

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6391

**Detekcija i klasifikacija tekstovnih
elemenata na slici koristeći duboke
neuronske mreže**

Lukas Šestić

Zagreb, svibanj 2019.

*Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu \izvornik.*

Zahvaljujem se prof. dr. sc. Domagoju Jakoboviću na pruženoj pomoći i sredstvima danima u svrhu uspješne izrade završnog rada.

Također se zahvaljujem svojim roditeljima na prilici za studiranje i stalnoj podršci.

SADRŽAJ

1. Uvod	1
1.1. Uvod	1
1.2. Računalni vid	1
1.2.1. Pregled	1
1.2.2. Konvolucijske neuronske mreže	2
2. Generiranje seta podataka	4
2.1. Značaj podataka u dubokom učenju	4
2.2. Generiranje slika	5
2.2.1. Generalizacija postupka	5
2.2.2. Prikupljanje fontova	6
2.2.3. Generiranje simbola	7
2.2.4. Transformacije	7
2.2.5. Kreiranje cjelovitih slika	10
2.3. TFRecords	11
3. Single Shot MultiBox Detector mreža	13
3.1. Zašto je nastala SSD mreža	13
3.2. Arhitektura	14
3.2.1. VGG-16 arhitektura	14
3.2.2. SSD arhitektura	15
4. Trening	16
4.1. Treniranje SSD neuronske mreže	16
4.1.1. Visoki pogled na treniranje	16
4.1.2. Određivanje pozicije objekata	16
4.1.3. Određivanje parametara za nastavak treniranja	17

5. Rezultati	19
5.1. Opis rezultata	19
5.2. Prikaz rezultata	19
5.3. <i>Tensorboard</i> rezultati	21
Literatura	23

1. Uvod

1.1. Uvod

Računalna moć uređaja koje gotovo neprestano nosimo sa sobom kao što su pametni telefoni i prijenosna računala je unazad deset godina eksponencijalno narasla. Naravno, sa modernim alatima dolaze i moderni problemi.

Jedan najraširenijih alata, koji se sve više i više koristi za rješavanje problema koji su do nedavno bili nemogući, ili izrazito algoritamski komplikirani za riješiti je *umjetna inteligencija*. Umjetna inteligencija podrazumjeva skup načina i metoda koje računalu opisuju početno i konačno stanje do kojeg mora doći sam.

Ovaj rad, bavit će se najkorištenijom metodom umjetne inteligencije, *dubokim učenjem*, i njegovim podskupom *računalnim vidom*. Kroz rad i programsku implementaciju, priхватiti ću se problema detekcije napisanog teksta na slici i daljnom obradom istog.

Detaljno ću kroz poglavlja obraditi postupke koje sam primjenio za generiranje raznolikih slika koje imitiraju rukopis i proces potreban da računalo nauči prepoznavati isti na slici.

Na kraju, izlučeni tekst sa slike, biti će moguće obraditi na željeni način. Način koji ću ja predstaviti biti će primjena jednostavne matematike, slično onome što pruža *Photomath, Inc.*. Na primjer, za sliku na kojoj je napisan tekst "2 + 2", izlaz će biti slika sa kvadratima oko prepoznatih simbola, i rješenje obrađenog teksta, u ovom slučaju "4".

1.2. Računalni vid

1.2.1. Pregled

Na najvišoj razini, *računalni vid* su metode koje koje računalima daju mogučnost razumjevanja slike na visokoj razini, najčešće s ciljem automatiziranja ljudskih

poslova. Osnovni zadatak je raspoznavanje veze između obrazaca na slici i rješenja na problem koji želi riješiti. Svi procesi koji koriste strojno učenje, u konačnici se svode na detekciju i klasifikaciju elemenata na slici. Metode računalnogvida temelje se na geometriji, statistici, fizici i teoriji učenja.

Danas, se velika količina problema rješava uz pomoć računalnogvida, često da ljudi za to nisu ni svjesni:

- Prepoznavanje znakova (Slika 1.1)
- Prepoznavanje lica
- Kompresija i restauracija slike
- Prepoznavanje elemenata na slici
- Analiza medicinskih snimki u svrhu detaljnije analize
- Itd.



Slika 1.1: Maskiranje elemenata na slici prometa

1.2.2. Konvolucijske neuronske mreže

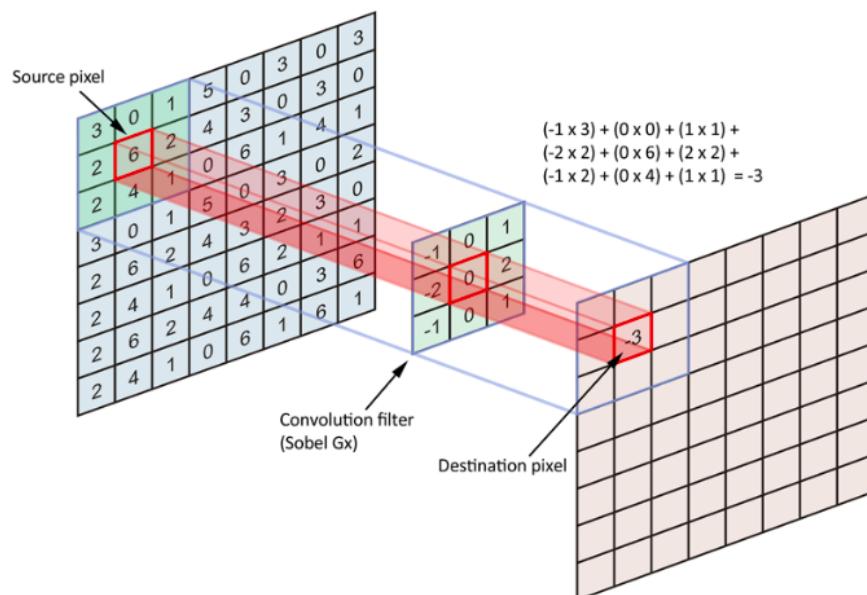
Konvolucijske neuronske mreže danas se koriste kao najefektivniji način postizanja računalnogvida. Glavna prednost nad potpuno povezanim neuronskim mrežama je manji broj *težina* za treniranje što znatno ubrzava treniranje. Ipak, ono što je možda najvažnije za napomenuti je to što pozicija traženog elementa na slici *konvolucijskoj neuronskoj mreži* ne igra ulogu.

Svaki sloj *duboke konvolucijske neuronske mreže* funkcioniра kao filter koji se kreće po slici, pamteći što ga je najviše aktiviralo. Najčešće se koristi filter veličine 3×3 . (Slika 1.2)

Dalje, uz konvolucijski sloj, nerijetko se postavlja *max pooling sloj*. Na apstraktnoj razini, princip rada max pooling sloja je sljedeći: Ako uzmemo veličinu pooling filtra kao onu koja se najčešće koristi, to jest 2×2 , on izlaz iz prethodnog sloja raspodjeli na kvadrate iste veličine. Zatim, filter se postavi između 4 kvadrata i sebi za vrijednost stavi najveću iz svakog u pripadajuće polje.

Prirodno je pitati se zašto se to koristi i zašto to radi.

Pooling filter jednostavno smanjuje "rezoluciju" prethodnog sloja, ne mjenjajući važne čimbenike potrebne za daljnji rad mreže. Na primjer, vertikalna linija, krug, ili elipsa, ostaje ono što je, jedino manje razlučivo. Bitno je napomenuti da smanjivanjem rezolucije dobivamo puno manje parametara za treniranje. Stavimo to u brojeve. Slike unutar *mnist* seta podataka su veličine 28×28 . To znači da bi se treniralo 28×28 parametara. Primjenom *Max pooling sloja* veličine 2×2 , treniralo bi se $\frac{1}{4} \times (28 \times 28)$ parametara.



Slika 1.2: Klizeći konvolucijski filter

Spomenute prednosti, referenciraju se na glavnu značajku *konvolucijskih mreža*. Cilj je ići dublje, ne šire. Za sliku veličine 100×100 , potpuno povezanoj neuronskoj mreži u prvom sloju treba 10 000 čvorova, svaki sa svojim parametrom za treniranje, dok konvolucijskoj to ne treba.

Svaki sljedeći sloj ima drugu ulogu. Prvi najčešće ima ulogu raspoznavanja najosnovnijih elemenata slike kao što su različiti rubovi, dok sve dublji koriste podatke od prošlih i osnovne elemente grupiraju u apstraktne strukture koji predstavljaju značajnije elemente slike (O'Shea i Nash (2015)).

2. Generiranje seta podataka

2.1. Značaj podataka u dubokom učenju

Prvi i najdulji praktični korak treninga predstavlja priprema podataka. Sve ovisi o zadatku koji mreža mora riješiti, ali, generalno je pravilo da je više podataka bolje. Konačna kvaliteta rješenja osim o arhitekturi mreže koju dizajniramo, ovisi o kvaliteti podataka kojom ju usmjeravamo. Priprema podataka vrši se u 3 glavna koraka (Gonfalonieri (2019)):

1. Prikupljanje
2. Klasifikacija
3. Označavanje

Prikupljanje podataka

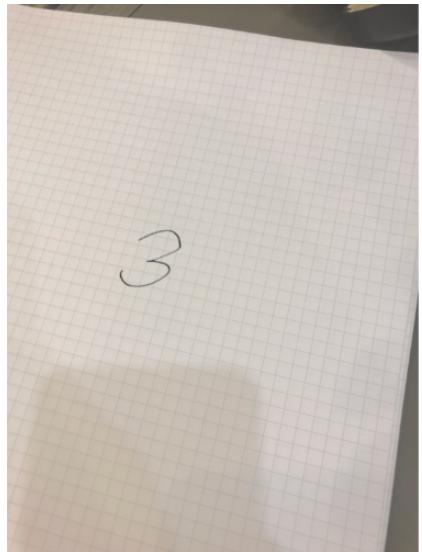
Prikupljanje podataka mora biti sustavan i smislen proces jer može otežati i olakšati daljnje korake. Najpreporučeniji način za prikupljanje je dugoročno i postepeno spremanje podataka jer rezultira velikim brojem objektivnih i kvalitetnih podataka. Ja sam se ipak odlučio na metodu računalnog generiranja vlastitog seta podataka. Razlog tome je raznolikost elemenata koje mreža mora moći detektirati i fleksibilnost koju dobivam jednomo kada ustanovim sve potrebe.

Klasifikacija i označavanje podataka

Generirani podaci na određeni način moraju biti prikazani mreži. Iako u mrežu slika ulazi kao vektor dimenzija (**visina x širina x kanali**) mreži su potrebni i podaci za uspoređivanje rezultata i računanje uspješnosti. U ovom radu koristio sa **.csv** datoteku za dohvaćanje i kao opisnik slika. Postupak automatskog generiranja slika uvelike je olakšao klasifikaciju i označavanje jer je cijeli postupak ostvaren kao "cjevod". Pri izlasku, slika bi bila prikazana kao na slici 2.1.

Datoteka bi upisano imala ime slike, simbol na slici, širinu, visinu i točan položaj elementa na slici. Prednost ovog pristupa je i u tom što slika nije zadana absolutnom putanjom, što znači, da sam slike mogao kreirati na vlastitom računalu, prenjeti ih na udaljeni server za treniranje i bez komplikacija koristiti iste.

Veličina opisnika je također bila zanemariva. Nakon raspodjеле 80:20 za trening i validaciju na 15 000 slika, veličine su bile 440kB i 110kB dok je direktorij sa slikama bio veličine 6,7GB.



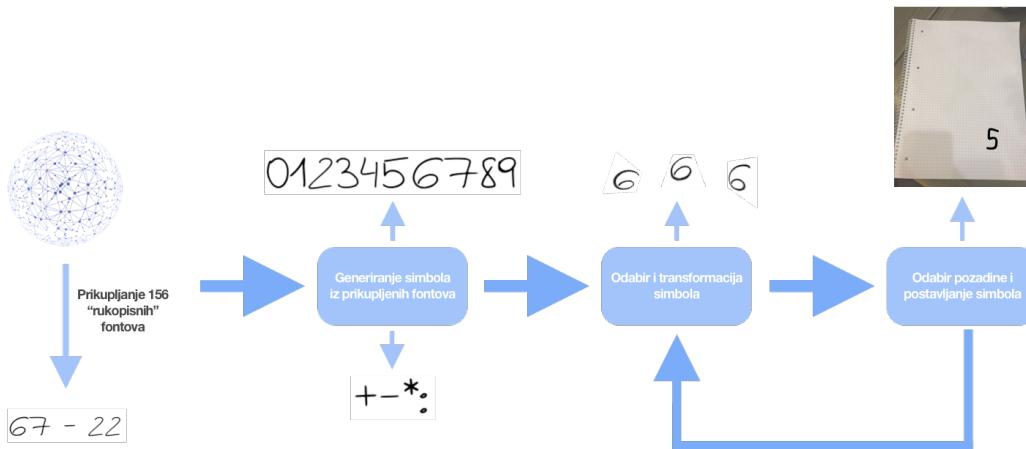
filename	class	imwidth	imheight	xmin	xmax	ymin	ymax
eqjodd.png	:	540	720	175	189	263	361
arnqxq.png	:	540	720	254	277	510	624
tvxdvf.png	*	540	720	394	455	579	669
fapsod.png	:	540	720	32	67	104	243
qardct.png	-	540	720	257	292	246	375
thzsfh.png	-	540	720	211	253	149	212
vdtpni.png	3	540	720	34	157	311	436
uyekux.png	3	540	720	116	169	542	636
iyjrea.png	4	540	720	142	300	131	206
uqqaqt.png	0	540	720	250	340	444	580
khwwh.png	0	540	720	280	384	121	332
yqumug.png	5	540	720	349	428	454	586

Slika 2.1: Slika i pripadajuća referenca u .csv datoteci

2.2. Generiranje slika

2.2.1. Generalizacija postupka

Za relativan uspjeh treniranja mreže za detekciju i klasifikaciju 14 tekstovnih elemenata (0-9, +, -, *, :) potrebno je minimalno 10 000 slika. Ne samo zbog broja elemenata već i zbog složenosti i raznolikosti između njih. Postupak koji sam razvio primjenjuje sve taktike (Chollet (2017)) potrebne za stvaranje raznovrsnog i kvalitetnog seta podataka. Zbog transformacija, opisanih u dalnjim djelovima poglavlja, gotovo je nemoguće da iako se isti font stavlja na pozadinu, nastane isti oblik. Na slici 2.2 prikazana je topologija cjevovoda koja kreira slike. Cijeli cjevovod, implementiran je unutar programskog paketa *ImageGenerator*, kojeg sam napisao u svrhu apstraktiranja cijelog postupka.



Slika 2.2: Prikaz visoke razine cjevovoda za generiranje slika

2.2.2. Prikupljanje fontova

Ispisivanje velikog broja simbola sa razlikom između varijacija istog monoton je i neisplativ posao, posebice zbog dostupnosti svih potrebnih resursa na internetu. U prilog je također išlo to što su dostupni fontovi, koji primjenjuju rukopisni stil, najčešće zbilja napisani rukom i vektorizirani, pa, generiranje i transformiranje neće negativno utjecati na kvalitetu. Osim rukopisnih fontova, skinuo sam mali broj fontova koji su stilski između čistog rukopisnog i tipkanog. Fontovi su bili prikupljeni sa sljedećih izvora, a na slici 2.3 vidljivi su primjeri istih:

- <https://www.dafont.com>
- <https://www.1001fonts.com>
- <https://www.1001freefonts.com>



Slika 2.3: Varijacije unutar simbola uzrokovane fontovima

2.2.3. Generiranje simbola

Nakon prikupljanja i sortiranja fontova, generiranje samih simbola bio je jednostavan posao. Važno je bilo očuvati transparentnost pozadine iza simbola jer u trenutku kada se postavi na pozadinu po izboru, ona mora biti vidljiva.

Algorithm 1 Generiraj sve simbole

```
1: function GENERIRAJTEXTSLIKU(font, simbol)
2:   velicina  $\leftarrow$  font.velicina(simbol)
3:   slika  $\leftarrow$  Image('RGBA', velicina, (255, 255, 255, 0))
4:   slika.text = simbol
5:   return slika
6: end function
7: function GENERIRAJSVESIMBOLE()
8:   for simbol in simboli do
9:     for font in fontovi do
10:      direktorijSimbola  $\leftarrow$  spoji(putDoSimbola, simbol)
11:      if direktorijSimbola ne postoji then
12:        KREIRAJDIREKTORIJ(direktorijSimbola)
13:      end if
14:      generiranaslika  $\leftarrow$  GENERIRAJTEXTSLIKU(font, simbol)
15:      SPREMI $\text{SLIKU}$ (generiranaSlika)
16:    end for
17:  end for
18: end function
```

2.2.4. Transformacije

Prije postavljanja simbola na nasumično odabranu sliku, svaki simbol prošao je kroz tri točke transformiranja:

1. Skaliranje
2. Rotacija
3. Afina transformacija

Cilj transformacija je maksimalno unjeti raznolikost unutar dataseta u slučaju premalog ili presličnog broja slika. Klasa *ImageGenerator* za to se brine na sličan način kao programski paket *Keras.preprocessing.image.ImageDataGenerator*

(Ker). Transformaciju nad slikama vršio sam pomoću programskog paketa *OpenCV* (Ope) jer apstraktira potrebne matematičke operacije na razumljiv, lako koristiv i prilagodljiv način. Tijekom faze transformiranja i postavljanja slike na pozadinu, one su u obliku matrice, definirane pomoću programskog paketa *Numpy*.

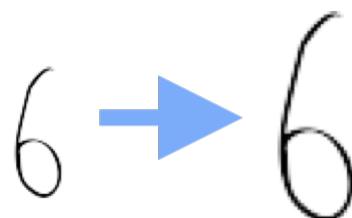
Skaliranje

Skaliranje pomoću *OpenCV* paketa može se vršiti ili ručno, specifirajući točnu veličinu, ili dajući faktor skaliranja. *OpenCV* također automatski primjenjuje *interpolaciju* kako bi se kvalitete maksimalno sačuvala. Skaliranje se vrši na način da se matrica slike pomnoži sa matricom skaliranja, zadanom na sljedeći način:

$$M = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix},$$

gdje je s_x faktor skaliranja u x dimenziji, a s_y faktor skaliranja u y dimenziji. Rezultat skaliranja vidljiv je na slici 2.4

```
1 image = cv2.imread(image_path, -1)
2 # Ne zelim umanjiti sliku
3 s_x = np.random.rand() + 1
4 s_y = np.random.rand() + 1
5 image = cv2.resize(image, fx=s_x, fy=s_y)
```



Slika 2.4: Rezultat primjene skaliranja sa $s_x = s_y = 1.25$

Rotacija

Rotacija slike, za kut θ ostvaruje se množenjem sa matricom rotacije:

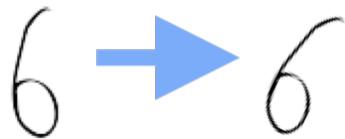
$$M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Iako sam ja rotiranje vršio iz središnje točke, *OpenCV* nudi podršku za eksplisitno zadavanje točke oko koje će se rotacija vršiti. Rezultat rotiranja vidljiv je na slici 2.5

```

1 image = cv2.imread(image_path, -1)
2 # Prevelika rotacija bi mogla izazvati dvosmislenost
3 theta = np.random.randint(-45, high=45)
4 rows, cols = image.shape
5 M = cv2.getRotationMatrix2D((cols / 2, rows / 2), theta, 1)
6 image = cv2.warpAffine(image, M, (cols, rows))

```



Slika 2.5: Rezultat primjene rotacije sa $\theta = 25$

Afne transformacije

Afne transformacije koristim za prividno transformiranje simbola "u prostoru", bez velikog rizika od prevelike distorzije slike jer sve paralelne linije, nakon transformacije ostaju paralelne. *OpenCV* Afinu transformaciju vrši tako da tri odabранe točke na slici pomakne za određeni koeficijent. Kao i ostale transformacije, matematički nastaje množenjem matrice slike s matricom afne transformacije oblika.

$$\begin{bmatrix} 1 & \tan(\beta) \\ \tan(\alpha) & 1 \end{bmatrix}$$

gdje su α i β razlike u kutevima prema pripadajućim koordinatnim osima. Rezultat primjene afne transformacije na simbolu vidljiv je na slici 2.6

```

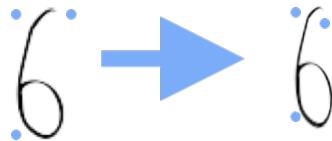
1 image = cv2.imread(image_path, -1)
2 height, width = image.shape
3 # Tocke u pripadajućim uglovima
4 affine_ref_points_1 = np.float32([[0, 0], [s_height, 0],
5 [0, s_width]])
6

```

```

7 t1_delta = [random_float(0, height / 3),
8         random_float(0, width / 3)]
9 t2_delta = [random_float(2 * height / 3, height),
10        random_float(0, width / 3)]
11 t3_delta = [random_float(0, height / 3),
12        random_float(2 * width / 3, width)]
13
14 affine_ref_points_2 = [t1_delta, t2_delta, t3_delta]
15
16 M = cv2.getAffineTransform(affine_ref_points_1,
17                         affine_ref_points_2)
18
19 image = cv2.warpAffine(image, M, (width, height))

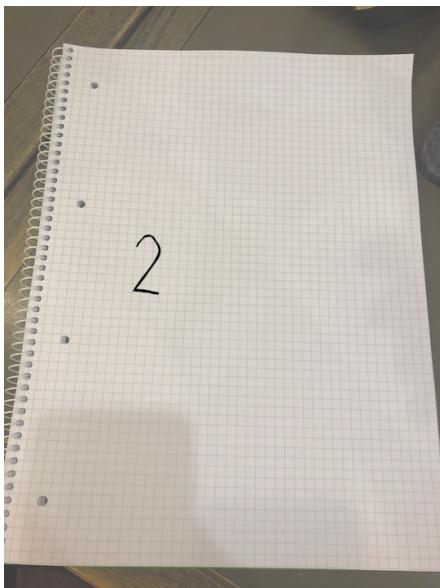
```



Slika 2.6: Rezultat primjene afine transformacije s pripadajućim referentnim točkama

2.2.5. Kreiranje cjelovitih slika

Kreiranje cjelovitih slika svodilo se na postavljanje generiranih i transformiranih simbola na pozadinu po izboru. Pozadina također igra veliku ulogu u prepoznavanju jer mreža pregledava cijelu sliku. Za svoje potrebe odlučio sam za pozadinu koristiti slike matematičkih bilježnica uz pretpostavku da bi se iste koristile najviše kada bi se ova mreža koristila u stvarnom svijetu. Izlazi iz mreže vidljivi su na slikama 2.7a i 2.7b. Slike su spremljene u direktorij **Images**, a **.csv** opisnik u direktorij **Data** odakle će se dalje referencirati za kreiranje **.record** datoteke za daljnje korištenje *Tensorflow-u*.



(a) Izlazna slika iz *ImageGenerator-a*

```
Smaller images: ['6/Easy/Symbol_6_86.png'] ; Background: /Users/lukassestic/MachineLearning/Završni Rad/Backgrounds/A4_math_5.png
Smaller images: ['5/Easy/Symbol_5_69.png'] ; Background: /Users/lukassestic/MachineLearning/Završni Rad/Backgrounds/A4_math_3.png
Smaller images: ['-/Easy/Symbol_-98.png'] ; Background: /Users/lukassestic/MachineLearning/Završni Rad/Backgrounds/A4_math_6.png
Smaller images: ['*/Easy/Symbol_*95.png'] ; Background: /Users/lukassestic/MachineLearning/Završni Rad/Backgrounds/A4_math_5.png
Smaller images: ['7/Easy/Symbol_7_139.png'] ; Background: /Users/lukassestic/MachineLearning/Završni Rad/Backgrounds/A4_math_6.png
Smaller images: ['2/Easy/Symbol_2_25.png'] ; Background: /Users/lukassestic/MachineLearning/Završni Rad/Backgrounds/A4_math_1.png
```

(b) Ispis za praćenje statusa generiranja slika

2.3. TFRecords

Zašto je bolje za mrežu da podatke čita iz `.record` datoteke nego odvojeno slike i pripadajuće opise? Zamislimo sljedeći scenarij. Treniranje se vrši na računalu sa HDD diskom, slike i oznake su u različitim direktorijima. Svako čitanje sljedeće slike i oznake rezultira potencijalnim pomicanjem glave diska. Cilj je da sve potrebne datoteke budu što bolje poravnate u memoriji. Tu se pokazuje najveći značaj *TFRecords* datoteke. Jedna binarna datoteka koja sadrži sve informacije za mrežu, jedinstveno poravnata u memoriji (TFR).

U pozadini, *TFRecords* je format koji koristi *Protocol buffer* tehnologiju. *Protocol buffer* ili kraće *Protobuf* je knjižnica za efikasnu serijalizaciju strukturiranih podataka (ten). Konkretno, koristimo *Protobuf* poruke oblika "`string` : value" za predstavljanje objekata mreži. U mom slučaju, slike su zapisane na sljedeći način:

- height = `int64`
- width = `int64`
- filename = `bytes`
- sourceid = `bytes`

- encoded = `bytes`
- format = `bytes`
- xmins = `float_list`
- xmaxs = `float_list`
- ymins = `float_list`
- ymaxs = `float_list`
- classes_text = `bytes_list`
- classes = `int64_list`

Svi navedeni podaci zapisuju se pod `feature` ključ.

Kako u našem slučaju vršimo detekciju i klasifikaciju objekata, bitno je da na neki način i klasama damo jedinstveni identifikator. Naime, u *TFRecords* datoteku pod ključ *classes* koji sadrži podatke o tom koji su svi objekti na slici ne pišemo doslovno ime objekta (npr. Automobil, kuća, ...). Pišemo brojčanu vrijednost istog objekta koja ga predstavlja. Isti način je precizniji i sažetiji.

Konkretno ime dnevnika koji sadrži mapiranja iz objekta u njegovu brojčanu vrijednost naziva se *Label map* i osim za stvaranje *TFRecords* datoteke, koristi ga i sama mreža i mi kad iz mreže čitamo što je ista prepoznala. Za kreiranje datoteke pratilo sam korake opisane na službenoj *Tensorflow* stranici.

3. Single Shot MultiBox Detector mreža

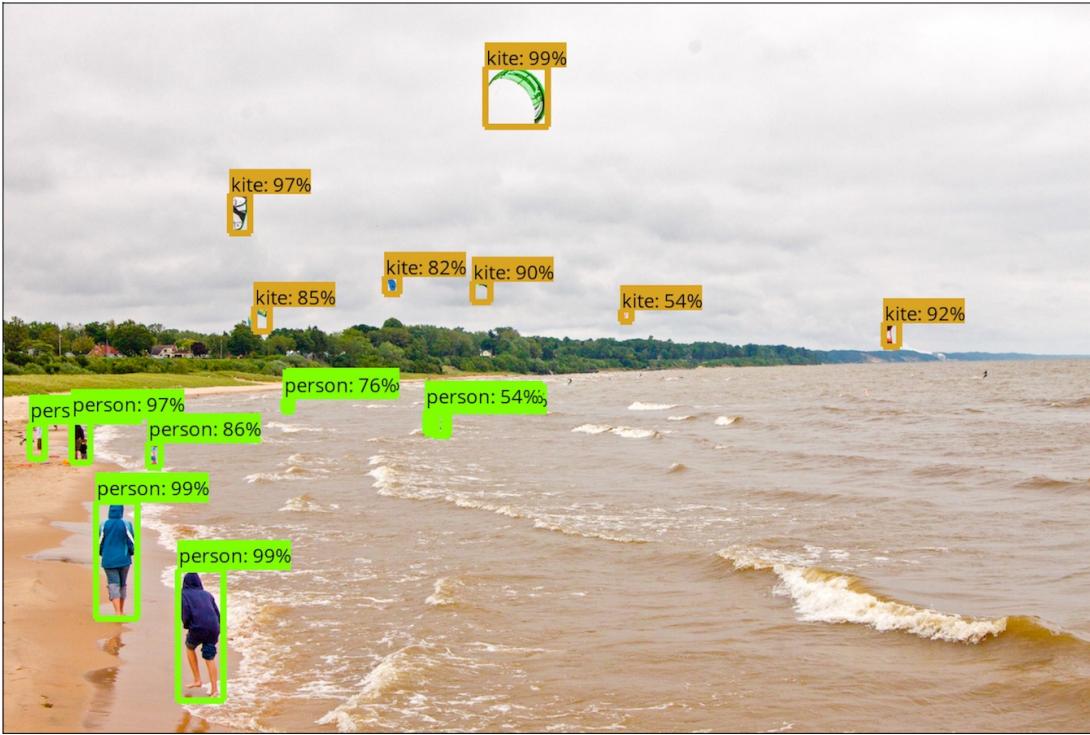
3.1. Zašto je nastala SSD mreža

Single Shot MultiBox Detector (dalje *SSD*) je mreža za detekciju i klasifikaciju kojoj je primarna svrha jednostavnost i brzina. Prije nastanka *SSD* mreže, najpoznatije mreže za isti zadatak bile su implementirane arhitekturom *Region-Convolutional Neural Network* (dalje *R-CNN*). *R-CNN* mreže na izlazu tipično daju set kvadrata koje opisuju objekt i klasu istog. Klasični izlaz iz *R-CNN* mreže vidljiv je na slici 3.1. Osim bazičnog, iz *R-CNN*-a nastale su i mreže *Fast(er)-R-CNN* koje dalje ubrzavaju i poboljšavaju preciznost iste arhitekture (Ren et al. (2015)). No, nijedna od tih nije uspjela doseći gotovo "real-time" brzinu sa značajnom preciznosću.

Iako su spomenute mreže imale pokazivale impresivne rezultate, također su imale i nekolicinu problema:

- Više faza treniranja
- Komplicirana mreža
- Mreža spora za stvarno korištenje

Zbog spomenutih problema, nastale su nove arhitekture od kojih je jedna i *SSD*, koju ovaj rad koristi cijelokupnu implementaciju zadatka detekcije i klasifikacije 14 različitih klasa.



Slika 3.1: Tipični izlaz iz mreže za detekciju i klasifikaciju sa prikazanim kvadratima i klasama

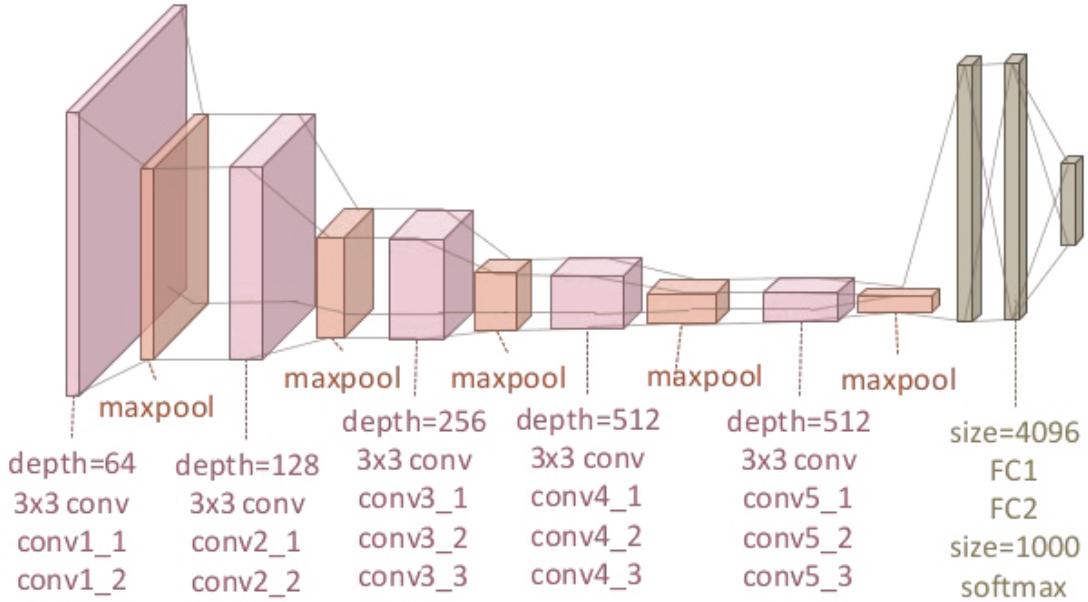
3.2. Arhitektura

3.2.1. VGG-16 arhitektura

VGG-16 je poznata neuronska mreža nastala na Oxfordu od strane *Visual Geometry Group-a*, odakle i potjeće ime. Mreža sama po sebi ostvaruje odlične rezultate na *ImageNet* datasetu no to nije jedini razlog zašto je jedna od najkoristenijih.

VGG mreža razvijena je da bude jednostavna, sadržavajući samo 3×3 konvolucijske i 2×2 pooling slojeve prije završnih gusto spojenih slojeva (Simonyan i Zisserman (2014)). Arhitektura mreže vidljiva je na slici 3.2. Također, cijela struktura, težine i cijela istrenirana mreža je dostupna besplatno na internetu na službenoj stranici projekta (http://www.robots.ox.ac.uk/~vgg/research/very_deep/). Mana i prednost *VGG-16* arhitekture je što je prostorno velika. Oko 60MB u svojoj cjelini sa čak 160M parametara za treniranje što je odlična stvar za ponovno korištenje mreže za druge primjene.

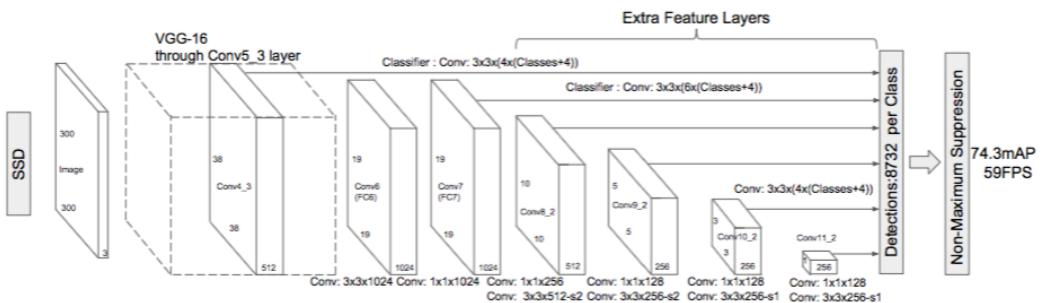
Jedna od tih primjena je *SSD* mreža, koja na svom početku sadrži baš *VGG-16* arhitekturu, sve do gusto spojenih slojeva koje odbacuje.



Slika 3.2: Arhitektura VGG-16 mreže

3.2.2. SSD arhitektura

Razlog zbog kojeg *SSD* mreža koristi *VGG-16* kao baznu mrežu je njezina snažna performansa na slikama visoke kvalitete i popularnost gdje tehnika *transfer learning*-a pomaže pri dobrom rezultatima. Umjesto gusto spojenih slojeva *SSD* mreža dodaje još konvolucijskih slojeva koji dalje izvlače značajke i progresivno smanjuju ulaz svakom dubljem sloju (Liu et al. (2016)). Cijela arhitektura *SSD* mreže vidljiva je na slici 3.3 Slojevi dodani na baznu mrežu dodani su s ciljem



Slika 3.3: Arhitektura SSD mreže

da proizvedu sljedeće značajke:

4. Trenin

4.1. Treniranje SSD neuronske mreže

4.1.1. Visoki pogled na treniranje

Bitna razlika u treniranju *SSD* mreže i tipične *R-CNN* mreže slične zadaće je ta da "ground truth" podatak mora biti dodjeljen točnom izlazu iz fiksнog skupa izlaza detektora (Liu et al. (2016)). Na sličan način radi i veliko poboljšanje na *R-CNN* arhitekturu, *Faster R-CNN*.

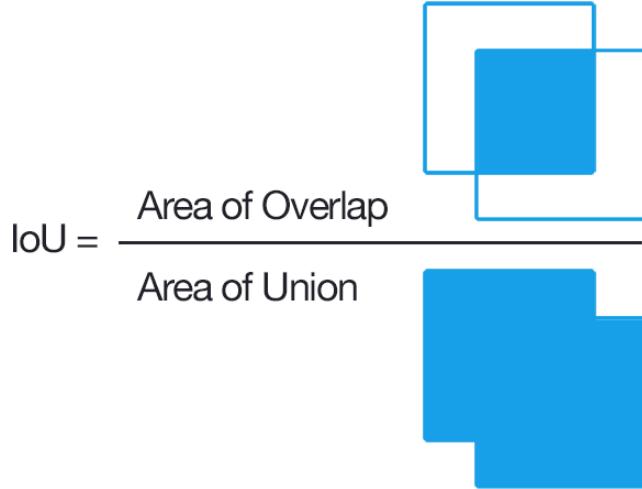
Kao i kod klasičnih neuronskih mreža, primjenjuje se funkcija gubitka, a za određivanja težina koristi se *back propagation* od kraja do kraja. Prije početka treniranja također se određuju prepostavljeni kvadrati, različite skale za detekciju i strategije za povećanje podataka.

O prepostavljenim kvadratima, skalama za detekciju i strategijama pisati ћu u nastavku.

4.1.2. Određivanje pozicije objekata

Tijekom treninga, cilj je odrediti koji prepostavljeni kvadrati najbolje odgovaraju "ground truth" kvadratima objekta, to jest onima specifiranim u dataset-u. Nakon što se odrede najprecizniji kvadrati, mreža se sukladno tome dalje prilagođava. Za svaki "ground truth" kvadrat imamo na izbor više prepostavljenih kvadrata, različitih lokacija, skala i omjera. Želimo naći onaj koji ima najveći *jaccardov index preklapanja* (dalje *IoU*) (slika 4.1).

U konfiguracijskoj datoteci koja će biti priložena na kraju rada, možemo ručno odrediti od koje točke prepostavljeni kvadrat prihvачamo. Prepostavljena vrijednost je da mora vrijediti $IoU \geq 0.5$. To višestruko olakšava treniranje jer mreža zadržava više prepostavljenih kvadrata umjesto da mora odabrati samo onaj sa najvećim preklapanjem.



Slika 4.1: Način na koji računamo jaccardov index preklapanja, tj. IoU

4.1.3. Određivanje parametara za nastavak treniranja

Određivanje parametara bilo bi puno lakše kada bismo imali samo jedan objekt za klasificirati, no postaje komplikiranije sa više objekata. Uzmimo $x_{ij}^p = \{1, 0\}$ kao indikator za podudaranje i -og prepostavljenog kvadrata na j -ti "ground truth" kvadrat kategorije p . Koristeći spomenutu strategiju određivanja pozicije objekata može nam se dogoditi situacija $\sum_i x_{ij}^p \geq 1$.

Ukupna funkcija gubitka računa se kao otežana suma lokalacijskog (loc) i klasifikacijskog ($conf$) gubitka:

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

N nam predstavlja broj "pogođenih" prepostavljenih kvadrata. Naravno, ako je $N = 0$, postavimo da je gubitak također = 0.

Lokalacijski gubitak (loc)

Za izračun lokalacijskog gubitka koristimo *Smooth L1* između predviđenih i "ground truth" kvadrata.

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^N \sum_{m \in (cx, cy, w, h)} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

Klasifikacijski gubitak ($conf$)

Klasifikacijski gubitak računa se kao *softmax* svih klasa koje podržavamo.

$$L_{conf}(x, c) = - \sum_{i \in Pos} x_{ij}^p \log(\hat{c}_i^0) - \sum_{i \in Neg} \log(\hat{c}_i^0) \text{ gdje } \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

5. Rezultati

5.1. Opis rezultata

Mreža se kroz vrijeme ponašala iznimno zanimljivo. Već u prvih par koraka vrijednost ukupne funkcije gubitka brzo se spustila sa ≥ 50 na ≤ 10 . No nevjerojatno dugo joj je trebalo da se spusti na vrijednost ≤ 4 . Razlog tome je pretpostavljam određeni broj *false positive*-a (Slika 5.1a) unutar dataseta. Naime, *false positive* slike nastale su primjenom transformacija nasumično generiranim parametrima koji su se našli na rubu prihvatljivih vrijednosti. Nažalost, zbog velikog broja slika (Nakon svih generiranja, pokušaja i treniranja ≥ 20000) nisam ručno mogao izbaciti sve. Ipak, smatram da je određeni mali broj takvih slika neophodan za uspješno treniranje slike.

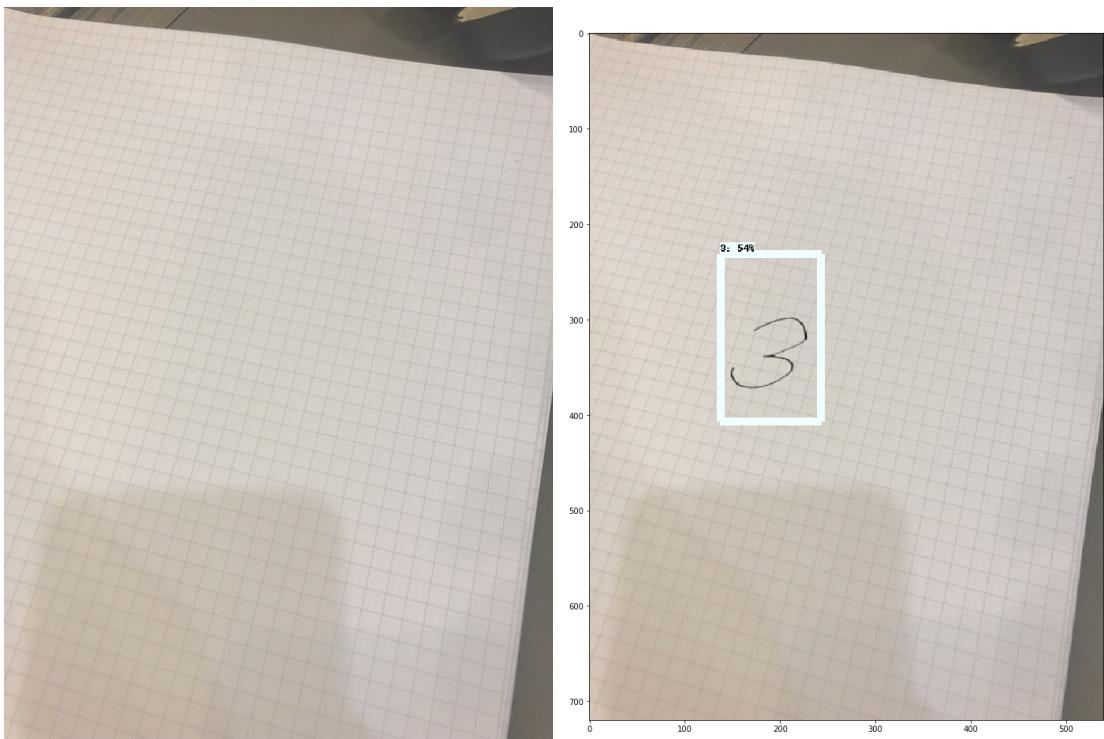
Nakon ukupno 500000 koraka, tj. $brojKoraka \times \frac{batchSlika}{brojSlika} = 500000 \times \frac{24}{15000} = 80$ epoha, ukupni gubitak iznosio je 3.879 gdje je po izgledu dosegao asimptotu.

5.2. Prikaz rezultata

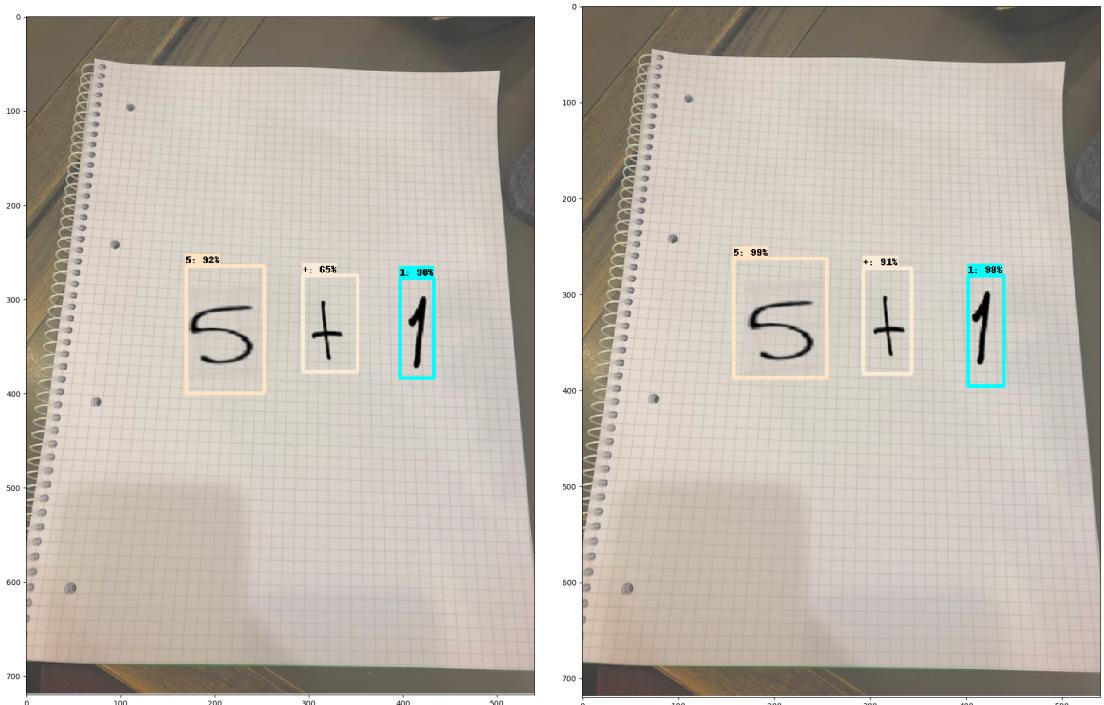
Na slici 5.1b prikazana je prva uspješno detektirana i klasificirana slika, dotad ne viđena mreži. Postotak pouzdanosti (0.54) je jasno na donjoj granici i naslučuje prve uspjehe prilagođenoj neuronskoj mreži koja nikad prije nije bila osmišljena za detekciju teksta.

S vremenom, mreža je postajala sve pouzdanija te je nakon ≈ 350000 prvi puta prepoznala cijeli izraz, odnosno sve članove istog. Priložene slike (5.1a i 5.1b) također nikad prije nisu bile viđene od mreže i dobar su primjer jer se jasno vidi kako iako su slike identične, nakon određenog dodatnog broja koraka, mreža preciznije određuje granice individualnih simbola i puno pouzdanije prenosi koji su.

Nakon završenog treninga bilo je zanimljivo probati je li mreža ušla u fazu *overfit-*

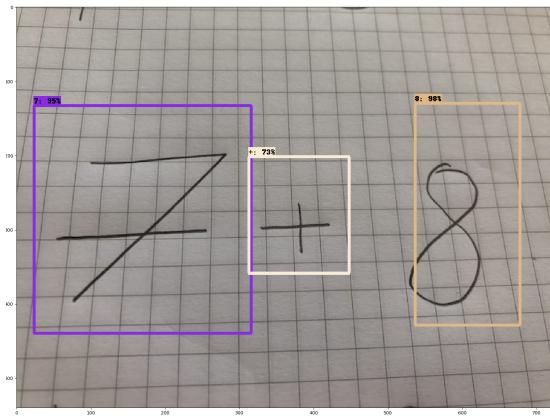


(a) *False positive* slika naizgled prazna no na (b) Prva detekcija računalno generirane brojke njoj je neodređeni element, nevidljiv zbog nepravilnosti na slici uz mali postotak pouzdanosti vilnih transformacija primjenjenih ma njemu



(a) Pouzdanost i preciznost nakon 350000 koraka (b) Pouzdanost i preciznost nakon 500000 koraka

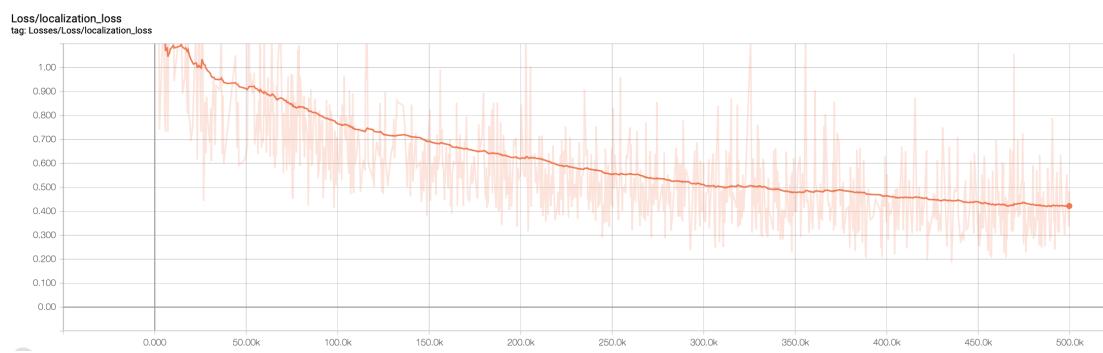
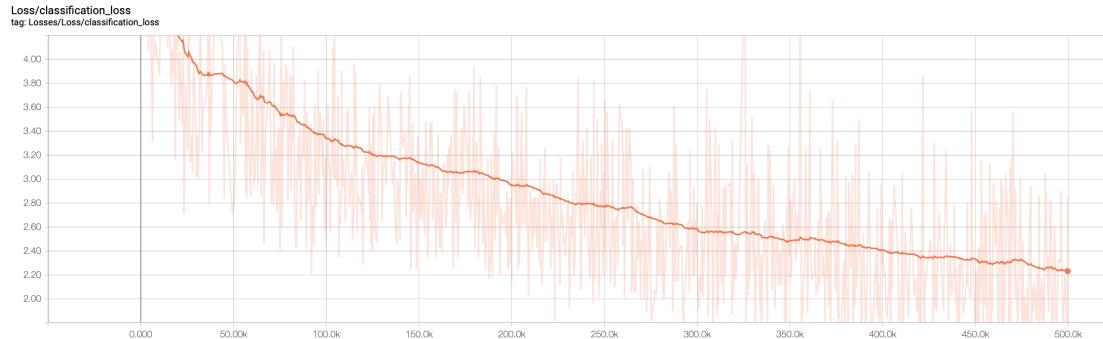
a što se vrlo lako moglo provjeriti dajući joj stvarni, rukom napisani primjer kojeg je bez problema detektirala i klasificirala (Slika 5.1).



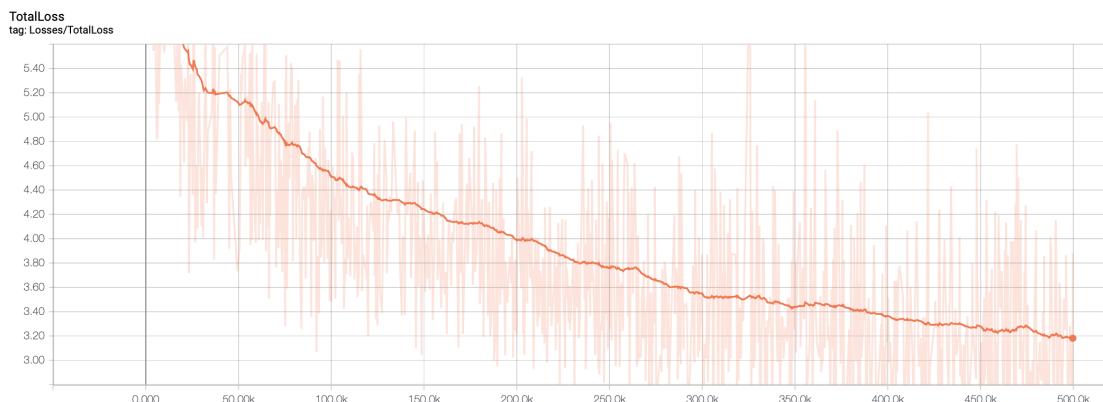
Slika 5.1: Jednostavan ručno napisan matematički izraz koji prikazuje uspjeh rada mreže na istom

5.3. *Tensorboard* rezultati

Velika prednost korištenja *Tensorflow*-a nad ostalim knjižnicama za duboko učenje je korištenje *Tensorboard*-a koji fantastično i uživo prikazuje napredak učenja mreže zadanog zadatka. Slika 5.2a prikazuje način na koji *Tensorboard* prikazuje gubitke kroz vrijeme, dok 5.2b prikazuje ukupan gubitak kroz vrijeme. Naravno, naprednim korištenjem alata mogu se prikazati i ostale korisne stvari kao što je prikaz rada mreže na određenom broju slika uživo, no za moje potrebe, prikaz gubitka je bio dostatan za razumjevanje procesa učenja.



(a) Napredak dvije komponente gubitka kroz vrijeme



(b) Ukupan gubitak nastao od lokalizacijskog i klasifikacijskog gubitka kroz vrijeme

LITERATURA

Keras.io. URL <https://keras.io/>.

Opencv-python dokumentacija. URL https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html.

Tfrecords vodič. URL <http://warmspringwinds.github.io/tensorflow/tf-slim/2016/12/21/tfrecords-guide/>.

Tensorflow službena stranica. URL https://www.tensorflow.org/tutorials/load_data/tf_records.

F. Chollet. *Deep Learning with Python*. Manning Publications Company, 2017. ISBN 9781617294433. URL <https://books.google.hr/books?id=Yo3CAQAAQAAJ>.

Alexandre Gonfalonieri. How to build a data set for your machine learning project, 2019. URL <https://towardsdatascience.com/how-to-build-a-data-set-for-your-machine-learning-project-5b3b871881ac>.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, i Alexander C Berg. Ssd: Single shot multibox detector. U *European conference on computer vision*, stranice 21–37. Springer, 2016.

Keiron O’Shea i Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

Shaoqing Ren, Kaiming He, Ross Girshick, i Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. U *Advances in neural information processing systems*, stranice 91–99, 2015.

Karen Simonyan i Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.