

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	13
3.2.1. VGG-16 arhitektura	13
3.2.2. SSD arhitektura	14
3.3. MultiBox tehnika	16
3.3.1. Lokalizacijski gubitak	17
3.3.2. Klasifikacijski gubitak	17
4. Treniranje SSD mreže	18
Literatura	19

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 komplicirani 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, 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 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 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 na problem koji želi riješiti. 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 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

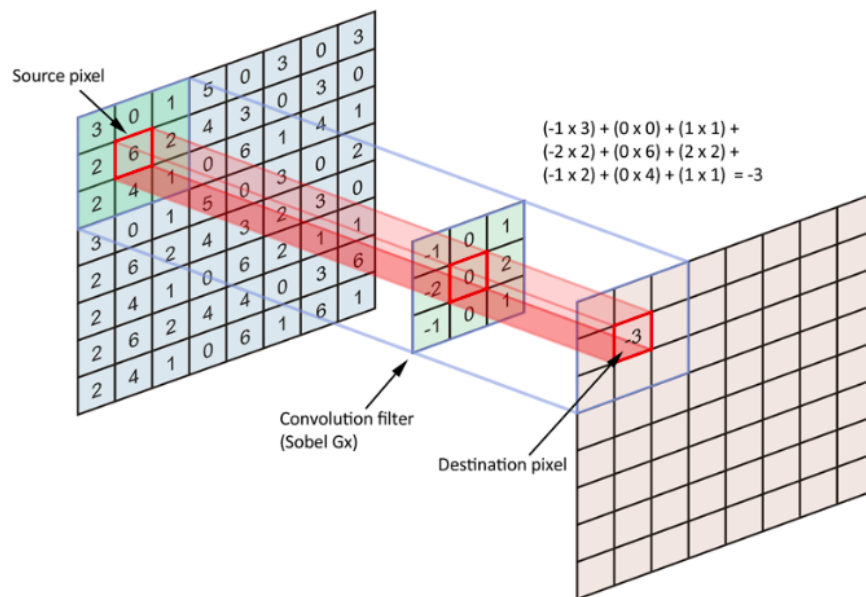
Konvolucijske neuronske mreže danas se koriste kao najefektivniji način postizanja računalnog vida. Glavna prednost nad potpuno povezanim neuronskim mrežama je manji broj *težina* za treniranje što znatno ubrzava treniranje. Ipak, ono što je možda najvažnije za napomenuti je to što pozicija traženog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

Svaki sloj *duboke konvolucijske neuronske mreže* 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 mijenjajući 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

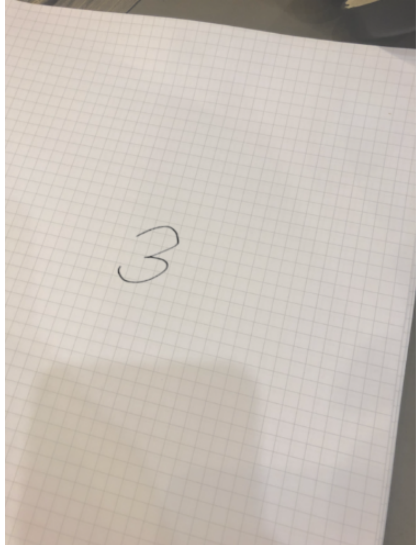
Prikupljanje podataka mora biti sustavan i smislen proces jer može otežati i olakšati daljnje korake. Najpreporučeniji način za prikupljanje je dugoročno i postepeno spremanje podataka jer rezultira velikim brojem objektivnih i kvalitativnih podataka. Ja sam se ipak odlučio na metodu računalnog generiranja 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* \times *širina* \times *kanali*) mreži su potrebni i podaci za uspoređivanje rezultata i računanje uspješnosti. U ovom radu koristio sam .csv datoteku 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 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 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.



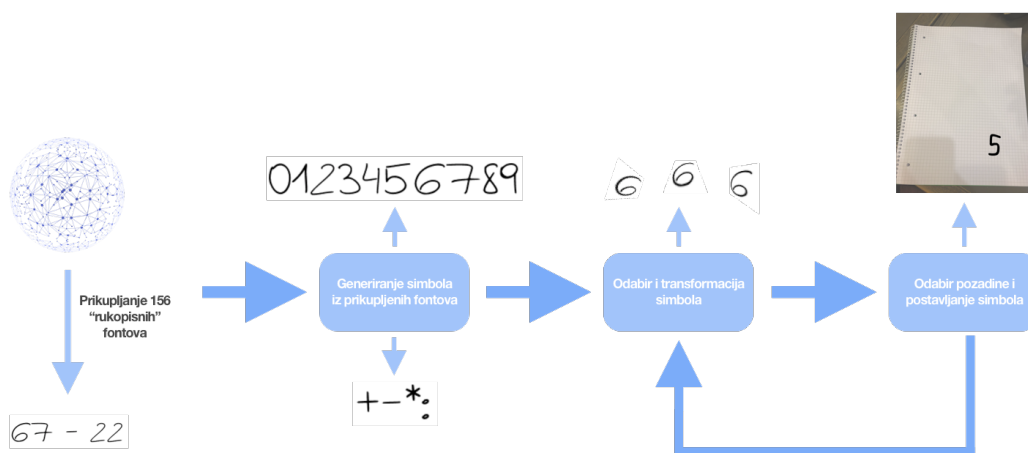
filename	class	imwidth	imheight	xmin	xmax	ymin	ymax
eqjodd.png	:	540	720	175	189	263	361
arnqxq.png	:	540	720	254	277	510	624
tvxdvf.png	*	540	720	394	455	579	669
fapsod.png	:	540	720	32	67	104	243
qardct.png	-	540	720	257	292	246	375
thzsfh.png	-	540	720	211	253	149	212
vdtpni.png	3	540	720	34	157	311	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
khhwhh.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 daljnjim 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.

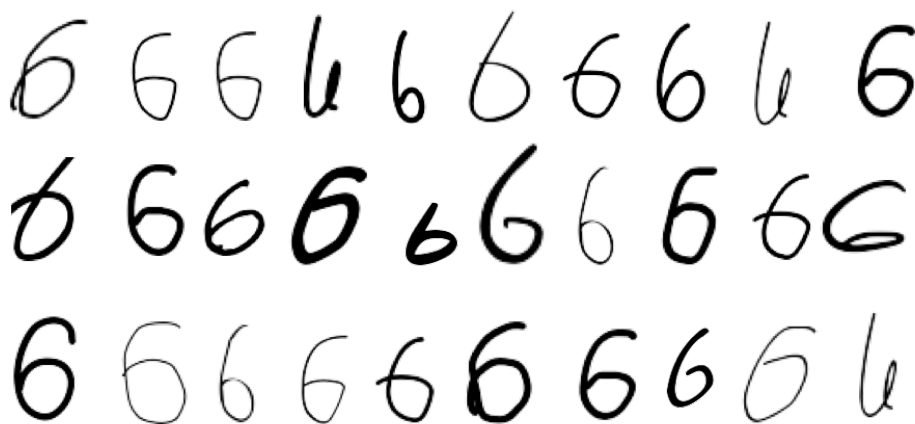


Slika 2.2: Prikaz visoke razine cjevovoda za generiranje slika

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 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*.

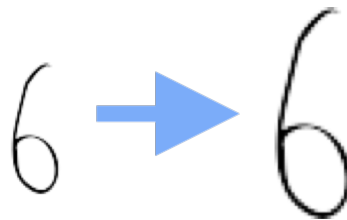
Skaliranje

Skaliranje pomoću *OpenCV* paketa može se vršiti ili ručno, specificirajući 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 $s_x = s_y = 1.25$

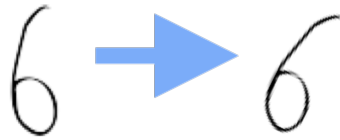
Rotacija

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 eksplicitno 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 $\theta = 25$

Afine transformacije

Afine 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 odabrane točke na slici pomakne za određeni koeficijent. Kao i ostale transformacije, matematički nastaje množenjem matrice slike s matricom afine transformacije oblika.

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

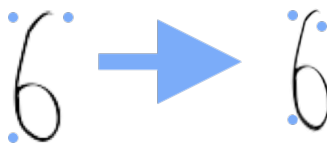
gdje su α i β razlike u kutevima prema pripadajućim koordinatnim osima. Rezultat primjene afine transformacije na simbolu vidljiv je na slici 2.6

```
1 image = cv2.imread(image_path, -1)
2 height, width = image.shape
3 # Točke u pripadajucim 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))

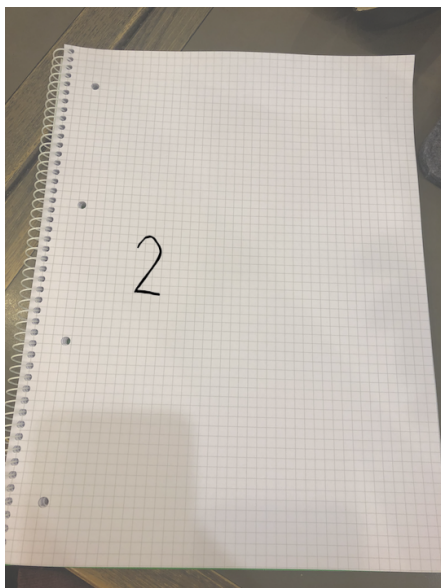
```



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/lukasesttic/MachineLearning/Završni Rad/Backgrounds/A4_math_5.png
Smaller images: ['5/Easy/Symbol_5_69.png'] ; Background: /Users/lukasesttic/MachineLearning/Završni Rad/Backgrounds/A4_math_3.png
Smaller images: ['-/Easy/Symbol_-98.png'] ; Background: /Users/lukasesttic/MachineLearning/Završni Rad/Backgrounds/A4_math_6.png
Smaller images: ['*/Easy/Symbol_*_95.png'] ; Background: /Users/lukasesttic/MachineLearning/Završni Rad/Backgrounds/A4_math_5.png
Smaller images: ['7/Easy/Symbol_7_139.png'] ; Background: /Users/lukasesttic/MachineLearning/Završni Rad/Backgrounds/A4_math_6.png
Smaller images: ['2/Easy/Symbol_2_25.png'] ; Background: /Users/lukasesttic/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 broječanu vrijednost istog objekta koja ga predstavlja. Isti način je precizniji i sažetiji.

Konkretno ime dnevnika koji sadrži mapiranja iz objekta u njegovu broječanu 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 pratio 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 preciznošću 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 cjelokupnu implementaciju zadatka detekcije i klasifikacije 14 različitih klasa.

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 potječe ime. Mreža sama po sebi ostvaruje odlične



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

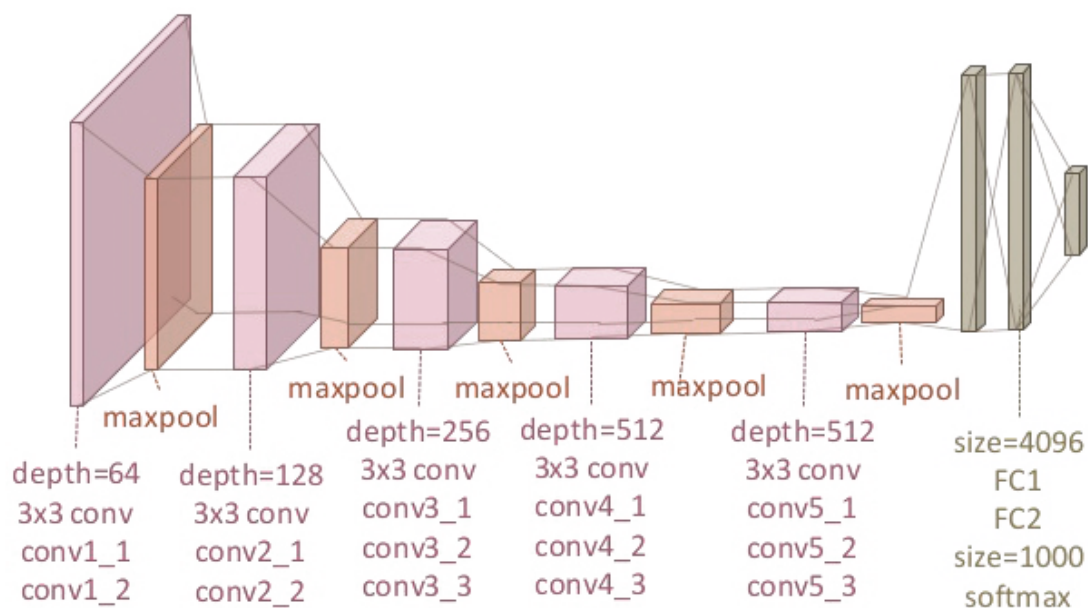
rezultate na *ImageNet* datasetu no to nije jedini razlog zašto je jedna od najkorištenijih.

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.

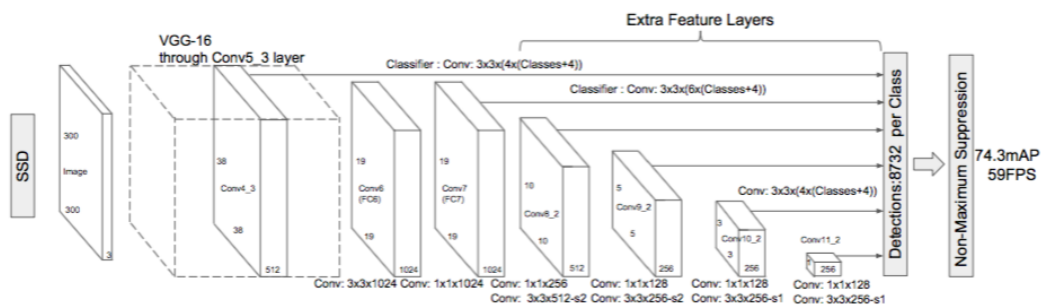
3.2.2. SSD arhitektura

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 dobrim rezultatima. Umjesto gusto spojenih slojeva *SSD*



Slika 3.2: Arhitektura VGG-16 mreže

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:

Detekcija značajki višestrukih veličina

Svaki dublji konvolucijski sloj postaje manji i time promatra značajke manje veličine. Konvolucijski model za predviđanje detekcija je drukčiji za svaki sloj.

Konvolucijsko predviđanje za detekciju

Za precizniju detekciju, različiti slojevi također prolaze kroz 3×3 konvoluciju kao što je prikazano na slici 3.3.

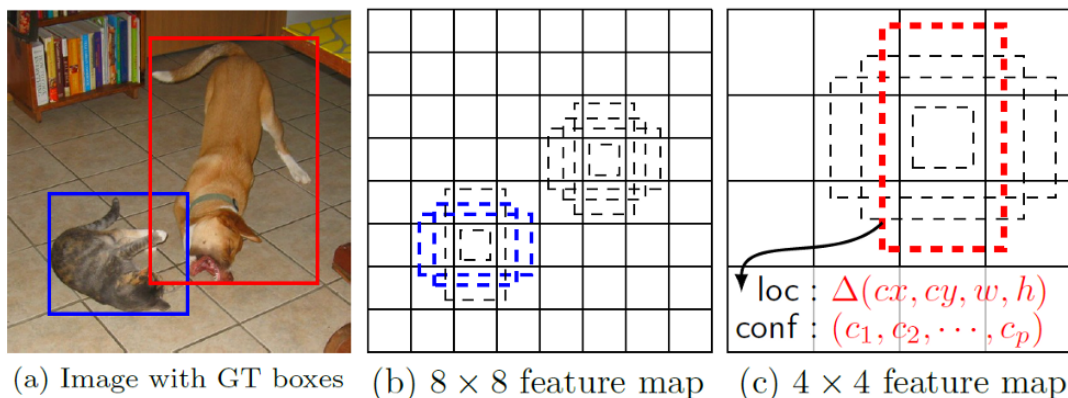
Uzmimo na primjer sloj `Conv4_3`. Veličina sloja je $38 \times 38 \times 512$. Na isti primjenjujemo konvoluciju veličine 3×3 . Imamo 4 kvadrata i svaki ima $klase + 4$ izlaza. Točnije, izlaz sloja `Conv4_3` oblika je $38 \times 38 \times 4 \times (klase + 4)$.

U mom slučaju, to bi bilo $38 \times 38 \times 4 \times (14 + 4)$ tj. 103968 izlaza, bez računanja klasa, dobili smo 5776 kvadrata. Isti postupak nastavlja se dalje na dublje slojeve (ssd).

3.3. MultiBox tehnika

MultiBox je tehnika za regresiju kvadrata koji ograničavaju poziciju objekta na slici. Pretpostavljamo da smo trening podatke pripremili na način da sadrže točne koordinate kvadrata objekta. Nakon što prođemo određeni broj konvolucija za izvlačenje značajki, dobivamo sloj veličine $m \times n \times p$ (Slika: 3.4). Za svako polje dobivamo k pretpostavljenih kvadrata.

Koncept je sljedeći. Uspravni kvadrat prikladan je npr. za znamenku "1" na slici, dok je za znamenku "4" ili simbol "+" prikladniji kvadrat kvadratnog oblika. Za svaki računamo x vrijednosti za klase i 4 pomaka relativna originalnom, te dobivamo $(c + 4) \times k \times m \times n$ izlaza. Kako izračunati kvalitetu lokalizacije i pouzdanje



Slika 3.4: Višestruki kvadrati za lokalizaciju i pouzdanje

klasifikacije?

3.3.1. Lokalizacijski gubitak

SSD za izračun lokalizacijskog gubitka koristi *L1-Norm*

$$|d|_{\text{smooth}} = \begin{cases} 0.5d^2, & \text{ako } |d| \leq 1 \\ |d| - 0.5, & \text{inače} \end{cases}$$

Iako nije precizan kao *L2-Norm*, iznimno je efektivna i ispunjava svrhu *SSD*-a koji se ne trudi biti točan u pixel.

3.3.2. Klasifikacijski gubitak

Za svaki dan kvadrat, *SSD* vrši klasifikaciju, te računa *softmax*.

Softmax se koristi kao standard kada imamo više od dvije klase jer sve dobivene vrijednosti stavlja u rang $[0, 1]$ uz sumu svih koja je jednaka 1.

$$\text{softmax}(a) = \frac{\exp a_i}{\sum a_i}$$

4. Treniranje SSD mreže

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=Yo3CAQAACAAJ>.

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.