Detekcija i raspoznavanje prometnih znakova i njihove udaljenosti korištenjem OpenCV funkcija

Safić, Dario

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:596063

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

Image not found or type unknown

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Dario Safić

Detekcija i raspoznavanje prometnih znakova i njihove udaljenosti korištenjem OpenCV funkcija

Završni rad

Mentorica: izv. prof. dr. sc. Marina Ivašić-Kos

Rijeka, kolovoz 2020.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 17. veljače 2020.

Zadatak za završni rad

Pristupnik: **Dario Safić**

Naziv završnog rada: **Detekcija i raspoznavanje prometnih znakova i procjena njihove udaljenosti korištenjem OpenCV funkcija**

Naziv završnog rada na eng. jeziku: **Detection and recognition of traffic signs and distance estimation using OpenCV**

Sadržaj zadatka: Objasniti osnovne pojmove povezane sa strojnim učenjem, klasifikacijom slika i detekcijom objekata s naglaskom na postupak klasifikacije slika korištenje konvolucijskih dubokih neuronskih mreža. Proučiti TensorFlow i Keras okoline za dubinsko učenje i biblioteke iz OpenCV-a za detekciju objekata te opisati njihovo korištenje na problemu detekcije prometnih znakova i klasifikacije slika.

Opisati ekspriment učenja modela neuronske mreže za klasifikaciju prometnih znakova, matriku koja je korištena za evaluacije te objasniti postignute rezultate. Opisati korštenje HAAR kaskadnog algoritma za detekciju prometnih znakova te metodu triangulacije za aproksimaciju udaljenosti prometnih znakova od kamere.

Mentor

izv. prof. dr. sc. Marina Ivašić-Kos

Voditelj za završne radove

doc. dr. sc. Miran Pobar

Zadatak preuzet: 17.2.2020

Dario Safić

(potpis pristupnika)

Sadržaj

1. Sažetak 3

2. Uvod 4

2.1. Računalni vid 4

2.2. Konvolucijske neuralne mreže 6

3. Korišteni alati 9

3.1. TensorFlow 9

3.2. OpenCV 11

4. Učenje modela za klasifikaciju prometnih znakova 13

4.1. Izrada modela konvolucijske neuralne mreže 18

5. Detekcija i klasifikacija prometnih znakova na slici 24

5.1. HAAR kaskadni algoritam 24

5.2. Klasifikacija područja interesa 25

6. Aproksimacija udaljenosti 28

7. Rezultati izvođenja programa 30

8. Zaključak 34

9. Prilozi 35

10. Popis slika 36

11. Literatura 37

1. Sažetak

U radu je opisan proces izrade klasifikacijskog modela za klasifikaciju prometnih znakova, kao i njegova uporaba na praktičnim primjerima. Objašnjen je način i svrha korištenja raznih funkcija OpenCV knjižnice te proces uporabe HAAR kaskadnih algoritama u svrhu detekcije objekata. Postupak klasifikacije odvija se uporabom duboke konvolucijske neuronske mreže koja ima široku praktičnu primjenu u domeni računalnog vida. Za izradu duboke neuronske mreže koristi se platforma TensorFlow sa knjižnicom Keras.

Rješavanje navedenog problema odvija se u dvije faze; kaskadna klasifikacija slike, nakon čeka slijedi klasifikacija odabranog područja interesa (eng. *Region of interes*t). Metodom triangulacije dolazimo do aproksimacije udaljenosti prometnih znakova od leće kamere.

Ključne riječi: OpenCV, algoritam, model, Keras, TensorFlow, klasifikacija, konvolucijska neuralna mreža, duboko učenje

2. Uvod

Pojam strojnog učenja usko se povezuje sa najnovijim tehnološkim otkrićima i naprecima 21. stoljeća, a moderna ljudska svakodnevica podrazumijeva svjesno ili nesvjesno korištenje raznih softvera pogonjenim strojnim učenjem i/ili umjetnom inteligencijom. U prilog tome ide činjenica kako je 97% korisnika mobilnih uređaja isprobalo neki oblik glasovnog asistenta (eng. *Voice assistant*) koji je pogonjen umjetnom inteligencijom [1]. Takav softver ima za zadatak što bolje predvidjeti potrebe korisnika uz pomoć velike količine podataka [2].

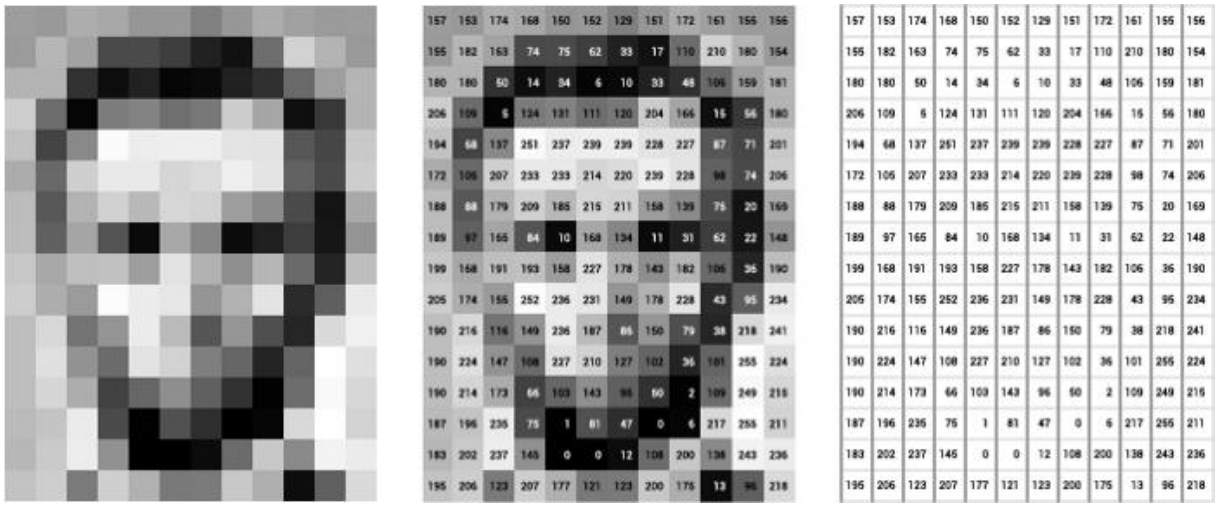
Kako je rasla potreba za rješavanjem složenijih zadaća uporabom računala, postalo je sve teže ručno stvarati algoritme koji su ih u stanju riješiti [3]. Model strojnog učenja koristi veliku količinu podataka kako bi, bez podrške programera, rješio određeni problem. Mnogi takvi problemi zahtjevaju prepoznavanje neposredne okoline ili određenih predmeta, a vizualnu percepciju računalima omogućava područje umjetne inteligencije „računalni vid“ [4].

2.1. Računalni vid

Računalni vid (eng. *Computer vision*) podrazumijeva područje umjetne inteligencije koje uči računalo kako interpretirati i shvatiti dijelove vizualnog svijeta [5]. Ipak, način na koji računalo promatra objekte različit je od onoga koji je intuitivan čovjeku. Čovjek okom objekte prepoznaje kao skup njegovih osnovnih komponenti, kao što su boja, oblik i dubina. Mozak interpretira signale iz očiju kako bi najprije pronašao rubove posmatranog objekta. Rubovi objekta zatim se udružuju u složeniju sliku koja čini oblik objekta [6].

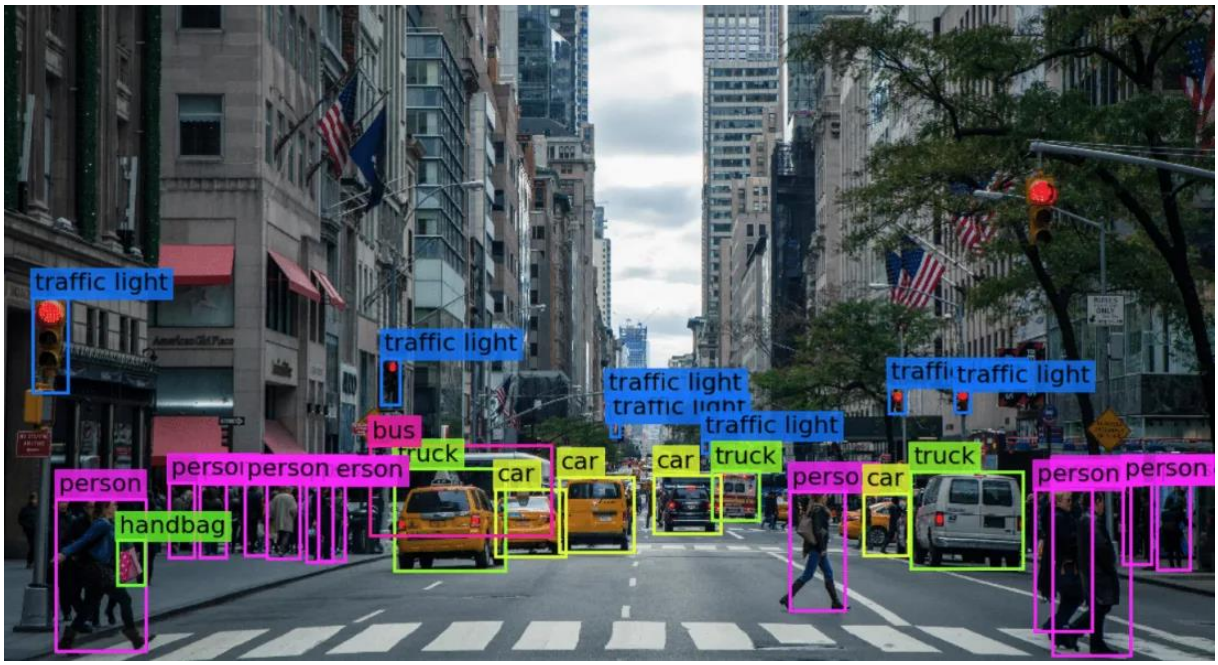
Računalo također započinje proces prepoznavanja objekta sa pronalaskom rubova objekta. Ključna je razlika što računalo takav objekt mora interpretirati kroz niz brojeva, s obzirom da je njegova slika sastavljena od velikog broja piksela (eng. Pixel). Računalo će svakom pojedinom pikselu dodijeliti određenu normaliziranu vrijednost tako što će promatrati razliku u svjetlini u slici koja je u tzv. „grayscale“ obliku [6].

„Grayscale“ u digitalnoj fotografiji i računalno-generiranim fotografija predstavlja fotografiju čiji je svaki piksel opisan samo razinom svjetline, tj. informacijom o jačini svjetlosti [7]. Vrijednosti svjetline kreću se od potpuno crne (označeno s 0), do potpuno bijele (označeno s 255). Ako u dijelu slike, (Slika1) dolazi do nagle promjene vrijednosti svjetline u susjednim pikselima, riječ je o rubu [6].



Slika 1 Dijagram vrijednosti piksela

Vrijednosti piksela kod slika sive razine spremaju u jednodimenzionalan niz (eng. One-dimensional array) [8]. Prije pojave strojnog učenja, pokušaji za implementaciju računalnog vida bili su gotovo bezuspješni, zbog potrebe za velikim naporima programera da iskoristi podatke ručno ih uspoređujući sa stvarnim slikama. Pojavom konvolucijskih neuralnih mreža došlo je do automatizacije tih zadataka. Na slici (Slika 2) vidljiv je rezultat uporabe konvolucijskih neuralnih mreža i računalnog vida u svrhu detekcije objekata na ulici.



Slika 2: Primjer rezultata detekcije objekata u prometu

Konvolucije su matematičke operacije koje služe kako bi se odredila razlika u vrijednosti piksela. Neuralna mreža te razlike koristi kako bi uočila ponavljanja. Slaganjem većeg broja konvolucijskih slojeva neuralna mreža dolazi do definiranih rubova i detalja slike koje koristi u drugim slojevima [6]. Dvodimenzionalna konvolucija u okvirima strojnog učenja je relativno jednostavna matematička operacija: početna težinska matrica koju nazivamo „kernel“ preslikava se preko matrice ulaznih podataka, gdje za svaki element ulazne matrice kernel izvodi množenje svih njenih elemenata na trenutnoj lokaciji, te njihovu sumu sprema kao jednu izlaznu vrijednost. Na taj način se od jedne dvodimenzionalne ulazne matrice stvara nova, izlaznu dvodimenzionalnu matricu [35].

2.2. Konvolucijske neuralne mreže

Konvolucijske neuralne mreže (CNN) su klasa dubokih neuralnih mreža koja se posebice koristi u analizi fotografija i ostalih vizualnih sadržaja [9]. Koriste ulaze (fotografije) kako bi dodijelile vrijednosti objektima unutar fotografije, u svrhu diferencijacije navedenih objekata. Naziv ove vrste neuralnih mreža proizlazi iz činjenice kako je konvolucija prisutna u barem jednom od njenih slojeva. Duboke konvolucijske neuralne mreže razlikuju se od klasičnih konvolucijskih mreža zbog uporabe većeg broja skrivenih slojeva koji zajedno s ulaznim i izlaznim slojevima tvore duboku neuralnu mrežu. Takva neuralna mreža ima sposobnost učiti bez nadzora čovjeka, filtrirajući informacije kroz veći broj skrivenih slojeva, slično kako to čini i ljudski mozak [46].

Danas, postoji velik broj besplatnih knjižnica (eng. *library*) koje olakšavaju proces izrade vlastitih neuralnih mreža, uključujući „Caffe“ [39], „Dlib“ [40], „PyTorch“ [41] i „TensorFlow“ [15].

2.2.1 Arhitektura konvolucijskih neuralnih mreža

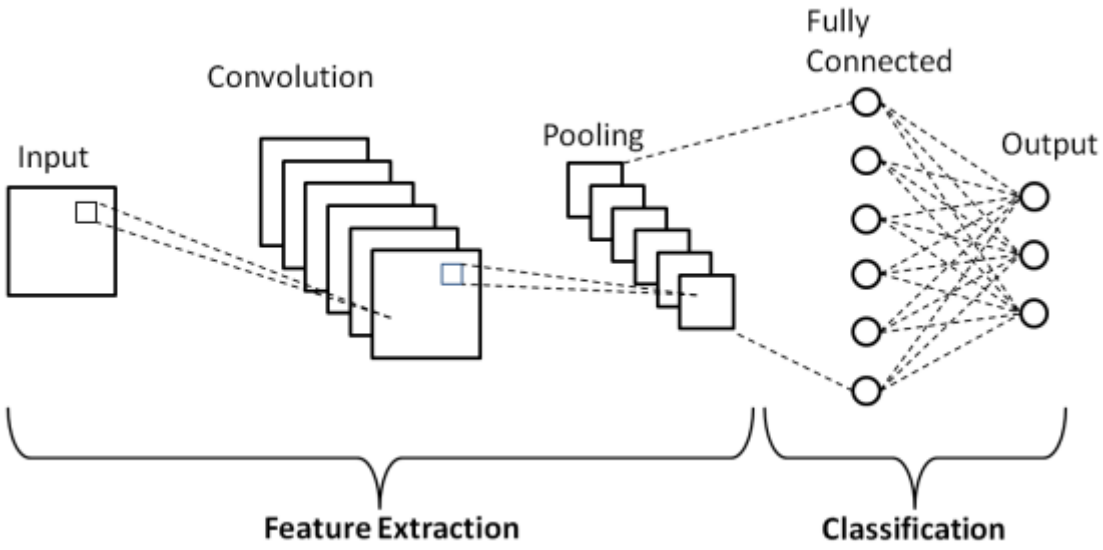
Arhitektura konvolucijskih neuralnih mreža sastoji se od ulaznog sloja koji određuje dimenziju njegovog ulaza, jednog izlaznog sloja koji može imati jedan ili više čvorova ovisno o vrsti neuralne mreže, i većeg broja skrivenih slojeva [9]. Skriveni su slojevi sastavljeni od većeg broja konvolucijskih slojeva, slojeva sažimanja, potpuno povezanih slojeva i normalizacijskih slojeva.

Konvolucijski slojevi procesiraju velik broj tensora trećeg reda (dvije prostorne i jedna semantička dimenzija) sastavljenih od broja slika, visine, širine i dubine slike, nad njima provode konvoluciju te rezultat konvolucije šalju sljedećem sloju. Izlazi konvolucijskih slojeva prolaze kroz aktivacijsku funkciju kako bi formirali mape značajki [36]. Filtri se primjenjuju na mapama značajki sloja koji prethodi te generiraju dvodimenzionalne matrice izlaznih vrijednosti. Konvolucijske mreže ne uče korištenjem jednog filtera, već se učenje provodi paralelnim djelovanjem velikog broja filtera (najčešće od 32 do 512) za dane ulazne podatke. To znači kako model može imati vrlo specijalizirane filtre za pojedine dijelove ulaznih podataka, poput specifičnih crta koje se pojavljuju u skupu za treniranje [45].

Slojevi sažimanja (eng. *pooling layers*) zatim mapiraju slične vrijednosti u jednu izlaznu vrijednost [11]. Nastoji se pritom istovremeno smanjiti količina podataka koja pristiže iz konvolucijskih slojeva i očuvati najbitnije podatke. Potpuno povezani slojevi uzimaju izlaze iz prijašnjih slojeva, te od njih stvaraju vektor koji u obliku pogodnom za slijedeći korak klasifikacije, tj. analize značajki i korištenje težinskih faktora kako bi im se odijelila odgovarajuća vjerojatnosti za svaku moguću klasu [12].

Nakon posljednjeg sloja sažimanja slijedi jedan ili više potpuno povezanih slojeva (eng. *fully connected layers*). Sloj povezuje svaki ulazni čvor sa svakim izlaznim čvorom slojeva s kojima je povezan. [9]. Vrši se klasifikacija ovisno o ulaznim podacima prijašnjih slojeva, a rezultat klasifikacije su izlazne vrijednosti vjerojatnosti za svaku oznaku koju model pokušava predvidjeti (dodjeljuju se vrijednosti 0 -1). Na slici (Slika

3) vidljiv je primjer jednostavne arhitekture neuralne mreže koja sadrži konvolucijske, slojeve sažimanja i potpuno povezane slojeve.



Slika 3 Primjer jednostavne CNN arhitekture

2.2.2 Primjena konvolucijskih neuralnih mreža

Konvolucijske neuralne mreže najčešće se koriste u sferi prepoznavanja fotografija, ali i procesuiranju prirodnih jezika, analizi videa i detekciji različitih anomalija. Zanimljiva je primjena konvolucijskih neuralnih mreža i u strateškim igrama na ploči [9], gdje su složene mreže sposobne nadjačati igrače u igrama poput Go, šaha, ili dame.

Naime, klasični kompjutorski Go programi evaluiraju milijune mogućih pozicija kako bi pronašli najbolji sljedeći potez. Takav pristup nije intuitivan čovjeku, zato neuralne mreže pristupaju problemu na potpuno drugi način. Predviđanjem poteza koje bi odigrao vrhunski igrač, ali i svih mogućih poteza, računalu s GPU procesorom i dovoljnim kapacitetom memorije je potrebna puno manja snaga i vremenski ulog kako bi odigrao „pravi“ potez [13].

Konvolucijske neuralne mreže koriste se i kao komponenta tehnologije za prepoznavanje lica u svrhu osobne identifikacije ili civilne zaštite. Postoji također i primjena u medicini, gdje složeni modeli mogu predvidjeti rizične faktore i generalno zdravstveno stanje pacijenata.

3. Korišteni alati

U izradi modela i programa za klasifikaciju i prepoznavanje prometnih znakova i mjerenja njihovih udaljenosti korišten je softver otvorenog koda za strojno učenje i računalni vid. Za kreaciju modela i klasifikaciju korišten je softver Keras i TensorFlow, dok je za prepoznavanje i računalni vid zadužena OpenCV knjižnica.

3.1. TensorFlow

TensorFlow je besplatna open-source knjižnica za protok podataka i diferencijacijskog programiranja [14]. Originalno nastao u 2011 kao „DistBelief“, vlastiti softver za strojno učenje baziran na dubokom učenju, Google se odlučuje za kreaciju kompletnog ekosustava za rješavanje problema iz stvarnog života uporabom strojnog učenja [15].

TensorFlow je vrlo brzo naišao na uporabu u različitim istraživačkim područjima, uključujući fiziku, robotiku, vjerojatnost i statistiku te analizi fotografija.



Slika 4 Tensorflow logotip

TensorFlow svoje funkcije programerima nudi kroz programski jezik „Python“, pa su iz tog razloga TensorFlow aplikacije zapravo Python aplikacije. Prednost koju knjižnica nudi za razvoj softvera strojnog učenja je visoka razina apstrakcije tako da korisnik ne mora vlastoručno implementirati algoritme ili povezivati slojeve mreže, već TensorFlow to obavlja „iza kulisa“, a na korisniku je samo da osigura ispravnu logiku svoje aplikacije. Također, prednost aplikacija kreiranih uporabom knjižnice je mogućnost distribucije na gotovo sve poznate platforme, mobilne uređaje, računala i oblak (eng. *cloud*) [16].

3.1.1 Keras

Keras je knjižnica otvorenog koda napisana u Pythonu sa podrškom za pokretanje na TensorFlow platformi. Dizajnirana je za jednostavno i brzo korištenje neuralnih mreža, a implementira velik broj funkcija i optimizacija za njihovu izradu [17].

Modeli izrađeni korištenjem Keras knjižnice dostupne su za mobilne platforme, osobna računala, web i mnoge druge. Važno je napomenuti kako buduća verzija TensorFlow platforme (TensorFlow 2.0) podržava i sadrži Keras knjižnicu.

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tenso
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

Slika 5: Primjer jednostavnog Keras programa

Na primjeru sa slike (slika 5) prikazan je jednostavan Keras program. Funkcijom Input() instancira se novi Keras tensor koji se sprema u varijablu „inputs“. Pozivaju se tri instance gustog sloja na tensor, a novi tensor se sprema u varijablu „predictions“. Zatim, kreira se model čiji je ulaz početni tensor varijable „inputs“, a izlaz tri gusta sloja varijable „predictions“. Konfigurira se model sa funkcijom gubitka i metrikama, a funkcijom model.fit() započinje se učenje modela.

Dostupne su gotove funkcije za kreaciju različitih slojeva modela, a svaki poziv funkcije sloja zahtjeva izvođenje samo jedne linije koda. Kompilacija, učenje i evaluacija modela izvode se pozivima funkcija koje su definirane u biblioteci, a također se pozivaju se tek jednom linijom koda.. Kod je prema tome čitljiv i razumljiv za nove korisnike.

3.2. OpenCV

OpenCV je knjižnica sa visokom razinom optimizacije za izradu aplikacija u stvarnom vremenu [19]. Podržava većinu modernih platformi, a njen je kod slobodno dostupan za komercijalnu uporabu. Stvorena je s namjerom da pruži jednostavan i univerzalan sustav za izradu aplikacija temeljenih na računalnom vidu i strojnom učenju. Knjižnica sadrži preko 2500 algoritama optimiziranih za izradu aplikacija za npr. detekciju ljudskog lica, detekciju i raspoznavanje objekata, raspoznavanje ljudi u videozapisima, praćenje kretnje objekata na slici, uklanjati tzv. „crvene oči“ sa fotografija slikanih sa aktivnim blicom, i slično [20]. Knjižnicu koriste tvrtke, istraživačke skupine i ostala znanstvena tijela, a dijelovi njenih mogućnosti mogu se pronaći u rješenjima tehnoloških divova poput Googlea, Yahooa, Microsofta, IBM-a i drugih [20].



Slika 6: OpenCV logo

U sljedećem primjeru prikazan je jednostavan OpenCV/Python program koji ima za zadatak učitati sliku iz datoteke *„starry_night.jpg“* i spremiti ju u varijablu *img*, zatim provjeriti je li učitana ispravna datoteka, te naposlijetku prikazati sliku u novom prozoru s naredbom imgshow() sve dok korisnik pritiskom na dugme „s“ ne naredi zatvaranje i spremanje fotografije u novu datoteku .png ekstenzije. Zahvaljujući uporabi Python jezika, i OpenCV knjižnice kod je vrlo čitljiv i jednostavan za korištenje.

```
import cv2 as cv
import sys

img = cv.imread(cv.samples.findFile("starry_night.jpg"))

if img is None:
    sys.exit("Could not read the image.")

cv.imshow("Display window", img)
k = cv.waitKey(0)

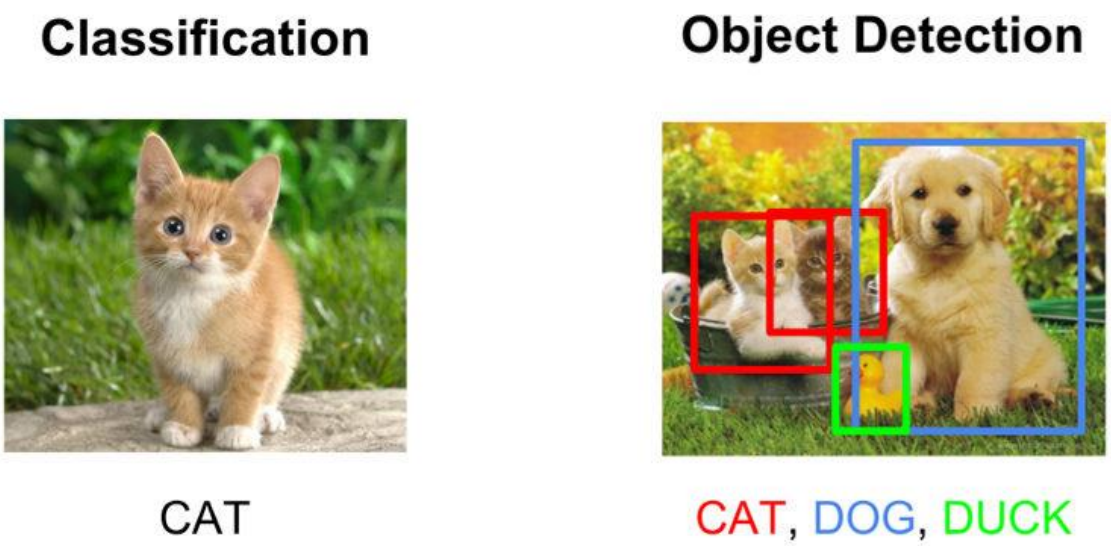
if k == ord("s"):
    cv.imwrite("starry_night.png", img)
```

Slika 7: Primjer jednostavnog programa u OpenCV

Važno je napomenuti kako je dostupna opširna i kvalitetna službena dokumentacija za sve korisnike OpenCV knjižnice, a postoji i mnoštvo gotovih projekata koji se mogu naći na platformama poput GitHub-a.

4. Učenje modela za klasifikaciju prometnih znakova

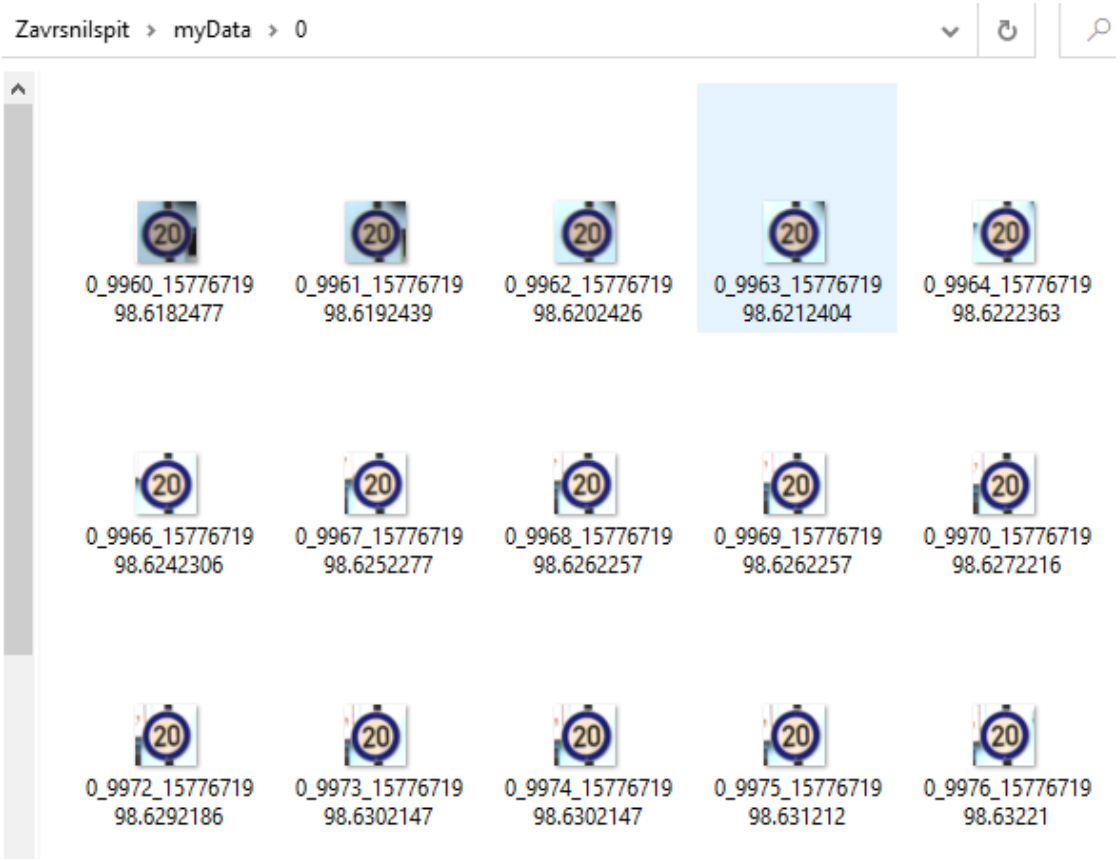
Kada govorimo o uporabi računalnog vida u svrhe prepoznavanja objekata, rješenje najčešće možemo podijeliti na zadatak klasifikacije i zadatak detekcije. Zadatak klasifikacije je odrediti kojoj klasi iz većeg broja klasa pripada svaki pojedini ulazni podatak (npr. fotografija, grupa piksela ili vektora) [37]. Kao klasifikacijski model u posljednje vrijeme se najčešće koriste duboke konvolucijske neuronske mreže. S druge strane, zadatak detekcije je osim određivanja klase kojoj objekti na slici pripadaju i određivanje točne pozicije ili lokacije tih objekata na slici. Kada modelu dodijelimo ulaznu sliku, prepoznati će na njoj klase i lokacije svih objekata i odrediti kolika je vjerojatnost da je detekcija uspješna [38]. Na sljedećem primjeru vidljiva je razlika između funkcije klasifikacije i detekcije (Slika 8), kod klasifikacije cijelu sliku klasificiramo kao 'Cat' jer se na njoj nalazi mačka i pri tom ne vodimo računa gdje se točno mačka nalazi na slici, a u slučaju detekcije cilj je označiti točno područje slike na kojem se nalazi određeni objekt od interesa.



Slika 8: Razlika zadataka klasifikacije i detekcije slike

U ovom radu prvi je cilj bio naučiti model za klasifikaciju prometnih znakova tako da se može odrediti sadrži li slika prometni znak, odnosno ako slici sadrži prometni znak o kojem se znaku radi. Nakon toga cilj je bio na slici koja sadrži prometnu scenu detektirati poziciju prometnog znaka i klasu kojoj pripada

Da bi se na slici mogli raspoznati prometni znakovi bilo je potrebno naučiti model za klasifikaciju prometnih znakova. U folderu „myData“ pohranjeni su primjeri slika za 43 klase prometnih znakova. Ukupno, skup podataka čini 36.413 fotografija. Format datoteke je .png veličine 32 x 32 piksela, a svaka slika znaka pohranjena je u mapu koja odgovara toj kategoriji znaka. Na slici 9 prikazani su primjeri slika koje odgovaraju prometnom znaku „ograničenje brzine 20km/h“. U praksi, to znači kako će budući model biti u stanju raspoznati 43 različita prometna znaka i jednu dodatnu kategoriju u kojoj se nalazi sve ono što nije prometni znak a nalazi se u njegovoj neposrednoj blizini (zid, stablo, svijetlo semafora, itd.).

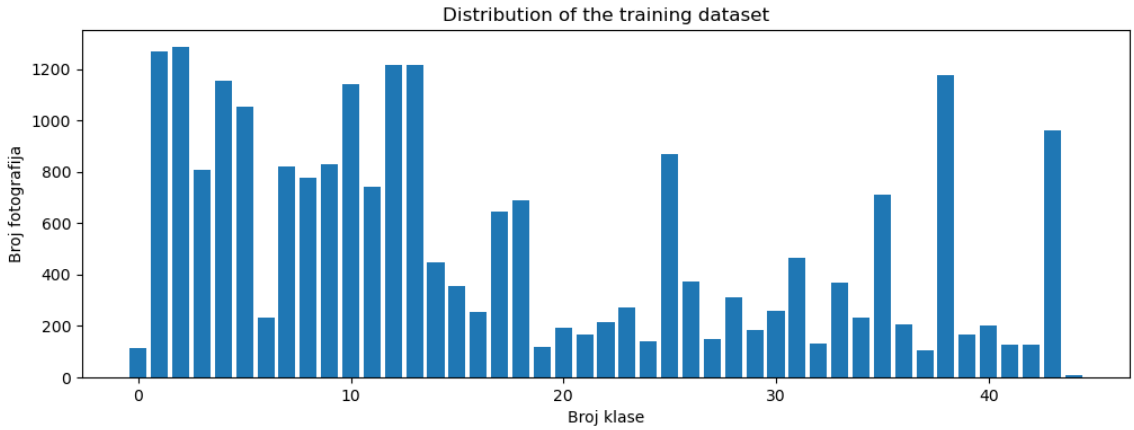


Slika 9: Primjer znakova za klasu „ograničenje 20km/n“

Svaka fotografija iterativno se učitava i pohranjuje u polje u glavnom dijelu programa, a njihova distribucija po klasama vidljiva je na grafu. Količina podataka nije približno jednaka za sve klase, što može dovesti do sklonosti da model „preferira“ određenu kategoriju nad drugom i samim time smanjiti preciznost modela. Ipak, u primjeni se pokazalo kako je takvo odstupanje u ovom slučaju zanemarivo u odnosu na povećanje točnosti modela do koje dovodi veća količina podataka. Svakoj fotografiji dodijeljena je i odgovarajuća oznaka s indeksom klase koja se učitava iz jednostavne .csv datoteke. Na slici (Slika 10) vidljivi su primjeri tri kategorije znakova sa odgovarajućom oznakom.

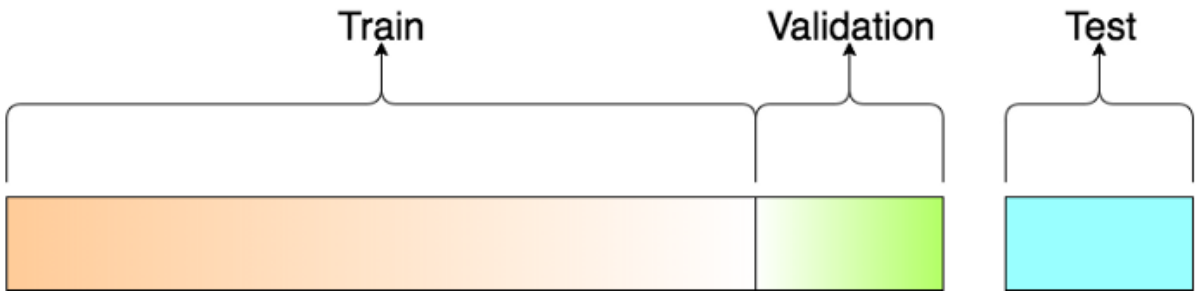


Slika 10: Primjer uparivanja slike i oznake



Slika 11: Distribucija fotografija po kategorijama

Na slici (Slika 11) vidljiva je raspodjela ukupnog skupa slika po klasama. Od ukupno 36.413 fotografija, 80% fotografija je nasumično odabrano za učenje a 20% za testiranje. 20% fotografija iz skupa za učenje koristi se za validaciju modela tijekom učenja (Slika 12). Skup podataka za učenje koristi se za učenje modela, skup podataka za validaciju koristi se kako bi se procijenila stvarna greška modela i podesili osjetljivi hiperparametri modela, dok je skup podataka za testiranje neovisan o ostalim skupovima i služi za procjenu točnosti dobivenog modela na novim podacima na kojima model nije bio učen [21]. Hiperparametri modela uključuju primjerice broj i veličinu skrivenih slojeva, inicijalizaciju težinksih faktora, stopu učenja i sl [23].



Slika 12: Vizualizacija raspodjele podataka

Sve fotografije u skupu za učenje zatim se pretvaraju u „grayscale“, izjednačuje im se svjetlina, te se vrijednosti pojedinih piksela normaliziraju između 0 i 255 kako bi definirali razinu svjetlosti svakog piksela. Skupovima podataka zatim se dodijeljuje vrijednost dubine „1“ kao četvrta dimenzija. Na slici (Slika 13) vidljiv je programski kod kojim se izvršavaju prethodno navedene operacije, a svakoj od njih definirana je vlastita funkcija kako bi se operacije mogle ponovno upotrijebiti.

```
def grayscale(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img

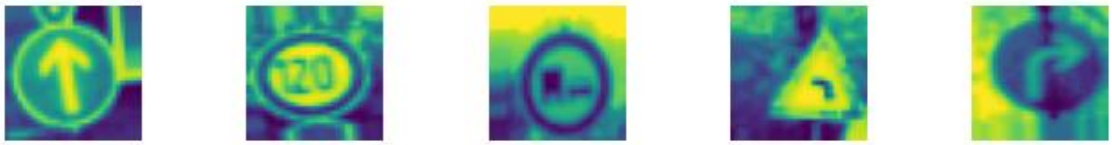

def equalize(img):
    img = cv2.equalizeHist(img)
    return img


def preprocessing(img):
    img = grayscale(img)
    img = equalize(img)
    img = img/255
    return img
```

Slika 13: Prikaz funkcija preprocesuiranja fotografije

Nadalje, koristi se proces augmentacije skupa slika s sitnim korekcijama koje ne utječu na promjenu informacije na slici kao što su to zakretanja, pomaci, smanjivanje ili povećanje slike, rezanja dijela slike i slične promjene fotografije kako bi proširili skup podataka s kojim radimo. Svakom augmentacijom nastaje nova, generirana slika čije se vrijednosti za pojedine piksele razlikuju u odnosu na originalnu sliku. Drugim riječima, od jedne ulazne slike možemo stvoriti veći broj izlaza koje će neuronska mreža tretirati kao potpuno različite slike [24]. Konvolucijske neuralne mreže imaju sposobnost klasificirati objekte čak i kada su vrijednosti piksela međusobno različite s obzirom na rotaciju, pomak, ili promjene u veličini fotografije, zahvaljujući svojstvu nepromjenjivosti [42]. Proces augmentacije značajno poboljšava točnost klasifikacije fotografija [25], bez potrebe za stvarnim proširenjem skupa podataka.

Za augmentaciju fotografije koristimo ugrađenu funkciju knjižnice Keras, „ImageDataGenerator“. Funkcija ima sposobnost obaviti različite transformacije na fotografiji, uključujući rotaciju, pomaka u odnosu na širinu ili visinu, približavanja ili udaljavanja fotografije i sl. Nasumično generirani rezultati procesa augmentacija vidljivi su na primjeru (Slika 14).



Slika 14: Primjer augmentiranih fotografija

4.1. Izrada modela konvolucijske neuralne mreže

Za izradu modela konvolucijske neuralne mreže koristili smo klasu „Sequential“ knjižnice Keras, koja grupira linearan stog slojeva jedan povrh drugog u modelu. Slojevi se sada vrlo jednostavno dodaju na stog funkcijom add(). U izradi modela koristili smo tri vrste slojeva: konvolucijske slojeve, „max pooling“ slojeve sažimanja i potpuno povezane (softmax) slojeve.

Najprije podešavamo hiperparametre modela, broj i veličinu filtera, „pool size“ i broj čvorova.

Prvi sloj svakog sekvencijalnog modela mora sadržavati podatak o obliku ulaza, što je u našem slučaju [32 x 32] (veličina fotografija u pikselima). Prva dva sloja koje dodajemo u stog su konvolucijski slojevi sa veličinom filtra 5x5. Zbog toga što su ulazi veličine [32x32x3], svaki neuron konvolucijskog sloja sadržavati će težine za prostor koji odgovara [5x5x3] dijela ulaza, točnije 75 težina. Zatim, na stog se dodaje jedan sloj sažimanja koji ima zadaću smanjiti prostorni udio trenutačne reprezentacije kako bi se smanjio broj potrebnih parametara, a samim time smanjio broj izračuna u mreži. Pooling layer također kontrolira učestalost pojave „overfittinga“, kako se model ne bi fiksirao samo na dio podataka u skupu za učenje i modelirao ga precizno do razine gdje to negativno utječe na njegovu generalizaciju i rezultate testiranja klasifikacije na novim podacima [26]. Iz tog razloga nužno je osigurati što manju šansu za pojavu prevelikog prilagođavanja modela

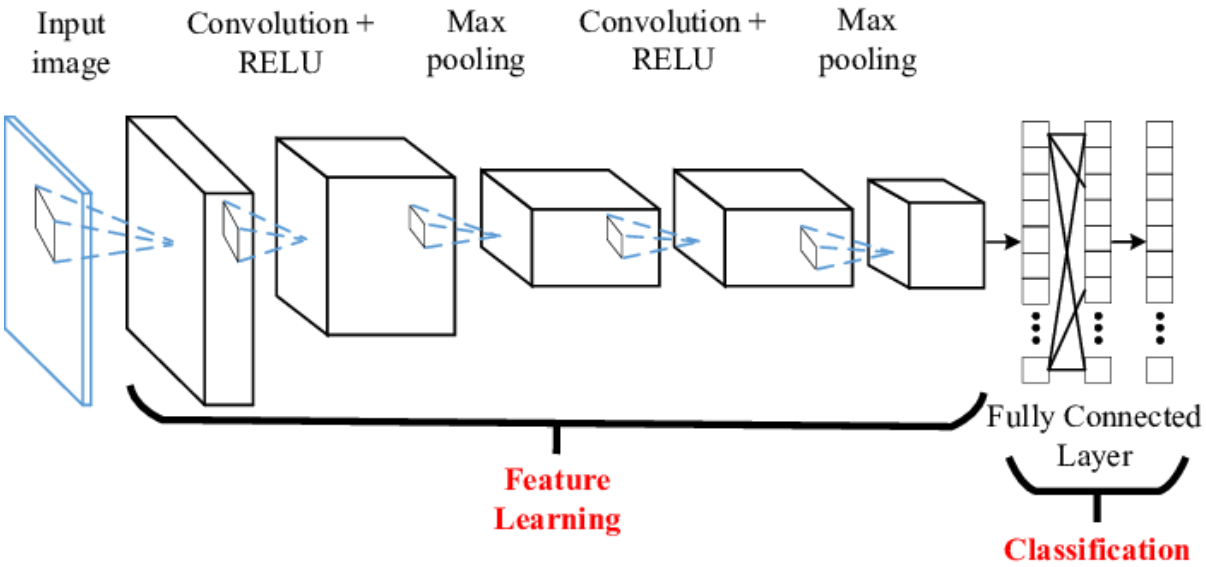
podacima u skupu za učenje (eng. *overfitting*). U ovome slučaju, koristimo najučestaliji oblik sloja sažimanja, sa filterom veličine [2x2].

Zatim slijede još dva konvolucijska sloja, sa veličinom filtera [3x3], kako bi „ulovili“ manje kompleksne apstrakcije. Iza ovog para konvolucijskih slojeva također koristimo sloj sažimanja. Niz konvolucijskih i slojeva sažimanja zaključujemo sa dodatnim slojem izostavljanja neurona (eng. *dropout*), čime se također smanjuje šansa za pojavom prevelikog prilagođavanja modela podacima dostupnim za učenje.

Prije dodavanja potpuno povezanih slojeva, potrebno je na stog dodati i sloj poravnanja (eng. flatten), koji će ulaznu matricu „izravnati“ u polje. Definiramo potpuno povezane slojeve koji generiraju vjerojatnosti za svih 45 dostupnih oznaka. Prvi od njih je običan gusti sloj koji koristi 500 čvorova, iza kojeg slijedi dodatni sloj izostavljanja neurona kako bi se još jednom smanjila učestalost prevelikog prilagođavanja modela podacima u skupu za učenje.

Izlazni „softmax“ sloj ima broj čvorova jednak ukupnom broju klasa koje se klasificiraju (u našem slučaju, to je 45 klasa koje odgovaraju prometnim znakovima).

Grafički, ovaj model može se prikazati kao:



Slika 15: Grafički prikaz modela neuralne mreže

```
no_filters = 60
filter_size = (5, 5)
filter_size2 = (3, 3)
pool_size = (2, 2)
no_nodes = 500

model = Sequential()
model.add((Conv2D(no_filters, filter_size, input_shape=(dimension[0], dimension[1], 1), activation='relu')))
model.add((Conv2D(no_filters, filter_size, activation='relu')))
model.add(MaxPooling2D(pool_size=pool_size))

model.add((Conv2D(no_filters//2, filter_size2, activation='relu')))
model.add((Conv2D(no_filters // 2, filter_size2, activation='relu')))
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(no_nodes, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(noOfClasses, activation='softmax'))
model.compile(Adam(lr=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
return model
```

Slika 16: Programski kod modela konvolucijske neuralne mreže

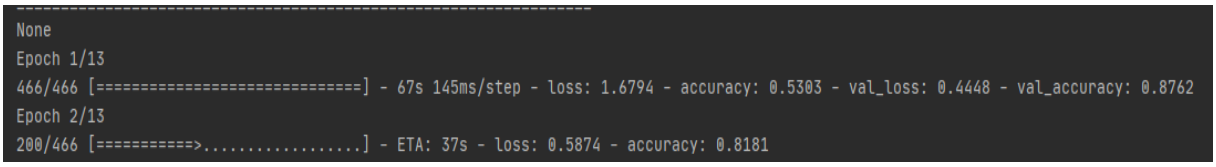
Prije nego što može započeti proces učenja modela, potrebno je konfigurirati i sastaviti (eng. *compile*) model [27]. Kao parametre koristimo osnovni optimizator „Adam“, funkciju gubitka (eng. *loss*) „categorical_crossentropy“, te čuvamo metrike o preciznosti modela.

Učenje modela u Kerasu je iznimno jednostavno, izvodi se uporabom samo jedne funkcije (fit()) i nekoliko pripadajućih parametara. Parametri koje dodijeljujemo funkciji su:

- Skup podataka za učenje, koji uključuje slike (označene sa „X_train“) i pripadajuće oznake (označene sa „y_train“).
- Veličina skupa podataka (eng. *batch size*), koja definira koliki udio ukupnog skupa podataka će algoritam izdvojiti za učenje istovremeno. Korištenje ove metode zahtjeva manji utrošak memorije, a neuralna mreža će učiti brže nego u slučaju gdje se nisu koristili skupovi podataka [28]. U ovome slučaju, koristiti ćemo veličinu skupa podataka od 50 slika.
- Broj koraka po epohi, koji se tradicionalno izračunava formulom: (veličina skupa za učenje // veličina skupa podataka). Svaki korakom procesuira se onoliko fotografija koliko je zadano veličinom skupa podataka. S obzirom kako broj fotografija u skupu za učenje iznosi 23.304, a veličina skupa podataka je 50, ukupno se po epohi izvodi 466 koraka.

- Broj epoha, gdje se broj određuje intrinzično, analizom grafova nakon završetka učenja procijenjujemo broj epoha koji će dati najveću točnost u najmanje vremena (tražimo posljednju epohu kod koje ne uočavamo preveliko prilagođavanje modela).
- Skup podataka za validaciju, koji uključuje slike (označene sa „X_validation“) i pripadajuće oznake (označene sa „y_validation“).

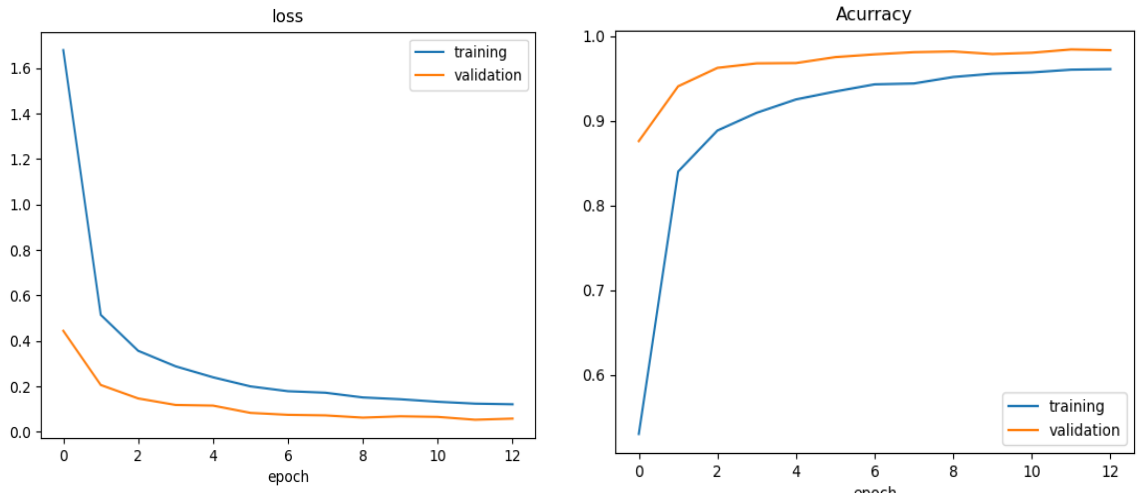
Pokretanjem glavnog programa započinje se učenje modela i u područje konzole ispisuju se rezultati pojedinih epoha u obliku:



Slika 17: Primjer faze jedne epohe

Završetkom posljednje epohe završava se učenje modela, a u području konzole se ispisuje rezultat učenja. Uz pomoć Pythonove knjižnice Matplotlib na ekranu se također prikazuju grafovi s ucrtanim rezultatima učenja modela na skupovima za učenje i validaciju (Slika 18). Uobičajen metrike koje se koriste za evaluaciju modela su gubitak (engl. *Loss*) i točnost (engl. *Accuracy*). Funkcija gubitka računa i vraća rezultat razlike očekivane vrijednosti i rezultata izlaznog sloja neuralne mreže [43] Za razliku od rezultata točnosti modela koji se izražava u postocima, rezultat gubitka je decimalan broj. Točnost podrazumijeva udio ispravnih predviđanja u skupu ukupnih predviđanja koje je model napravio [44]. Formalno, točnost definiramo kao:

$$točnost = \frac{broj\ točnih\ predviđanja}{ukupan\ broj\ predviđanja}$$



Slika 18: Rezultati učenog modela

Sa grafa (Slika 18) je vidljivo kako je postignuta visoka razina točnosti na skupu za učenje i na skupu za validaciju podataka, kao i stabilne i niske stope funkcije gubitka. Niske razine funkcije gubitka daju naslutiti visoku preciznog samog modela. Izostanak većih anomalija u obliku grafova u bilo kojoj etapi govori kako nije došlo do značajnog prilagođavanja modela na skup podataka za učenje.

Kako bi izračunali i analizirali rezultat učenja modela na testnom skupu podataka koristimo funkciju knjižnice Keras model.evaluate(). Izvođenjem navedene funkcije vraćaju se vrijednosti gubitka i točnosti za trenirani model, a rezultati su vidljivi u konzoli:

```
Epoch 13/13
466/466 [==============================] - 70s 149ms/step - loss: 0.1216 - accuracy: 0.9613 - val_loss: 0.0587 - val_accuracy: 0.9837
Test Score: 0.05343569815158844
Test Accuracy: 0.9848963618278503
```

Slika 19: Prikaz rezultata na testnom skupu

kako je vidljivo sa slike (Slika 19), „Test Score“ predstavlja vrijednost gubitka, a „Test Accuracy“ označava točnost modela na testnom skupu. Uočavamo kako su vrijednosti točnosti i gubitka na testnom skupu slični ili bolji nego na skupovima validacije i treninga. Možemo zaključiti kako je generalizacija modela vrlo dobra, te nije došlo do prilagođavanja modela na skup podataka za učenje, u kojem slučaju bi rezultat na testnom skupu bio lošiji nego na ostalim skupovima podataka.

Ovakav se model čini dovoljno pouzdanim i spreman je za korištenje u procesu izrade detektora prometnih znakova. Preostaje još samo pohraniti navedeni model u datoteku, za što koristimo funkciju .save() te spremamo model kao datoteku .h5 ekstenzije.

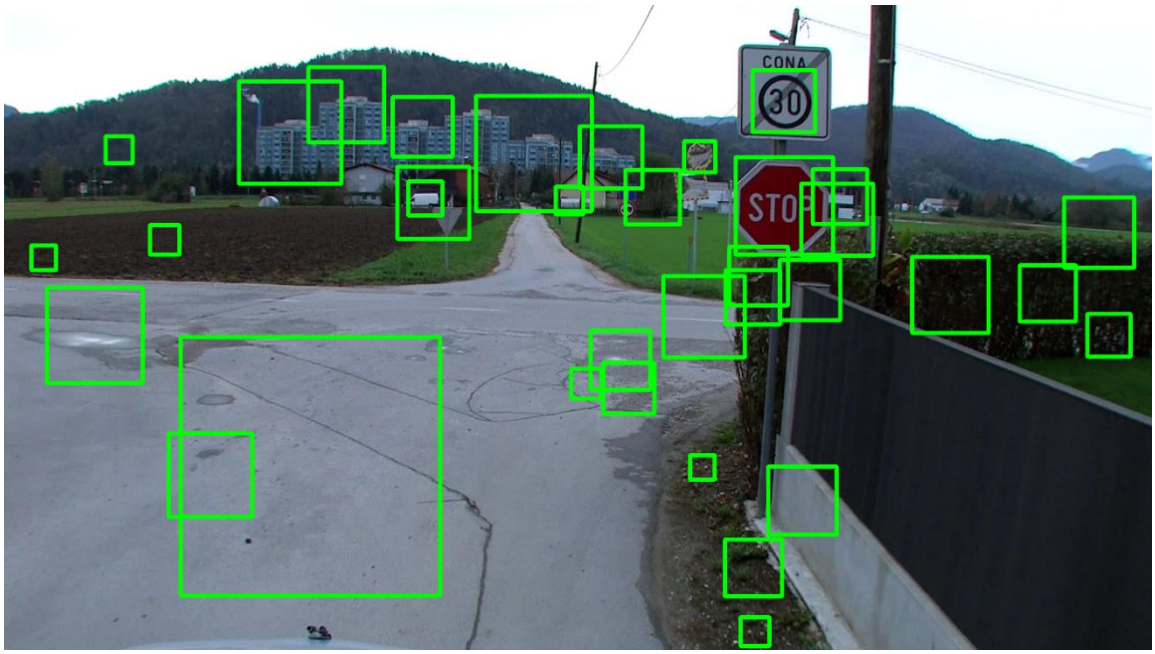
5. Detekcija i klasifikacija prometnih znakova na slici

Detekcija i klasifikacija prometnih znakova odvija se u dvije faze; kaskadna detekcija objekata na slici, nakon čega slijedi specifična klasifikacija odabranog područja interesa (eng. *Region of interes*t). Detekcijom s HAAR kaskadnim algoritmom pronalazi se veći broj područja interesa, od kojih samo nekoliko može predstavljati prometne znakove. Iz tog razloga, svako područje interesa predstavlja samo potencijalni prometni znak koji model konvolucijske neuralne mreže potom klasificira kao određeni prometni znak, ili kao pogrešnu detekciju.

5.1. HAAR kaskadni algoritam

Haar kaskadni algoritam je detekcijski algoritam temeljen na strojnom učenju koji se koristi za prepoznavanje objekata na fotografiji ili videozapisu [30]. Kaskadna funkcija uči na velikoj količini pozitivnih i negativnih fotografija. U našem slučaju, algoritmu je potrebna velika količina fotografija prometnih znakova (pozitivnih fotografija), i svega onoga što se može pronaći u blizini prometnih znakova, a to nije, poput ceste, drveća, ograde i vozila (negativne fotografije) [31]. Krajnji rezultat je algoritam izuzetne brzine izvođenja koji može vrlo dobro detektirati velik broj područja fotografije.

U praktičnoj primjeni gdje se problemu prepoznavanja i klasifikacije pristupa u dvije faze kao što je ovdje slučaj, povoljno je algoritmu dopustiti „opuštenije“ prepoznavanje. Naime, s obzirom da potencijalna područja interesa moraju proći kroz drugi stupanj klasifikacije, detektirati veći broj pogrešnih područja je optimalnije nego propustiti detekciju traženog područja interesa (prometnog znaka).



Slika 20: Primjer izvršene detekcije HAAR kaskadnim algoritmom

Kao što je vidljivo sa slike (Slika 20), detektiran je veći broj pogrešnih područja, no rezultat detekcije jest povoljan jer su oba prometna znaka uspješno detektirana kao područja interesa.

Sva detektirana područja interesa pohranjuju se u novu listu u obliku spremnom za korištenje modelu neuralne mreže.

U domeni OpenCV-a, detekcija HAAR kaskadnim algoritmom [31] provodi se s tek dvije ugrađene funkcije. Funkcijom cv2.CascadeClassifier() učitava se .xml datoteka klasifikatora. Funkcijom cv2.CascadeClassifier.DetectMultiScale() izvodi se detekcija na fotografiji čija se lokacija definira u parametru funkcije [31]. Kako bi detekcija bila preciznija, fotografija se najprije pretvara u svoj „grayscale“ oblik korištenjem ugrađene funkcije cv2.cvtColor() s parametrom „cv2.COLOR_BGR2GRAY“.

5.2. Klasifikacija područja interesa

U glavnom dijelu programa učitava se model konvolucijske neuralne mreže sa ciljem klasifikacije detektiranih područja interesa. Svako područje interesa sprema se kao zasebna fotografija, a njena se veličina postavlja na [32x32] piksela, kako bi odgovaralo veličini slika na kojima je model učen. Nova fotografija se pretvara u „grayscale“ oblik te se vrijednosti

svjetline spremaju za svaki pojedini piksel u vrijednosti od 0 do 255 u novi niz Pythonove biblioteke Numpy.

Naredbom model.predict(), model koristi ulazni niz kako bi generirao izlazne vrijednosti predviđanja za svaku pojedinu oznaku. Takve izlazne vrijednosti koristimo kao parametar u funkciji amax() biblioteke Numpy kako bi dobili i pohranili najveću pripadnu vrijednost oznake. Funkcijom model.predict_classes također pohranjujemo indeks pripadajućih klase najizglednije oznake. (Slika 21).

```
# Model koristi niz kako bi predvidio vrijednosti za svaku oznaku (kategoriju)
predictions = model.predict(imgNew)
classIndex = model.predict_classes(imgNew)
# Sprema se vrijednost vjerojatnosti za najizgledniju kategoriju
probabilityValue = np.amax(predictions)
```

Slika 21: Programski kod predviđanja modela

Također, definiramo vrijednost koja označava minimalan prag (eng. *threshold*) koji najveća vrijednost vjerojatnosti oznake mora postići kako bi se detekcija smatrala uspješnom. Ukoliko model nije uspio dodijeliti vrijednost većoj od minimalnog praga niti jednoj oznaci, tada možemo reći kako je rezultat predviđanja nepouzdan i takvo područje interesa odbacujemo. Važno je istaknuti i kako jedna od oznaka odgovara svemu onome što nije prometni znak. Zahvaljujući tome model je sposoban prepoznati je li područje interesa zapravo pogrešna detekcija. U slučaju da je model najveću vjerojatnost dodijelio upravo toj oznaci, takvo područje interesa također odbacujemo.

Nakon provođenja prethodna dva uvjeta možemo reći kako je na području interesa detektiran i klasificiran jedan prometni znak s vrijednosti vjerojatnosti većoj od traženog minimalnog praga. Svako takvo područje interesa ucrtava se na originalnu fotografiju i dodjeljuje mu se pripadni tekst koji sadrži naziv prometnog znaka i odgovarajuću vjerojatnost (Slika 22).



Slika 22: Primjer uspješne detekcije i klasifikacije fotografije

Važno je napomenuti kako se sva odbačena područja interesa koja nisu zadovoljila oba uvjeta pohranjuju u mapu „images“ kako bi se mogla koristiti u svrhe daljnjeg učenja modela.

6. Aproksimacija udaljenosti

Kako bi odredili udaljenost prometnog znaka od objektiva kamere koristiti ćemo se zakonitosti kako je omjer percipirane veličine objekta na senzoru i stvarne veličine objekta jednak omjeru fokalne udaljenosti i udaljenosti leće od objekta [33]. S obzirom kako širina fotografije može biti različita od širine traženog objekta, a u ovom slučaju traži se udaljenost od specifičnog dijela fotografije, omjer širine fotografije i širine objekta na fotografiji u pikselima proporcionalna je udaljenosti objekta od leće (Što je omjer širine fotografije i širine objekta veći, udaljenost između njih je veća). Uvrštavanjem navedenog u formulu vrijedi kako je:

$$udaljenost\ (mm) = \frac{fokalna\ udaljenost\ (mm)\ x\ stvarna\ širina\ (mm)\ x\ širina\ fotografije\ (piksel)}{širina\ objekta\ (piksela)\ x\ percipirana\ širina\ (mm)}$$

S obzirom kako je širina područja interesa u pikselima jednaka širini prometnog znaka u pikselima (područje interesa obuhvaća upravo područje prometnog znaka), možemo preformulirati izraz:

$$udaljenost\ (mm) = \frac{fokalna\ udaljenost\ (mm)\ x\ stvarna\ širina\ (mm)}{percipirana\ širina\ (mm)}$$

Važno je napomenuti kako nije moguće odrediti veličinu objekta na fotografiji s velikom preciznošću s obzirom kako fokalna duljina može u potpunosti varirati od fotografije do fotografije, a izračun fokalne udaljenosti u praksi ne može biti precizan. Ipak, ovdje smo se odlučili za prosječnu fokalnu dužine leće koja se nalazi u auto kamerama za snimanje vožnje [34].



Slika 23: Primjer fotografije s percipiranom udaljenosti (isječak)

Krajnji rezultat detekcije i klasifikacije na fotografiji sa aproksimiranom udaljenosti vidljiv je na primjeru iznad (Slika 23). Takvi se rezultati automatski spremaju prilikom izvođenja programa u podmapu „results“.

7. Rezultati izvođenja programa



Slika 24: Primjer 1

Na primjeru (Slika 24) vidljivo je kako je program uspješno prepoznao znak za zabranu prolaska teških vozila. Iako su na slici prisutni mnogi elementi koji bi se mogli interpretirati kao prometni znakovi (kotači automobila, udaljeni prozori), algoritam je uspješno prepoznao kako je riječ o nečemu što nije prometni znak.

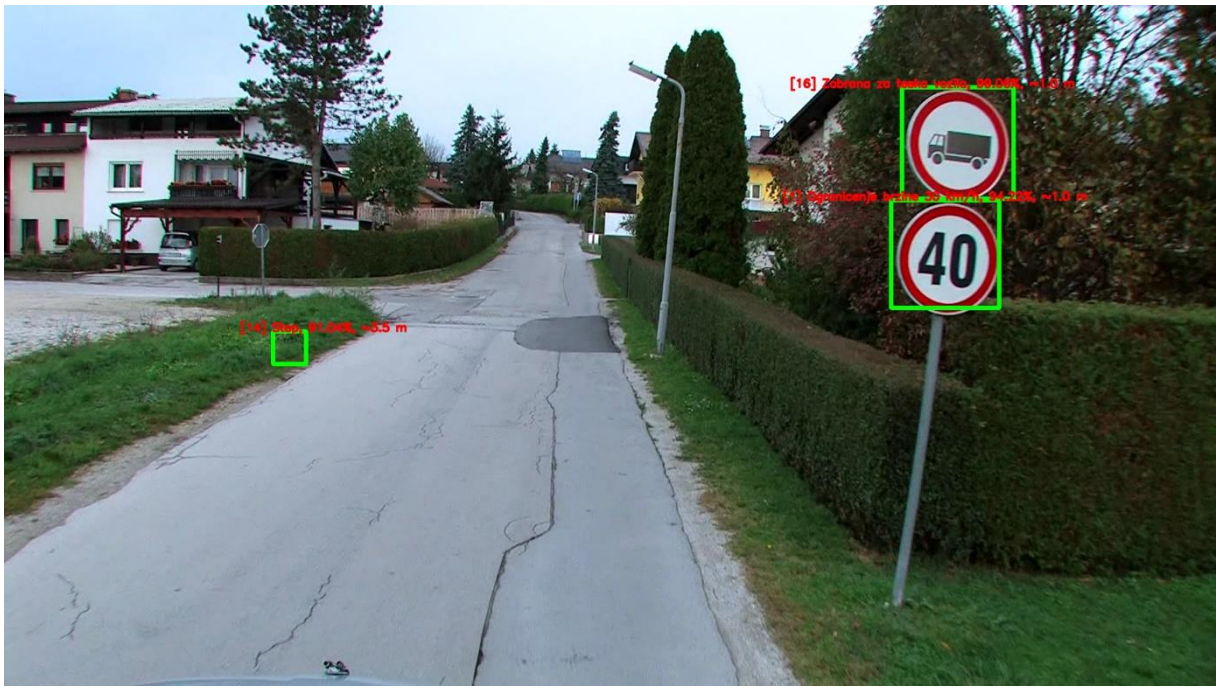


Slika 25: Primjer 2



Slika 26: Primjer 3

Na primjerima (Slika 25, Slika 26) vidljiv je jednak znak (prolaz za bicikle ili biciklistička staza) koji je u oba slučaja ispravno prepoznat. Iako se na slikama nalazi veći broj znakova različitih vrsta, program je uspješno prepoznao kako nije riječ o prometnim znakovima.



Slika 27: Primjer 4

Na prethodnom primjeru (Slika 27) vidljivo je kako je program uspješno prepoznao i klasificirao oba prometna znaka okruglog oblika, no došlo je i do greške prilikom klasifikacije znakova. Naime, model nije ispravno prepoznao kako je potencijalno područje interesa (u ovom slučaju isječak s travnjaka) pogreška, već ga je klasificirao kao znak „Stop“ uz visoku stopu vjerojatnosti. Rješenje ove krive klasifikacije je potencijalno uvesti veći broj sličnih isječaka prilikom treniranja modela u klasu koja odgovara pogreškama. Na slici koja slijedi (Slika 28) vidljivo je kako je model na sličnoj pozadini ipak u potpunosti točno prepoznao dijelove fotografije i klasificirao prometni znak.



Slika 28: Primjer 5

8. Zaključak

Izrada rješenja za detekciju i klasifikaciju prometnih znakova složen je problem koji ima potencijalno široku komercijalnu primjenu kao komponenta samovozećih automobila (eng. *self-driving car*), no s obzirom na odgovornost koja dolazi s primjenom softvera za automatizaciju vožnje potrebna je izrazita preciznost i brzina izvođenja detekcije prometnih znakova u stvarnom vremenu.

Iako je softver u potpunosti funkcionalan, razina i brzina detekcije ne zadovoljavaju smjernice koje bi garantirale da je program siguran i pouzdan u stvarnom vremenu. Takvo je rješenje i dalje predmet intenzivnog istraživanja u najvećim svjetskim automobilskim tvrtkama koje ulažu velike količine vremena i resursa u njegov pronalazak. Detekcija bi se potencijalno mogla poboljšati dodatnim eksperimentiranjem sa hiperparametrima modela i procesom augmentacije slika, te bolje balansiranim skupom za učenje.

Dodatnim radom mogla bi se izraditi aplikacija koja bi zaprimala korisnikove fotografije kroz grafičko sučelje i nad njima izvodila detekciju i klasifikaciju.

9. Prilozi

Uz rad priložene su sljedeće datoteke:

- TrafficClassifier.zip (Paket sadrži izvorni kod, pripadajući model i datoteku s oznakama, kaskadni algoritmi, kao i fotografije na kojima je model učen i nekoliko testnih fotografija. Program se pokreće pokretanjem datoteke „PrometniZnakoviTest.py“.)

10. Popis slika

Slika 1 Dijagram vrijednosti piksela 5
Slika 2: Primjer rezultata izvođenja računalnog vida 5
Slika 3 Primjer jednostavne CNN arhitekture 8
Slika 4 Tensorflow logotip 9
Slika 5: Primjer jednostavnog Keras programa 10
Slika 6: OpenCV logo 11
Slika 7: Primjer jednostavnog programa u OpenCV 12
Slika 8: Razlika zadataka klasifikacije i detekcije slike 13
Slika 9: Primjer znakova za klasu „ograničenje 20km/n“ 14
Slika 10: Primjer uparivanja slike i oznake 15
Slika 11: Distribucija fotografija po kategorijama 15
Slika 12: Vizualizacija raspodjele podataka 16
Slika 13: Prikaz funkcija preprocesuiranja fotografije 17
Slika 14: Primjer augmentiranih fotografija 18
Slika 15: Grafički prikaz modela neuralne mreže 19
Slika 16: Programski kod modela konvolucijske neuralne mreže 20
Slika 17: Primjer faze jedne epohe 21
Slika 18: Rezultati učenog modela 22
Slika 19: Prikaz rezultata na testnom skupu 22
Slika 20: Primjer izvršene detekcije HAAR kaskadnim algoritmom 25
Slika 21: Programski kod predviđanja modela 26
Slika 22: Primjer uspješne detekcije i klasifikacije fotografije 27
Slika 23: Primjer fotografije s percipiranom udaljenosti (isječak) 29
Slika 24: Primjer 1 30
Slika 25: Primjer 2 31
Slika 26: Primjer 3 31
Slika 27: Primjer 4 32
Slika 28: Primjer 5 33

11. Literatura

1. Strojno učenje kao revolucija u svijetu mobilnih uređaja
„https://medium.com/@its.mattfitzgerald/machine-learning-revolutionizing-the-mobile-app-industry-is-it-true-ed37c201cff“ – pristupljeno 12.8.2020.
2. Što je strojno učenje?
„https://www.zdnet.com/article/what-is-machine-learning-everything-you-need-to-know/“ - Pristupljeno 12.8.2020.
3. Strojno učenje
„https://en.wikipedia.org/wiki/Machine_learning“ - Pristupljeno 12.8.2020.
4. Računalni vid
„https://hr.wikipedia.org/wiki/Ra%C4%8Dunalni_vid“ – Pristupljeno 12.8.2020.
5. Računalni vid – zašto je važan
„https://www.sas.com/en_us/insights/analytics/computer-vision.html“ – Pristupljeno 12.8.2020.
6. Što je računalni vid?
„https://www.unite.ai/what-is-computer-vision/“ – Pristupljeno 12.8.2020.
7. „Grayscale“
„https://en.wikipedia.org/wiki/Grayscale“ – Pristupljeno 12.8.2020.
8. Računalni vid – sve o
„https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e“ – Pristupljeno 12.8.2020.
9. Konvolucijske neuralne mreže
„https://en.wikipedia.org/wiki/Convolutional_neural_network“ – Pristupljeno 12.8.2020.
10. Vodič prema konvolucijskim neuralnim mrežama
„https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53“ – Pristupljeno 12.8.2020.
11. ConvNet
„http://www.zemris.fer.hr/~ssegvic/du/du2convnet.pdf“ – Pristupljeno 12.8.2020.
12. Potpuno povezani slojevi
„https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/“ – Pristupljeno 12.8.2020.
13. CNN i Go
„http://proceedings.mlr.press/v37/clark15.pdf“ – Pristupljeno 12.8.2020.
14. TensorFlow
„https://en.wikipedia.org/wiki/TensorFlow“ – Pristupljeno 12.8.2020.
15. O TensorFlowu
„https://www.tensorflow.org/about“ – Pristupljeno 12.8.2020.
16. Što je TensorFlow?
„https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html“ – Pristupljeno 12.8.2020.
17. Keras
„https://en.wikipedia.org/wiki/Keras“ – Pristupljeno 12.8.2020.
18. Što je Keras?

„https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html“ – Pristupljeno 12.8.2020.

19. OpenCV

„https://opencv.org/“ – Pristupljeno 12.8.2020.

20. O knjižnici OpenCV

„https://opencv.org/about/“ – Pristupljeno 12.8.2020.

21. Trening, validacija i testiranje

„https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7“ – Pristupljeno 12.8.2020.

22. „https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets“ – Pristupljeno 12.8.2020.

23. Hiperparametri

„https://datascience.stackexchange.com/questions/14187/what-is-the-difference-between-model-hyperparameters-and-model-parameters“ - Pristupljeno 12.8.2020.

24. Augmentacija podataka

„https://www.kdnuggets.com/2018/09/data-augmentation-bounding-boxes-image-transforms.html“ – Pristupljeno 12.8.2020.

25. Augmentacija podataka – točnost

„https://arxiv.org/abs/1906.11172“ – Pristupljeno 12.8.2020.

26. Konvolucijske neuralne mreže

„https://cs231n.github.io/convolutional-networks/“ – Pristupljeno 12.8.2020.

27. Keras i CNN

„https://victorzhou.com/blog/keras-cnn-tutorial/“ – Pristupljeno 12.8.2020.

28. Što je grupa podataka?

„https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network“ – Pristupljeno 12.8.2020.

29. Što je korak?

„https://tolotra.com/2018/07/25/what-is-the-difference-between-step-batch-size-epoch-iteration-machine-learning-terminology/“ – Pristupljeno 12.8.2020.

30. Haar kaskadni algoritam

„http://www.willberger.org/cascade-haar-explained/“ – Pristupljeno 12.8.2020.

31. OpenCV kaskadni algoritmi

„https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html“ – Pristupljeno 12.8.2020.

32. Trijangulacija udaljenosti u OpenCV

„https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/“ – Pristupljeno 12.8.2020.

33. Udaljenost objekata od leće

„https://photo.stackexchange.com/questions/12434/how-do-i-calculate-the-distance-of-an-object-in-a-photo“ – Pristupljeno 12.8.2020.

34. Specifikacije auto kamera

https://www.cnet.com/products/garmin-dash-cam-20/specs/ - Pristupljeno 12.8.2020.

35. Konvolucija

„https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1“ – „ - Pristupljeno 12.8.2020.

36. Konvolucijski slojevi
„https://www.sciencedirect.com/topics/computer-science/convolution-layer“ - Pristupljeno 12.8.2020.
37. Klasifikacije slike
„https://desktop.arcgis.com/en/arcmap/latest/extensions/spatial-analyst/image-classification/what-is-image-classification-.htm“ - Pristupljeno 25.8.2020.
38. Detekcija slike
„https://sightcorp.com/knowledge-base/image-recognition/“ - Pristupljeno 25.8.2020.
39. Caffe
„https://caffe.berkeleyvision.org/“ - Pristupljeno 25.8.2020.
40. Dlib
„http://dlib.net/“ - Pristupljeno 25.8.2020.
41. PyTorch
„https://pytorch.org/“ - Pristupljeno 25.8.2020.
42. Svojstvo nepromjenjivosti
„https://stats.stackexchange.com/questions/208936/what-is-translation-invariance-in-computer-vision-and-convolutional-neural-netwo“ - Pristupljeno 25.8.2020.
43. Backpropagation
„https://en.wikipedia.org/wiki/Backpropagation“ - Pristupljeno 25.8.2020.
44. Točnost
„https://developers.google.com/machine-learning/crash-course/classification/accuracy“ – Pristupljeno 26.8.2020.
45. Kako rade konvolucijske neuralne mreže
„https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/“ - Pristupljeno 26.8.2020.
46. Duboke neuralne mreže
„https://bernardmarr.com/default.asp?contentID=1789“ – Pristupljeno 26.8.2020.