test skup prijavljuje tačnost od 95,86% dok je kao konačna tačnost za klasifikaciju prijavljeno 95,93%. Ta razlika proizilazi iz toga što je za konačnu tačnost korišćen validacioni skup. Na slici 17 je dat grafik toka obučavanja finalnog modela neuronske mreže. Može se primeti da mreža konvergira nakon 500 epoha obučavanja.



Slika 17. Tok obučavanja neuronske mreže. Vertikalna skala je tačnost, a na horizontalnoj osi je prikazan broj epoha.

Tabela 1. Proces podešavanja parametara.

Redni br.	Beč	Brzina učenja	Verovatnoća odbacivanja za kovolucijski	Verovatnoća odbacivanja za potpuno povezani	Br. epoha	Trening tačnost	Test tačnost
1)	128	0,0001	/	0,5 (samo prvi)	500	95.32%	90.01%
2)	64	0,0001	0,4 (oba)	0,4 (samo prvi)	500	97,98%	90,21%
3)	32	0,0001	/	0.5 (prvi) i 0.2 (drugi)	400	96,26%	91,65%
4)	32	0,0001	/	0,45 (prvi) i 0.3 (drugi)	500	96,91%	92,52%
5)	32	0,0005	/	0,45 (prvi) i 0,35 (drugi)	500	99,59%	94,42%
6)	32	0,0005	0,2 (oba)	0,5 (prvi i drugi)	500	99,34%	94,27%
7)	32	0,0005	0,5 (oba)	0,65 (prvi i drugi)	500	97,77%	95,45%
8)	32	0,0005	0,6 (oba)	0,7 (prvi i drugi)	500	95,51%	95,29%
9)	32	0,0005	0,65 (oba)	0,67 (prvi i drugi)	500	96,58%	95,86%

5.2 Diskusija

Ovaj rad je baziran na radu Amal Butija i ostalih [5]. U tom radu je predstavljen sistem za prepoznavanje saobraćajnih znakova i njihovo klasifikovanje. Korišćen je isti skup podataka kao i u ovom radu. Detekcija je bazirana na HOG-u i MPV-u dok je klasifikacija urađena konvolucijskom neuronskom mrežom baziranoj na *LeNet5* arhitekturi. U tom radu je u potpunosti ispoštovana *LeNet5* arhitektura. Odnosno, treći konvolucijski sloj nije zamenjen potpuno povezanim slojem. Ostatak arhitekture je skoro identičan. Za finalni model u tom radu se koristi brzina učenja od 10^{-4} što je 5 puta manje nego u slučaju ovog rada. I pored toga, na osnovu krive koja predstavlja tačnost mreže tokom epoha učenja možemo zapaziti da mreža postiže tačnost od preko 95% u prvih deset epoha. Iz ovoga

možemo zaključiti da je za obučavanje korišćena manja veličina beča. Rezultate koje je taj sistem postigao su 96,85% u detekciji i 96,23% u klasifikaciji.

U radu Domena Tabernika i Danijela Skočaja [6] je korišćen *Mask R-CNN* (*Mask Region-Based Convolutional Neural Network*) tip neuronske mreže. Ono što je interesantno za ovaj rad je to da skup podataka sadrži 200 kategorija različitih saobraćajnih znakova i obuhvata znakove kao što su putokazi. Pa je zbog toga ovo jedan od retkih radova koji obrađuju ovoliki broj različitih saobraćajnih znakova. Taj sistem je za saobraćajne znakove veličine do 50 piksela imao srednju preciznost od 96,0%.

6 Zaključak

Na početku je želja bila kreiranje sistema koji može da prepoznaje i klasifikuje saobraćajne znakove na slikama iz skupa *The German Traffic Sign Benchmark* i da se istraže savremene tehnike mašinskog učenja i njihova moguća primena u rešavanju ovog problema. Nakon inicijalnog velikog uspeha u ostvarenju prethodno navedenog cilja novi frontovi su bili otvoreni. Postavljen je novi cilj, a to je da se u ovom radu predstavi i kreira sistem koji može da u realnom vremenu detektuje i klasifikuje saobraćajne znakove. Sistem je trebalo da bude dovoljno robustan da podnese različite vremenske uslove i osvetljenja na samim video snimcima. Takođe je želja bila da se izvrši optimizacija koja bi omogućila izvršavanje i na uređajima sa malom računarskom i memorijskom snagom.

Sistem koji je ovde predstavljen daje dosta dobre rezultate kada je reč o slikama iz skupa *The German Traffic Sign Benchmark* i snimcima iz Nemačke. Podjednako dobro obrađuje scene iz gradova kao i one iz ruralnih područja. Na računaru koji ima procesor *Intel® Core™ i9-10900K* i 16 gigabajta memorije moguće je obraditi oko 30 frejmova po sekundi. U prethodnim poglavljima je pored opisa moguće implementacije sistema dat i pregled sprovedenog istraživanja u oblasti mašinskog učenja i njegove primene.

Sistem koji je ovde napravljen nije moguće koristi u komercijalnim svrhama zbog njegove nedovoljne preciznosti i tačnosti. Zbog velikih rezolucija slika koje obrađuje zahteva dosta računarske snage pa ga nije moguće koristi na mikrokontrolerima. S obzirom da je sistem obučavan nad znakovima iz Nemačke njegove performanse bi opale ako bi bio korišćen u nekoj drugoj državi. Ako je cilj korišćenje ovakvog sistema u drugim državama onda je potrebno ovaj sistem obučiti na skupu podataka iz te države. Performanse je moguće popraviti korišćenjem savremenih tehnika mašinskog učenja, kao što su YOLO (*You only look once*) ili rezidualne neuronske mreže, i dodatnom obradom slike i videa u cilju neutralisanja pomenutih šumova i nepravilnosti.

Transport je jedna od osnovnih grana ekonomije. Automatizacija vožnje bi mogla da podigne efikasnost i bezbednost transporta ljudi i robe. Preduslov za tako nešto je da vozila postanu inteligentne mašine koje mogu da prepoznaju svoju okolinu i da u zavisnosti od nadražaja koje dobijaju iz te okoline reaguju na smislen i proračunat način. Kompjuterski vid je osnova za ostvarivanje tog cilja.

Literatura

[1] Dalal N, Trigs B, „Histogram of oriented gradients for human detection.“ *Proceedings of IEEE computer vision and pattern recognition (CVPR)*, 2005.

[2] Marko M. Dabović, Igor I. Tartalja, „Duboke konvolucijske neuronske mreže – koncepti i aktuelna istraživanja.“, *Zbornik 61. Konferencije za elektroniku, telekomunikacije, računarstvo, automatiku i nuklearnu tehniku*, 2017.

[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, The MIT Press, 2016.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, lya Sutskever, Ruslan Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“, *Journal of Machine Learning Research*, 15, 1929-1958, 2014.

[5] Amal Bouti, M. A. Mahraz, Jamal Riffi, Hamid Tairi, „A robust system for road sign detection and classification using LeNet architecture based on convolutional neural network“, *Soft Computing*, 24, 6721–6733, 2020.

[6] Domen Tabernik, Danijel Skočaj, „Deep Learning for Large-Scale Traffic-Sign Detection and Recognition“, *IEEE Transactions on Intelligent Transportation Systems*, 21, 1427 – 1440, 2019.

[7] https://roadsignsdirect.co.uk/, (30.09.2021.)

[8] https://www.vectorstock.com/royalty-free-vector/traffic-sign-speed-limit-70-vector-24903314, (30.09.2021.)

[9] https://www.ontheroadtrends.com/wp-content/uploads/2020/07/Se%C3%B1aletica_TW-02.jpg, (30.09.2021.)

[10] https://www.benjoffe.com/code/tools/functions3d/examples, (30.09.2021.)

Spisak skraćenica

C - MPV	C – Mašina/e podržavajućih vektora.
CSV	*Comma Separated Values*.
HOG	Histogram orijentisanih gradijenata.
Mask R-CNN	*Mask Region-Based Convolutional Neural Network*.
MPV	Mašina/e podržavajućih vektora.
NMS	*Non-maximum suppression*.
ReLU	*Rectified Linear Unit*.
YOLO	*You only look once.*

Spisak slika

Spisak tabela