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, lipanj 2019.

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. Uvod

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

Jedan od najraširenijih alata koji se danas koristi za rješavanje algoritamski nemogućih ili izrazito kompliciranih problema je *umjetna inteligencija*. Umjetna inteligencija podrazumjeva skup načina i metoda koje računalu opisuju početno i konačno stanje do kojeg mora doći sam.

Ovaj rad će se baviti najkorištenijom metodom umjetne inteligencije, *dubokim učenjem*, i njegovim podskupom *računalnim vidom*. Kroz rad i programsku implementaciju prihvatiti 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 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 mogućnost razumjevanja slike na visokoj razini, najčešće s ciljem automatiziranja ljudskih

poslova. Osnovni zadatak je raspoznavanje veze između obrazaca na slici i rješenja problema koji se želi riješiti (1). Svi procesi koji koriste strojno učenje u konačnici se svode na detekciju i klasifikaciju elemenata na slici. Metode računalnog vida temelje se na geometriji, statistici, fizici i teoriji učenja.
Danas se velika količina problema rješava uz pomoć računalnog vida, često da ljudi toga 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.

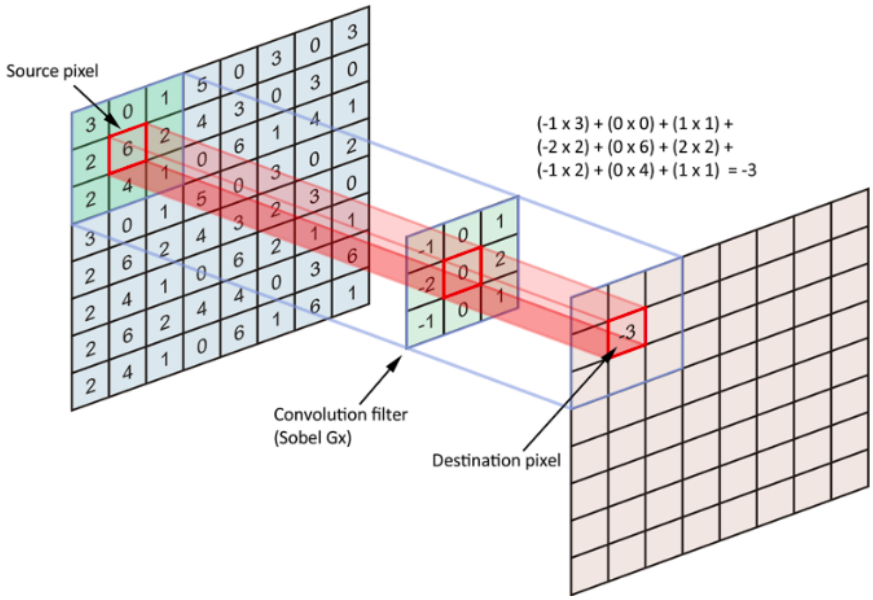
Slika 1.1: Maskiranje elemenata na slici prometa

1.2.2. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže su podskup dubokih neuronskih mreža, većinom primjenjene nad vizualnom mediju (slika, video) (**?**). Glavna prednost nad potpuno povezanim neuronskim mrežama je manji broj *težina* za učenje što ga i cijeli proces znatno ubrzava. 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* funkcionira 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 jest jedino manje razlučivo. Bitno je napomenuti da smanjivanjem rezolucije dobivamo puno manje parametara za učenje. Stavimo to u brojeve.
Slike unutar *mnist* skupa podataka su veličine 28×28. To znači da bi se učilo $28 * 28$ parametara. Primjenom *Max pooling sloja* veličine 2×2, učilo 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 (2).

2. Generiranje skupa 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 (3):

1. Prikupljanje
2. Klasifikacija
3. Označavanje

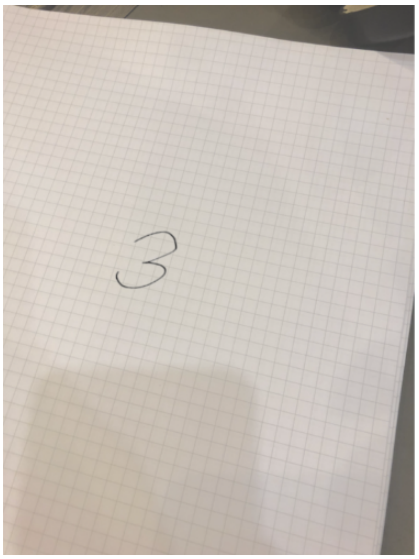
Prikupljanje podataka

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. Odlučeno je koristiti metodu računalnog generiranja vlastitog skupa podataka. Razlog tome je raznolikost elemenata koje mreža mora moći detektirati i fleksibilnost koju dobivamo jednom kada ustanovimo 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 koristi se `.csv` datoteka za dohvaćanje i kao opisnik slika. Postupak automatskog generiranja slika uvelike je olakšao klasifikaciju i označavanje jer je cijeli postupak ostvaren kao "cjevovod". 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 tome što slika nije zadana apsolutnom putanjom, što znači da su slike mogle biti kreirane na vlastitom računalu, prenešene na udaljeni server za treniranje i bez komplikacija biti korištene.
Veličina opisnika je također bila zanemariva.
Nakon raspodjele `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`.



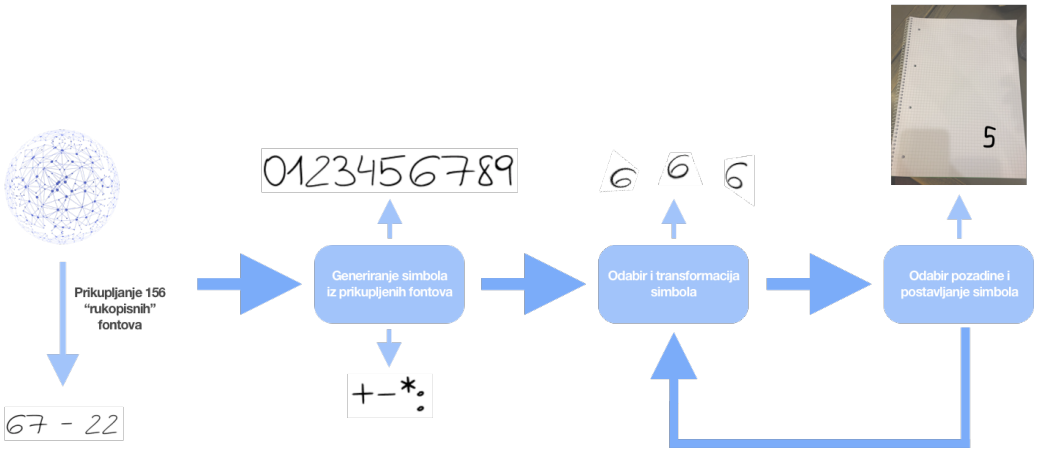
filename	class	imwidth	imheight	xmin	xmax	ymin	ymax
eqjodd.png	:	540	720	175	189	263	361
arnqxq.png	:	540	720	254	277	510	624
tvxdvf.png	*	540	720	394	455	579	669
fapsod.png	:	540	720	32	67	104	243
qardct.png	-	540	720	257	292	246	375
thzsfh.png	-	540	720	211	253	149	212
vdtpni.png	3	540	720	34	157	311	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
khhwwh.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, +, -, *, :) rezultati su pokazali da je potrebno minimalno 10 000 slika. Ne samo zbog broja elemenata već i zbog složenosti i raznolikosti između njih. Razvijeni postupak primjenjuje sve taktike (4) potrebne za stvaranje raznovrsnog i kvalitetnog skupa podataka. Zbog transformacija opisanih u daljnjim dijelovima 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*, razvijen 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, prikupljen je i 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`

2.2.3. Generiranje simbola

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

Slika 2.3: Varijacije unutar simbola uzrokovane fontovima

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 unijeti raznolikost unutar skupa podataka u slučaju premalog ili presličnog broja slika. Klasa *ImageGenerator* za to se brine na sličan način kao programski paket *Keras.preprocessing.image.ImageDataGenerator* (5). Transformacije nad slikama izvedene su pomoću programskog paketa *OpenCV* (6) 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*.

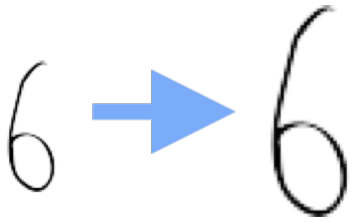
Skaliranje

Skaliranje pomoću *OpenCV* paketa može se izvoditi ili ručno, specifirajući točnu veličinu, ili dajući faktor skaliranja. *OpenCV* također automatski primjenjuje *interpolaciju* kako bi se kvaliteta maksimalno sačuvala. Skaliranje se izvodi 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

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



Slika 2.4: Rezultat primjene skaliranja uz faktore $s_x = s_y = 1.25$

Rotacija

Rotacija slike za kut θ ostvaruje se množenjem s matricom rotacije:

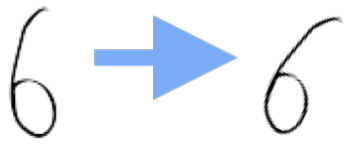
$$M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Iako je rotiranje izvedeno iz središnje točke, *OpenCV* nudi podršku za eksplicitno zadavanje točke oko koje će se rotacija izvoditi. Rezultat rotiranja vidljiv je na slici 2.5.

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

Afine transfromacije

Afine transformacije koristimo za prividno transformiranje simbola "u prostoru", bez velikog rizika od prevelike distorzije slike jer, sve paralelne linije nakon tran-



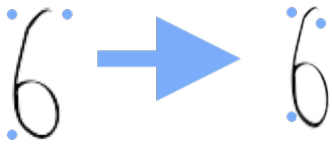
Slika 2.5: Rezultat primjene rotacije s $\theta = 25$

sformacije ostaju paralelne. *OpenCV* afinu transformaciju vrši tako da tri odabrane točke na slici pomakne za određeni koeficijent. Kao i ostale transformacije, matematički nastaje množenjem matrice slike s matricom afine transformacije oblika:

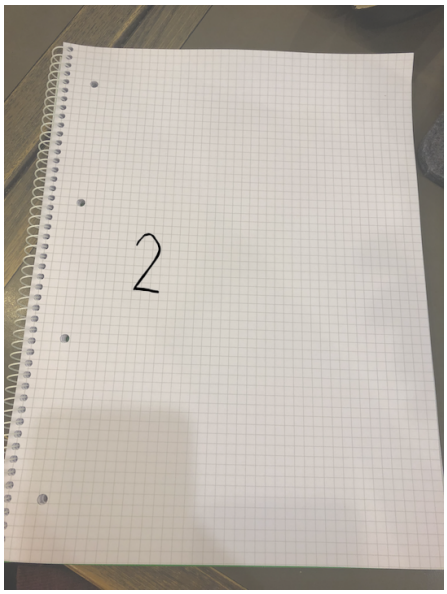
$$\begin{bmatrix} 1 & \tan(\beta) \\ \tan(\alpha) & 1 \end{bmatrix},$$

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

```
image = cv2.imread(image_path, -1)
height, width = image.shape
# Tocke u pripadajucim uglovima
affine_ref_points_1 = np.float32([[0, 0],[s_height, 0],
    [0, s_width]])

t1_delta = [random_float(0, height / 3),
        random_float(0, width / 3)]
t2_delta = [random_float(2 * height / 3, height),
        random_float(0, width / 3)]
t3_delta = [random_float(0, height /3),
        random_float(2 * width / 3, width)]

affine_ref_points_2 = [t1_delta, t2_delta, t3_delta]

M = cv2.getAffineTransform(affine_ref_points_1,
        affine_ref_points_2)

image = cv2.warpAffine(image, M, (width, height))
```



Slika 2.6: Rezultat primjene afine transformacije s pripadajućim referentnim točkama



(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

2.2.5. 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 potrebe ovog rada izabrana je pozadina matematičkih bilježnica uz pretpostavku da bi se iste najčešće koristile kada bi se naučena 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*.

2.3. TFRecords

Zašto je bolje za mrežu da podatke čita iz `.record` datoteke nego odvojeno slike i pripadajuće opise? Zamislimo sljedeći scenarij. Učenje se izvodi 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 (7).
U pozadini, *TFRecords* je format koji koristi *Protocol buffer* tehnologiju. *Protocol buffer* ili kraće *Protobuf* je knjižnica za efikasnu serijalizaciju strukturiranih podataka ((8)). 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 ključ `feature`.
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.
Primjerice, 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 praćeni su koraci opisani 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 skup pravokutnika koji 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 (9). No, nijedna od tih nije uspjela doseći gotovo "real-time" brzinu sa značajnom preciznosću

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

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

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

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

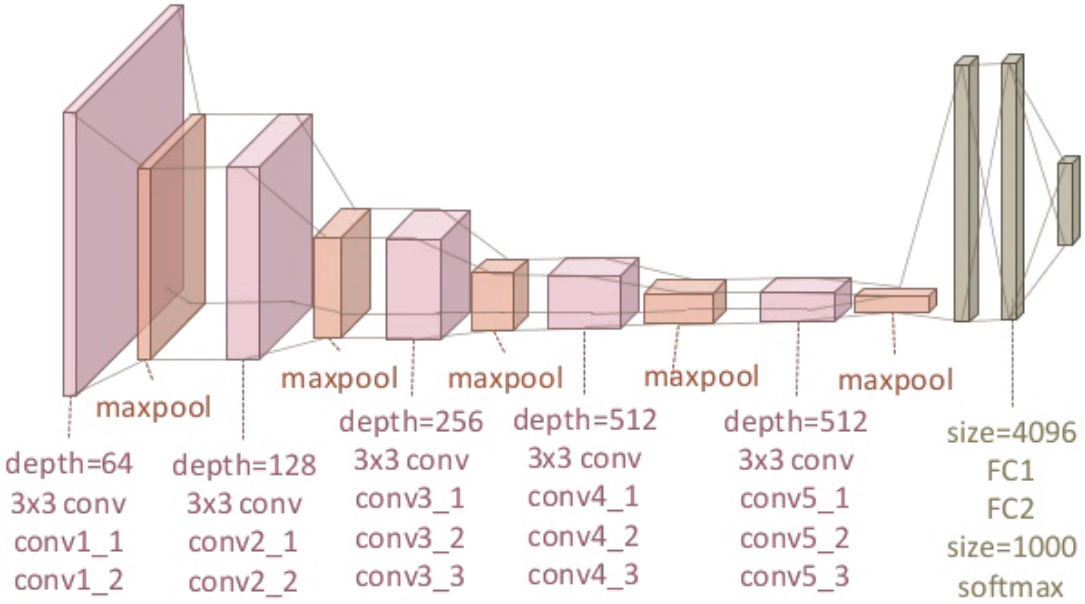
3.2. Arhitekture korištenih mreža

3.2.1. Arhitektura *VGG-16*

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

VGG mreža razvijena je da bude jednostavna, sadržavajući samo `3x3` konvolucijske i `2x2` pooling slojeve prije završnih gusto spojenih slojeva (10). Arhitektura mreže vidljiva je na slici 3.2 Također, cijela struktura, težine i cijela naučena 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 učenje što je odlična stvar za ponovno korištenje mreže za druge primjene.

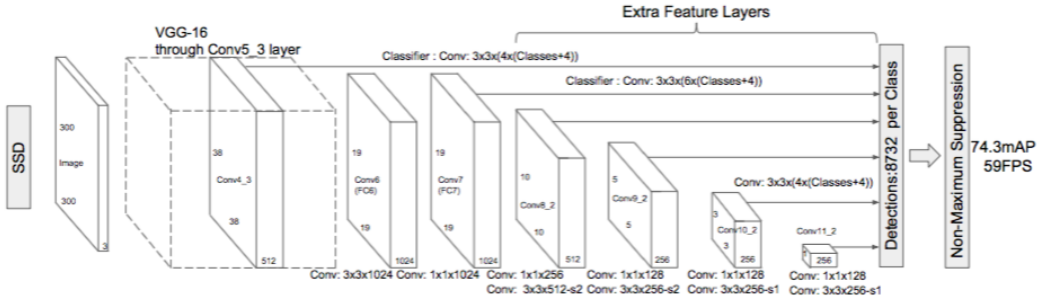
Jedna od tih primjena je *SSD* mreža, koja na svom početku sadrži baš *VGG-16* arhitekturu, sve do gusto spojenih slojeva koje odbacuje.



Slika 3.2: Arhitektura VGG-16 mreže

3.2.2. Arhitektura SSD

Razlog zbog kojeg *SSD* mreža koristi *VGG-16* kao baznu mrežu je njezina snažna performansa na slikama visoke kvalitete i popularnost gdje tehnika prenošenja težina pomaže pri dobrim rezultatima. Umjesto gusto spojenih slojeva *SSD* mreža dodaje još konvolucijskih slojeva koji dalje izvlače značajke i progresivno smanjuju ulaz svakom dubljem sloju (11). Cijela arhitektura *SSD* mreže vidljiva je na slici 3.3.



Slika 3.3: Arhitektura SSD mreže

4. Učenje

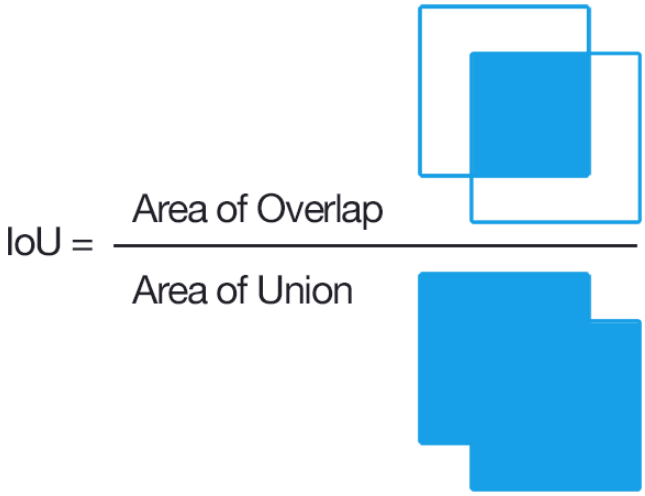
4.1. Učenje SSD neuronske mreže

4.1.1. Visoki pogled na učenje

Bitna razlika u učenju *SSD* mreže i tipične *R-CNN* mreže slične zadaće je ta da "ground truth" podatak mora biti dodjeljen točnom izlazu iz fiksnog skupa izlaza detektora (11). 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 određivanje težina koristi se tehnika *back propagation*. Prije početka učenja također se određuju pretpostavljeni pravokutnici, različite skale za detekciju i strategije za povećanje podataka.
O pretpostavljenim kvadratima, skalama za detekciju i strategijama pisati će se u nastavku.

4.1.2. Određivanje pozicije objekata

Tijekom učenja, cilj je odrediti koji pretpostavljeni pravokutnici najbolje odgovaraju "ground truth" pravokutnicima objekta, to jest onima specifiranim u skupu podržavamotaka. Nakon što se odrede najprecizniji pravokutnici, mreža se sukladno tome dalje prilagođava. Za svaki "ground truth" pravokutnik imamo na izbor više pretpostavljenih pravokutnika, različitih lokacija, skala i omjera. Želimo naći onaj koji ima najveći *jaccardov index preklapanja* (dalje IoU) (slika 4.1) (12).
U konfiguracijskoj datoteci možemo ručno odrediti od koje preciznosti pretpostavljeni pravokutnik prihvačamo. Pretpostavljena vrijednost je da mora vrijediti $IoU \geq 0.5$. To višestruko olakšava treniranje jer mreža zadržava više pretpostavljenih pravokutnika 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 učenja

Određivanje parametara bilo bi puno lakše kada bismo imali samo jedan objekt za klasificirati, no postaje kompliciranije uz više objekata. Uzmimo $x_{ij}^p = \{1, 0\}$ kao indikator za podudaranje i-tog pretpostavljenog pravokutnika na j-ti "ground truth" pravokutnik 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 lokalizacijskog (*loc*) i klasifikacijskog (*conf*) gubitka (11):

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \tag{4.1}$$

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

Lokalizacijski gubitak (*loc*)

Za izračun lokalizacijskog gubitka koristimo *Smooth L1* između predviđenih i "ground truth" pravokutnika (11).

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in (cx,cy,w,h)} x_{ij}^k \; smooth_{L1}(l_i^m - \hat{g}_j^m) \tag{4.2}$$

Klasifikacijski gubitak (*conf*)

Klasifikacijski gubitak računa se kao *softmax* svih klasa koje podržavamo (11).

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p \log(\hat{c}_i^0) - \sum_{i \in Neg} \log(\hat{c}_i^0) \; gdje \; \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)} \tag{4.3}$$

4.1.4. Definiranje trajanja učenja

Tijek i trajanje učenja može se opisati koristeći više parametara. Korišteni *Tensorflow object detection API* zahtjeva određivanje broja koraka, dok se u litraturi, npr. (4) najčešće koristi epoha.

Korak

Mreži se u konfiguracijskoj datoteci predaje parametar *batch size*. *Batch size* definira koliko će se u jednom trenutku uzeti slika za učenje. Parametar, iako može biti jedan, najčešće je 24 (8). Nakon što mreža obradi sve slike uzete u jednom trenutku ona popravlja svoje parametre, odnosno težine. Upravo to razdoblje od uzimanja određenog broja slika, i obrade svih definira *korak*. Koliko korak traje? Učenje mreže za ovaj rad izvodilo se na dva uređaja zbog potrebe mjerenja i vidljivo je na tablici 4.1.4 koja prikazuje povezanost procesnih jedinica uređaja na kojima se učenje vršilo i pripadajuća vremena po jednom koraku treniranja.

Specifikacije uređaja	Vrijeme po koraku (s)
Macbook Pro 2016, Intel i5 2.9GHz, 8GB	13.57
ZEMRIS C22, NVidia GTX 1080 Ti, 11GB	0.86

Epoha

Epoha označava pregled cijelog skupa podataka. To naravno ne znači da je svaka slika pogledana točno jednom jer su nasumično učitavane u mrežu ali onaj trenutak kada su sve barem jednom prošle kroz mrežu označava kraj jedne epohe. Uz pretpostavku da je svaka slika pogledana jednom, približno se može izračunati broj epoha iz broja koraka formulom 4.4.

$$brojepoha = brojKoraka \times \frac{batchSlika}{brojSlika} \tag{4.4}$$

5. Rezultati

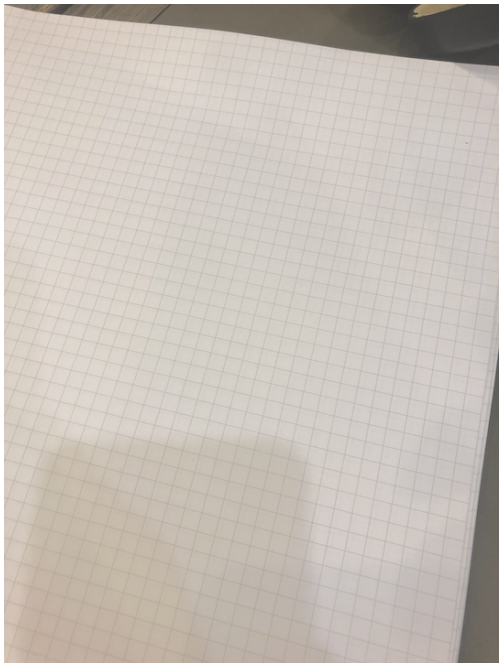
5.1. Opis rezultata

Mreža se s vremenom ponašala iznimno zanimljivo. Već u prvih par koraka vrijednost ukupne funkcije gubitka 4.1 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 skupa podataka. 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 učenja ≥ 20000) nisam ručno mogao izbaciti sve. Ipak, smatram da je određeni mali broj takvih slika neophodan za uspješno učenje slike.
Nakon ukupno 500000 koraka, tj. 80 epoha po formuli 4.4, ukupni gubitak iznosio je 3.879 gdje je po izgledu dosegao asimptotu.

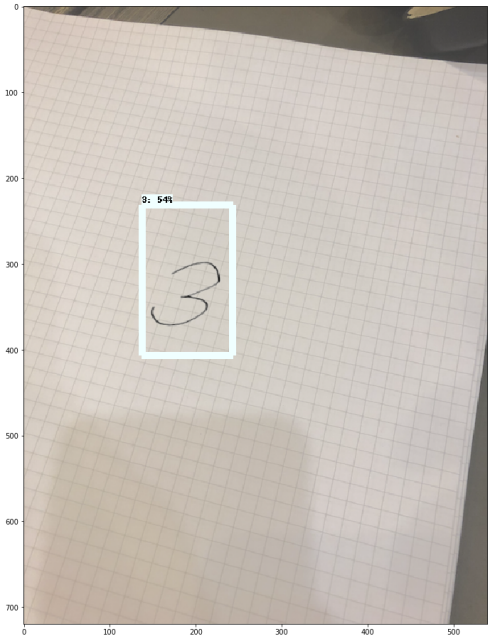
5.2. Prikaz rezultata

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 naslućuje 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 koraka 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.

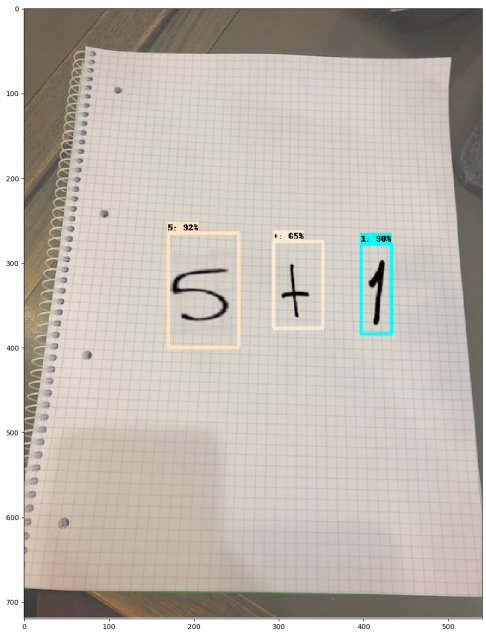
Nakon završenog učenja bilo je zanimljivo probati je li mreža ušla u fazu *pre-*



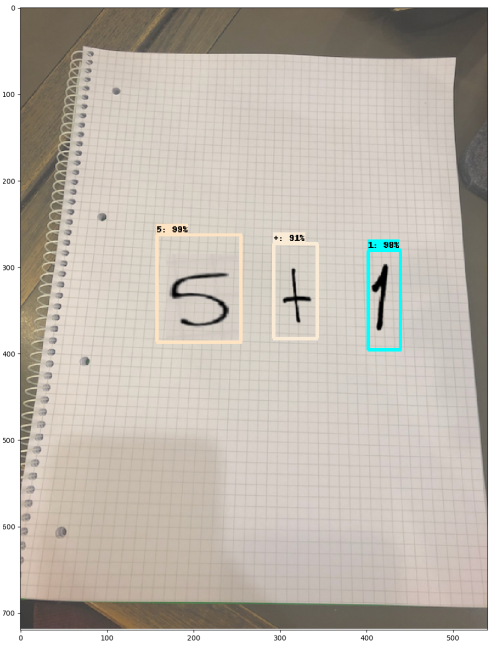
(a) *False positive* slika koja sadrži neodređeni element naizgled nevidljiv zbog nepravilnih transformacija primjenjenih na istom



(b) Prva detekcija računalno generirane znamenke 3 na slici uz mali postotak pouzdanosti

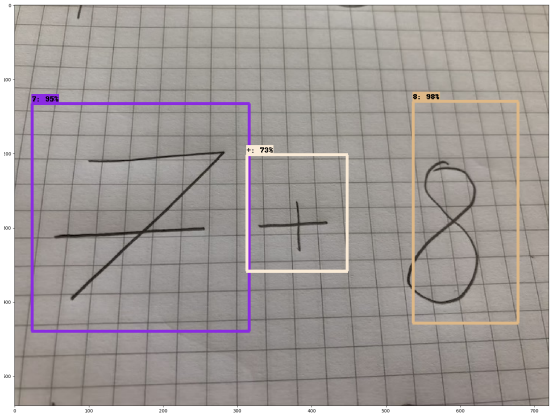


(a) Pouzdanost i preciznost nakon 350000 koraka



(b) Pouzdanost i preciznost nakon 500000 koraka

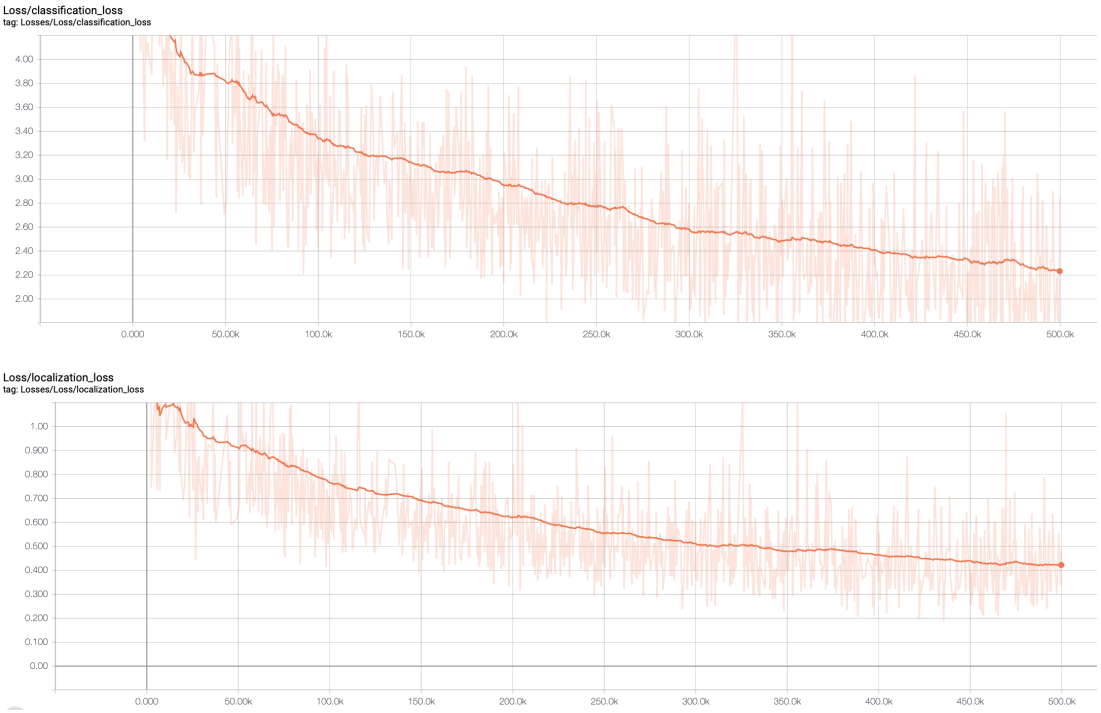
naučenosti-a što se vrlo lako moglo provjeriti dajući joj stvarni, rukom napisani primjer kojeg je bez problema detektirala i klasificirala (Slika 5.1).



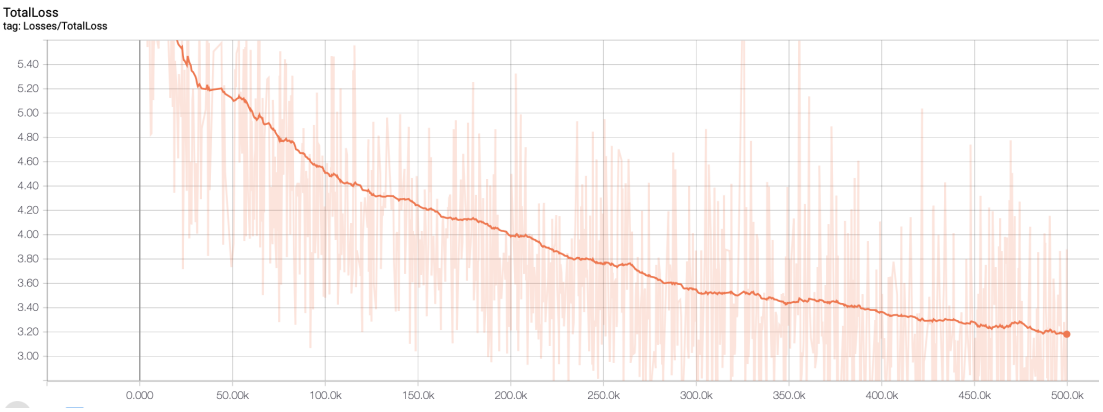
Slika 5.1: Jednostavan ručno napisan matematički izraz koji prikazuje uspjeh rada mreže na istom

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

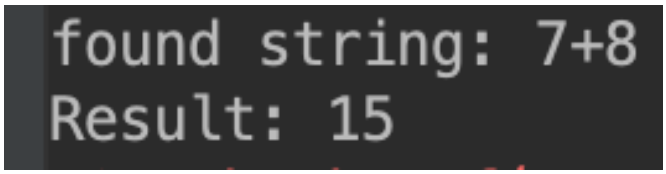
6. Programska podrška

6.1. Korištenje programske podrške

Programska podrška napravljena u svrhu demonstracije svih spomenutih značajki ima dva glavna načina rada. Ispis pronađenog teksta i pretraga unešenog teksta.

6.1.1. Ispis pronađenog teksta

Način rada koji se pretpostavlja da korisnik želi koristiti. Kroz argument predaje se apsolutna putanja do slike nad kojom se želi izvršiti detekcija a program na standardni izlaz ispisuje pronađeni tekst. Pronađeni tekst zatim prolazi evaluaciju koja je dio standardne biblioteke programskog jezika *Python*. Ako je izraz moguće evaluirati npr. "2 + 2", na standardni izlaz ispisuje se rezultat operacije kao što je vidljivo na slici 6.1.



Slika 6.1: Izlaz nakon evaluacije pronađenog teksta

6.1.2. Pretraga unešenog teksta

Drugi način rada nakon što slika prođe kroz korak detekcije i klasifikacije teksta sprema pronađeni izraz u obliku niza. Korisnik pri izvršavanju programa prethodno mora unjeti regularni izraz koji opisuje ono što želi znati nalazi li se na slici na što program odgovara na standardni izlaz postoji li ili ne. Način pretrage unešenog teksta dostupan je postavljanjem zastavice `r` kao argument pri pokretanju.

6.2. Budući rad

Iako programska podrška trenutno pruža osnovnu funkcionalnost, postoji dosta prostora za rast i napredak. Trebala bi se znati vršiti segmentacija izraza po retcima jer se trenutno pronađeni objekti sortiraju s lijeva na desno i na isti način evaluiraju. Segmentacija po retcima pružila bi mogućnost za kompliciranije izraze i kompletniju implementaciju.
Također, bilo bi korisno mrežu spojiti sa jednostavnim grafičkim sučeljem koje bi znatno olakšalo korištenje van komandne linije.

7. Zaključak

Ovaj rad obuhvatio je cjelokupni cjevovod jednog ciklusa pripreme duboke neuronske mreže za specifičan zadatak. Proces stvaranja slika omogućio je gotovo neograničen skup podataka za neuronsku mrežu. Isti skup podataka korišten je za za prenamjenjivanje *SSD* neuronske mreže za detekciju na zadatak prepoznavanja rukom napisanih simbola. U detalje je opisana arhitektura spomenute mreže i metode koje koristi za računanje kvalitete izlaza.

Inicijalno, mreža je bila naučena za prepoznavanje objekata unutar 6000 kategorija. Dalje je korištena metoda prenamjenjivanja mreže na druge kategorije koristeći postojeće težine. Spomenuta pretpostavka uvelike je olakšala i ubrzala cijeli postupak jer mnogi primitivi koji postoje unutar tih 6000 kategorija, pojavljuju se i u simbolima.

Nakon procesa učenja u 15000 koraka tj. 80 epoha, dosegnuta je zadovoljavajuća točka preciznosti i pouzdanosti na većini simbola. Jedini simbol, koji nije uspješno naučen je simbol množenja što je i jedan od budućih izazova. Ostvariti zadovoljavajuću preciznost na svim simbolima i vršiti segmentaciju po retcima. Također, preporučuje se i korištenje drugih arhitektura (npr. *Faster R-CNN*) te usporedba rezultata.

Literatura

[1] D. H. Ballard and C. M. Brown, "Computer vision. englewood cliffs," *J: Prentice Hall*, 1982.

[2] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[3] A. Gonfalonieri, "How to build a data set for your machine learning project," 2019.

[4] F. Chollet, *Deep Learning with Python*. Manning Publications Company, 2017.

[5] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[6] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[7] D. Pakhomov, "Tfrecords guide," 2016.

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[9] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[12] E. Forson, "Understanding ssd multibox," 2017.

Detekcija i klasifikacija tekstovnih elemenata na slici koristeći duboke neuronske mreže

Sažetak

Rad je opisao postupak automatskog pozicioniranja, transformiranja i spremanja tekstovnih elemenata na sliku u oblik čitljiv neuronskoj mreži. Korišten je *Tensorflow Object Detection API* za postupak učenja detekcije i klasifikacije istih elemenata na slici. Ako je na slici pronađen matematički izraz, evaluacijom je dan rezultat istog.

Ključne riječi: detekcija, klasifikacija, duboke neuronske mreže, strojno učenje, Tensorflow, Keras, generiranje slike

Detection and classification of text based elements on an image using deep neural networks

Abstract

This paper describes the whole process of automatically positioning, transforming and saving textual elements on an image in a form that is readable by a neural network. *Tensorflow Object Detection API* is used for training detection and classification of generated elements on an image. If an image contains a mathematical expression, evaluating returns the solution.

Keywords: detection, classification, deep neural networks, machine learning, Tensorflow, Keras, generating images