Raspoznavanje objekata iz skupa podataka CIFAR-10 konvolucijskom neuronskom mrežom

Gazdek, Denis

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Denis Gazdek

Raspoznavanje objekata iz skupa podataka CIFAR-10 konvolucijskom neuronskom mrežom

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Denis Gazdek

Matični broj: 44851/16-R

Studij: Informacijski sustavi

Raspoznavanje objekata iz skupa podataka CIFAR-10 konvolucijskom neuronskom mrežom

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Ivković Nikola

Varaždin, rujan 2019.

Denis Gazdek

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

U ovom radu ću pojasniti što je strojno učenje, umjetna neuronska mreža, te konvolucijska neuronska mreža i kako je ona drukčija od obične neuronske mreže. U praktičnom dijelu ću razviti svoju konvolucijsku neuronsku mrežu u programskom jeziku *Python*, za razvrstavanje slika iz skupa podataka CIFAR-10, u 10 kategorija.

Ključne riječi: neuronske mreže; računska inteligencija; raspoznavanje objekata; CIFAR-10; *Python*; *Tensorflow*

Sadržaj

1. Uvod

Računalna inteligencija (eng. *Computational intelligence*) je grana umjetne inteligencije (eng. *artificial intelligence*) koja se bavi proučavanjem prilagodljivih mehanizama kako bi se omogućilo inteligentno ponašanje u složenim i promjenjivim okruženjima. Postoji pet glavnih paradigmi računalne inteligencije. To su umjetne neuronske mreže (eng. *artificial neural networks*) koje imitiraju biološke mreže neurona, evolucijsko računarstvo (eng. *evolutionary computation*) koje je bazirano na evolucijskoj biologiji, inteligencija roja (eng. *swarm intelligence*) bazirana na rojevima kukaca kao što su pčele i mravi, umjetni imunosni sustavi (eng. *artificial immune systems*) koji imitiraju ljudski imunosni sustav i neizraziti sustavi (eng. *fuzzy systems*), koji nalažu da odgovori na pitanja često nisu binarni (točno ili netočno), nego negdje između [1].

Strojno učenje je polje znanosti koje se bavi istraživanjem algoritama i statističkih modela koje računala koriste kako bi riješili neki problem, bez unaprijed zadanih uputa kako riješiti taj problem. Strojno učenje spada pod umjetnu inteligenciju, granu znanosti koja nastoji imitirati ljudsku inteligenciju, ili kako riješiti neki problem za koji je čovjeku potrebna inteligencija da bi ga riješio. Uz pomoć strojnog učenja, računalni sustavi iz novih podataka izvlače uzorke kako bi u budućoj sličnoj situaciji točnije reagiralo [2].

U radu je izrađena konvolucijska mreža za kategoriziranje CIFAR-10 seta podataka, te su pojašnjeni pojmovi potrebni za razumijevanje mreže. Konvolucijska mreža se može sastojati od različitih vrsta slojeva neurona, kao što su konvolucijski sloj, gusti sloj, sloj združivanja ili sloj ispadanja. Mreža je izrađena korak po korak, tako da se vidi jačina utjecaja pojedinih slojeva na točnost i trajanje treninga, te je izrađena u jeziku *Python*.

2. Umjetne neuronske mreže

Umjetne neuronske mreže su računalni sustavi koji su bazirani na biološkom mrežama neurona, te pokušavaju imitirati njihovo djelovanje. Sastavljene su od umjetnih neurona. Veze između umjetnih neurona imaju istu ulogu kao sinapse u stvarnim neuronima. Ti umjetni neuroni često su organizirani u slojeve. Umjetna neuronska mreža ima ulazni sloj, izlazni sloj, te može imati srednje, skrivene slojeve. Srednji slojevi se zovu skriveni jer se rezultat neurona srednjih slojeva šalje u sljedeći sloj, bez da postaju vidljivi korisniku. U ulaznom sloju neuronska mreža prima podatak (npr. u ulaznom sloju mogu biti pojedini pikseli slike). Iz izlaznog sloja se iščitava rezultat (npr. kategorija ulazne slike). Svaka od veza između neurona ima svoju težinsku vrijednost (eng. *weight*), koja određuje koliko je pripadni signal bitan [3].

2.1. Unaprijedne i povratne mreže

Postoje dvije glavne vrste neuronskih mreža: unaprijedne (eng. *feedforward*) mreže i povratne (eng. *recurrent*) mreže. Kod unaprijednih mreža, podaci prolaze od ulaza, preko srednjih slojeva, do izlaznih slojeva u jednom smjeru. U povratnim mrežama postoje povratne veze prema prošlim slojevima neurona. Povratne mreže su korisne ako ulazni podaci imaju vremensku dimenziju, te za procesiranje jezika [4].

2.2. Umjetni neuron

Umjetni neuron modelira biološki neuron. Umjetni neuron je funkcija koja ima R ulaza. Izlazna vrijednost neurona je rezultat funkcije:

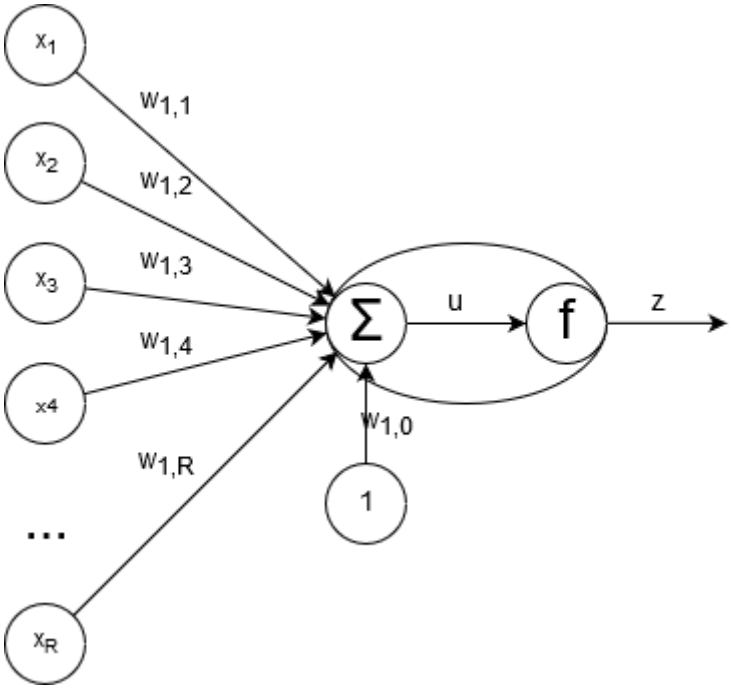
$$z = f\left(\sum_{i=1}^{R} x_i w_i + b\right)$$

gdje je $\mathrm{f}(\,)$ funkcija aktivacije, R broj ulaznih vrijednosti, w_i težinska vrijednost pojedine ulazne vrijednosti, x_i ulazna vrijednost, a b prag (eng. *threshold* ili *bias*), koji uglavnom ima vrijednost 1. Funkcija aktivacije se ovdje koristi kako bi se rezultat preslikao u neki pozitivni interval (kod sigmoide, između 0 i 1), te kako jako veliki i jako mali rezultati ne bi jako utjecali na aktivaciju. Prag modificira rezultat: ako rezultat neurona treba biti veći kako bi bio aktivan, rezultat se može smanjiti pragom [3]. Kao funkcija aktivacije često se je koristila funkcija sigmoide [5]:

$$f(u) = \frac{1}{1+e^{-u}}$$

dok se danas preporuča „ispravljena linearna jedinica“ (eng. *rectified linear unit*, kraće *ReLU*) [6]:

$$f(u) = \max(0, u).$$



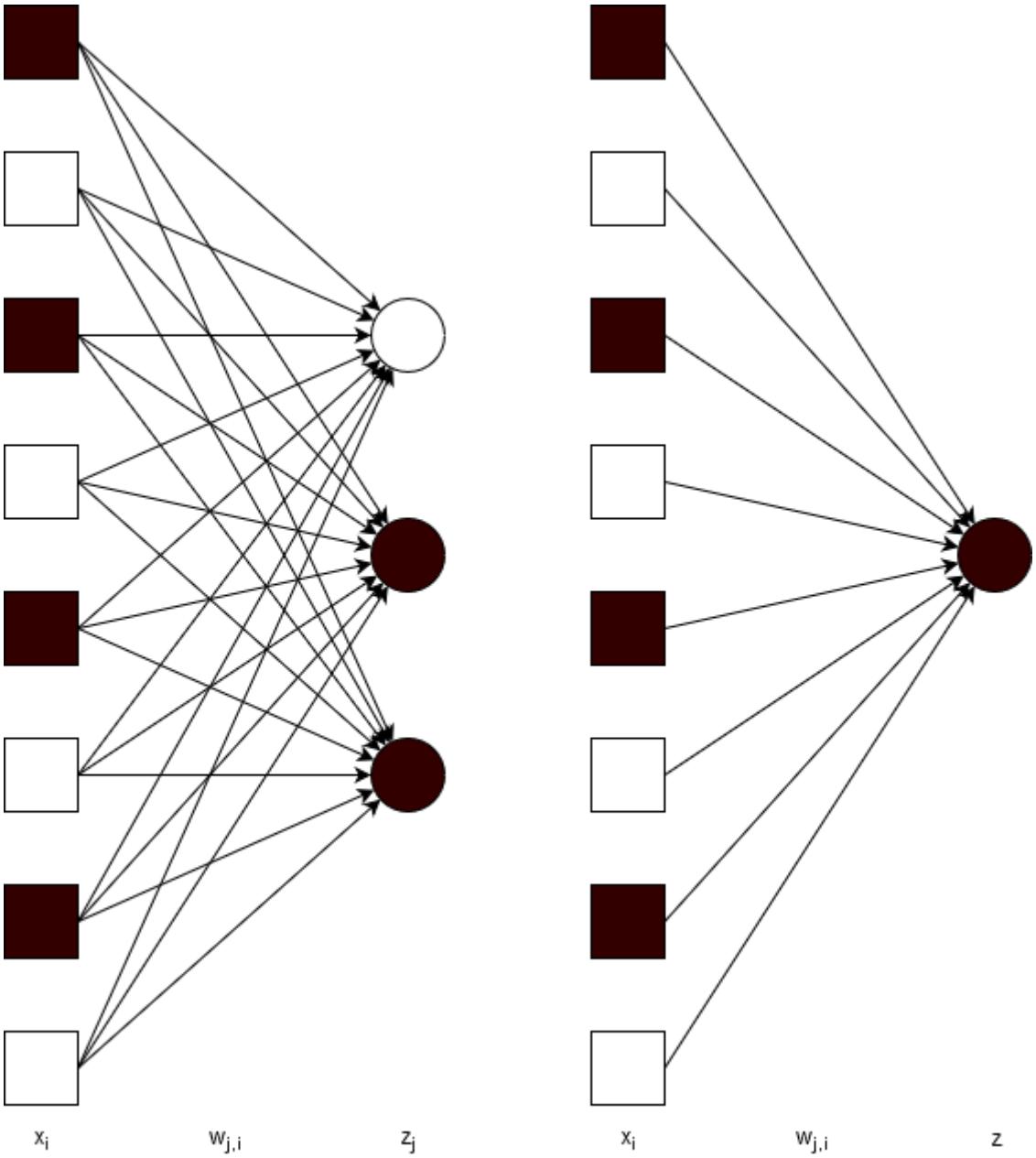
Slika 1: Primjer neurona (Prema [4])

2.3. Perceptron

Perceptron je model za nadgledano učenje. Sačinjen od jednog neurona. Koristi sljedeću aktivacijsku funkciju:

$$y(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

Rezultat te funkcije je nula ili jedan. Zbog toga, perceptron može biti koristan za binarno klasificiranje podataka [4]. No, ako podatke trebamo klasificirati na više od dvije klase, potreban nam je složeniji model. Moguće je složiti više perceptrona u mrežu perceptrona, gdje su svi ulazni podaci spojeni sa svim izlazima, te tako klasificirati podatke na više od dvije klase [2]. Ako želimo povećati preciznost mreže, potrebno je uvesti dodatne, skrivene slojeve.

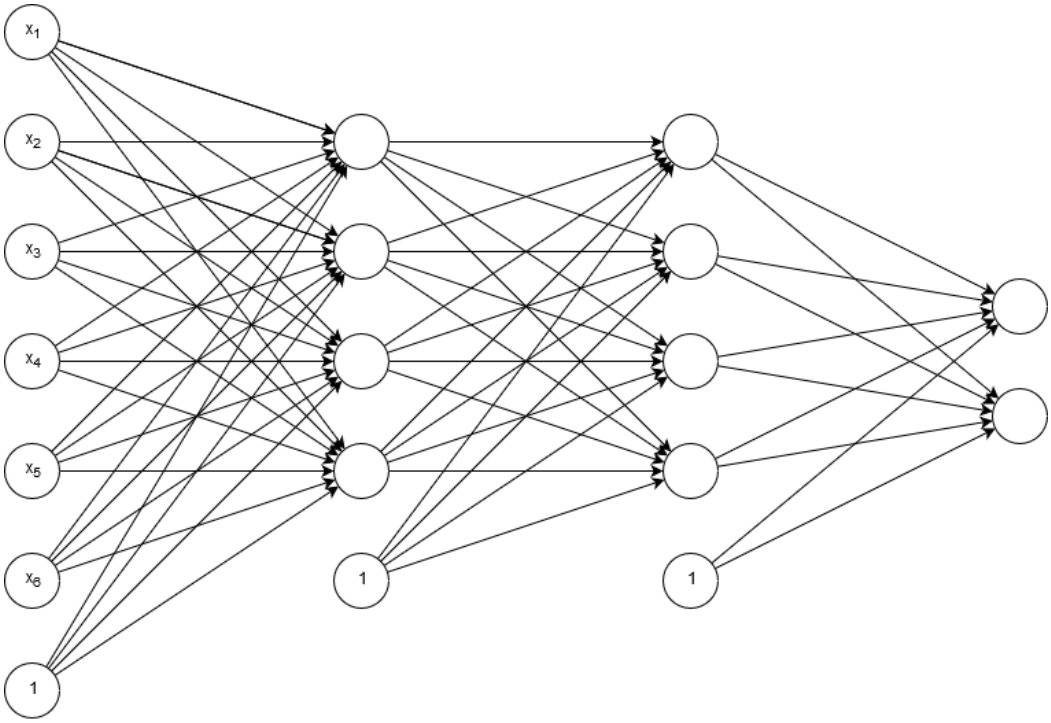


Slika 2: Mreža perceptrona (lijevo) i perceptron (desno) (prema [2])

2.4. Višeslojni perceptron

Višeslojni perceptron je unaprijedna (eng. *feedforward*) neuronska mreža. Sastoji se od ulaznog sloja, izlaznog sloja, te jednog ili više srednjih slojeva. Srednji slojevi koji čiji su neuroni povezani sa svim neuronima prošlog sloja i svim slojevima sljedećeg sloja se zovu gusti (eng. *dense*) slojevi. Svaki od neurona u sloju zbraja umnoške svih rezultata i težinskih vrijednosti prošlog sloja i svoj prag. Taj zbroj neuron stavlja u funkciju aktivacije, te svoj rezultat šalje svim neuronima sljedećeg sloja, kako bi oni napravili svoje izračune. U zadnjem, izlaznom sloju, iščitava se rezultat neuronske mreže i donosi zaključak o rezultatu. Slojevi u višeslojnom

perceptronu ne moraju imati jednak broj neurona [7]. Višeslojne perceptrone se nekad naziva i „običnim“ neuronskim mrežama, pogotovo ako imaju samo jedan skriveni sloj [8].



Slika 3: Primjer višeslojnog perceptrona sa šest ulaza, dva skrivena sloja i izlaznim slojem od dva neurona

2.5. Propagacija unatrag

Kako bi neuronska mreža „naučila“ kako točno klasificirati podatke, potrebno je podesiti pragove i težinske vrijednosti tako da što bolje smanjimo grešku između traženog i dobivenog rezultata. Najčešći način na koji se ta greška smanjuje je algoritam propagacije unatrag (eng. *backpropagation*).

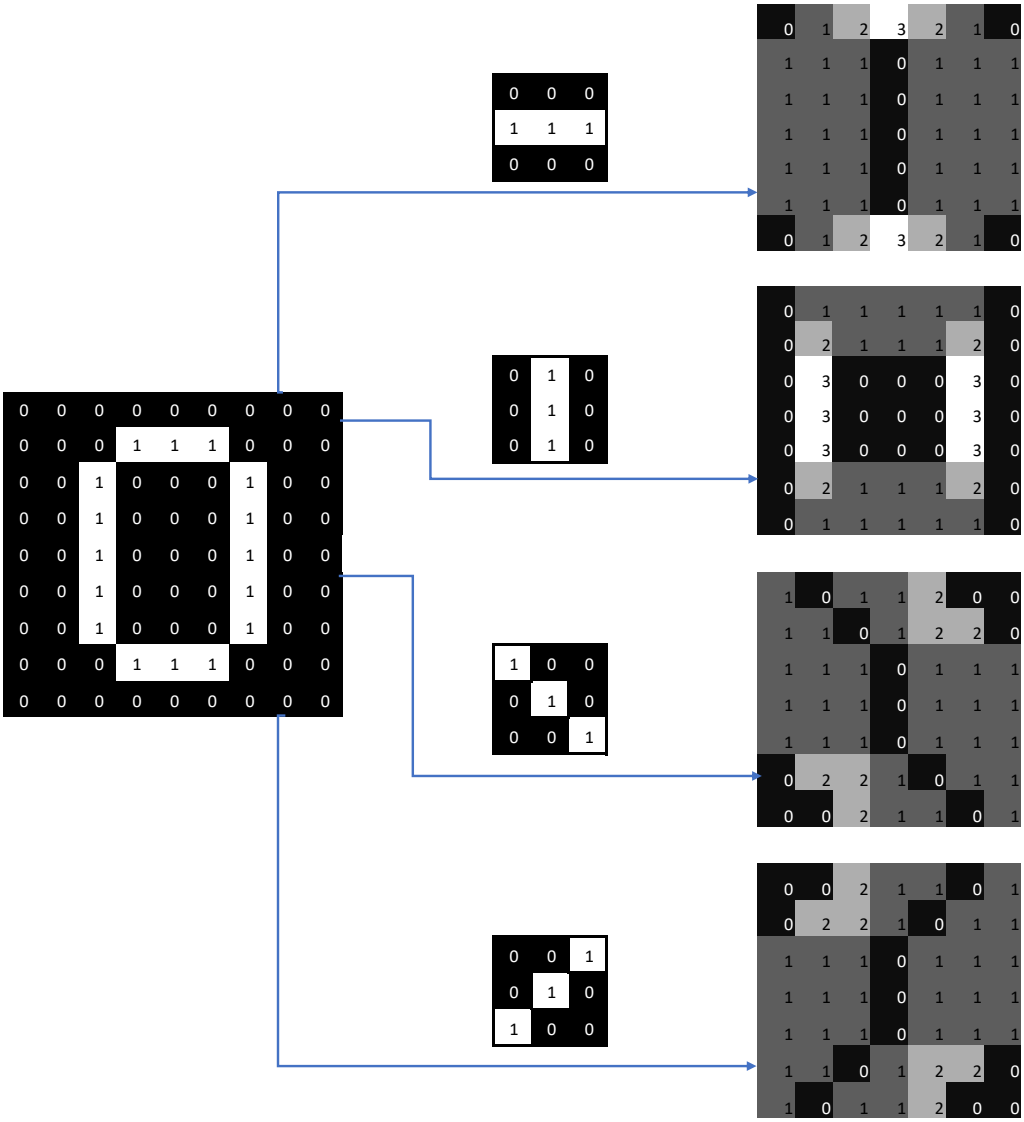
Prvo pošaljemo podatke kroz neuronsku mrežu unaprijed kako bi saznali koliko je precizna. Nakon toga, prolazimo natrag kroz neuronsku mrežu (od kud dolazi naziv „propagacija unatrag“). Dok prolazimo kroz mrežu, podešavamo težinske vrijednosti tako da sljedeće ponavljanje treninga (epoha, eng. *epoch*) daje točniji rezultat. Propagacija unatrag se ponavlja dok je greška veća od željene vrijednost. Često se pomak množi sa stopom učenja, kako se pomak ne bi prebrzo micao, te kako pomak ne bi prešao preko rješenja [2].

3. Konvolucijske neuronske mreže

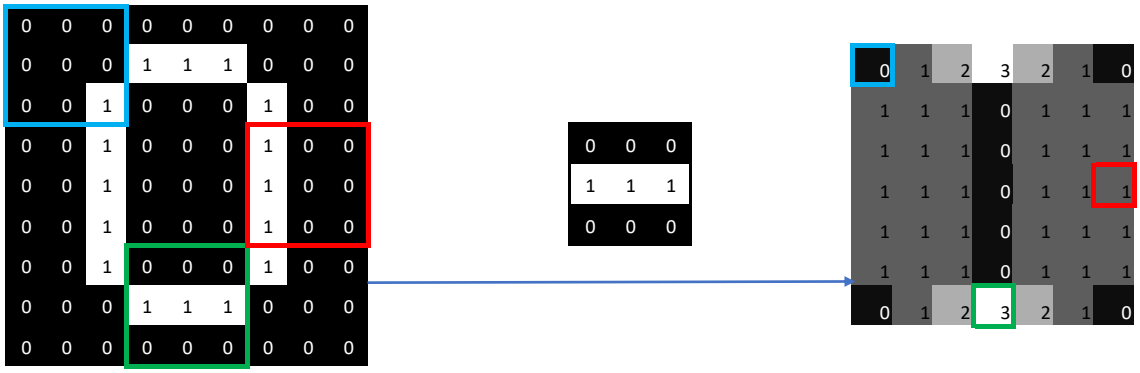
Konvolucijske neuronske mreže su mreže koje sadrže konvolucijski sloj. Korisne su za podatke koji su unaprijed poredani u obliku rešetke, npr. slike. Također mogu biti jako korisne u računalnom vidu, pošto računalo prima vizualne podatke u obliku serija slika. Konvolucija počiva na pretpostavci da su pikseli na slici koji su blizu jedan drugom povezaniji nego udaljeniji pikseli. Takve veze se ne mogu iščitati iz običnih, gustih slojeva. Konvolucijski sloj izvlači značajke slika, kao što su rubovi, oblici ili, u dubljim slojevima, cijeli objekti [6].

Na slici 4 prikazan je primjer konvolucije slike veličine 9x9 s 4 filtera (ili kernela) veličine 3x3. Filteri su postavljeni tako da izvuku neku značajku slike. Na primjeru sa slike 4, prvi filter izvlači horizontalne rubove, drugi filter izvlači vertikalne rubove, dok treći i četvrti filter izvlači dijagonalne rubove.

Filter provlačimo kroz svaki 3x3 dio slike, kao na slici 5. Prvi element dijela slike pomnožimo s prvim elementom filtera, drugi element slike s drugim elementom filtera itd. Na kraju, sve te umnoške zbrojimo. Rezultat konvolucije su više verzija slike (u ovom slučaju, četiri verzije), svaka je rezultat prolaza slike kroz pojedini filter. Dobivene slike (mape značajki, eng. *feature maps*) su manje od originalne slike (u ovom slučaju su veličine 7x7). Svaka mapa značajki prikazuje koliko je vjerojatno da ćemo pojedinu značajku naći u određenom dijelu slike [6].



Slika 4: Konvolucija slike sa četiri 3x3 filtera



Slika 5: Primjer konvolucije slike s jednim filterom, sa označenim dijelovima slike i pripadnim rezultatima

4. Izrada konvolucijske mreže u Pythonu

U ovom radu ću izraditi konvolucijsku mrežu za kategoriziranje slika u CIFAR-10 setu podataka. Mrežu ću izraditi u programskom jeziku *Python*, uz pomoć modula *Tensorflow* i *Keras*. Prije nego krenem sa izradom, potrebno je objasniti neke pojmove.

4.1. CIFAR-10

CIFAR-10 (*Canadian Institute for Advanced Research*) je set od 60000 slika dimenzije 32x32x3, sa tri kanala boja. Slike pripadaju jednoj od 10 kategorija: zrakoplov, automobil, ptica, mačka, jelen, pas žaba, konj, brod i kamion. Svaka kategorija ima 6000 slika: 5000 trening slika i 1000 test slika. Ukupno ima 50000 trening slika i 10000 test slika. Svaka slika ima i pripadnu labelu [9]. U ovom radu ću izraditi konvolucijsku neuronsku mrežu koja će klasificirati CIFAR-10 set slika.

4.2. TensorFlow i Keras

TensorFlow je biblioteka otvorenog koda za računanje koristeći grafove toka podataka. Uz pomoć biblioteke *TensorFlow*, u programskom jeziku (uglavnom *Python* ili *C++*) se opisuje graf i ulazni podaci, dok propagaciju unaprijed i unatrag obavlja biblioteka. *TensorFlow* podatke organizira u obliku tenzora. U matematici, tenzor je zapravo n-dimenzionalna „matrica“. Bitna svojstva tenzora su rang, oblik i tip. Tenzor ranga 0 (0-tenzor) je skalar:

$$t_0 = 20$$

, tenzor ranga 1 (1-tenzor) je vektor:

$$t_1 = [2, 5, 8]$$

, tenzor ranga 2 (2-tenzor) je matrica:

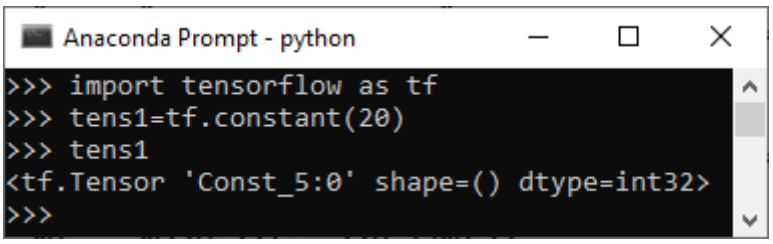
$$t_2 = \left[[2, 3, 8], [3, 5, 7], [4, 6, 1]\right]$$

dok je tenzor ranga 3 (3-tenzor) trodimenzionalna „matrica“:

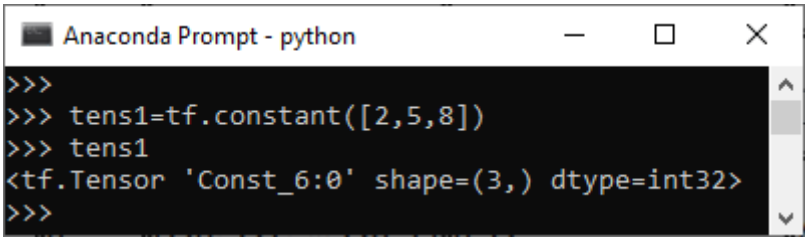
$$t_3 = \left[\left[[2, 3, 8], [3, 5, 7], [4, 6, 1]\right], \left[[8, 4, 7], [5, 9, 3], [1, 2, 6]\right], \left[[6, 3, 5], [2, 7, 1], [9, 8, 4]\right]\right].$$

Drugo bitno svojstvo tenzora je oblik. Na slikama 6, 7, 8 i 9 prikazan je oblik gore navedenih tenzora, u polju „*shape*“: slika 6 prikazuje tenzor t_0, slika 7 prikazuje tenzor t_1, slika 8 prikazuje tenzor t_2, a slika 9 prikazuje tenzor t_3. Oblik pokazuje koliko elemenata pojedinog reda se nalazi u tenzoru. Na primjer, t_0 nema oblik, t_1 ima oblik (3), što znači da sadrži tri 0-tenzora. t_2 ima oblik (3, 3), što znači da sadrži tri 1-tenzora, koji svaki sadrži tri 0-tenzora. Tenzor t_3 ima oblik (3, 3, 3), što znači da sadrži tri 2-tenzora, svaki od tih 2-tenzora sadrže tri

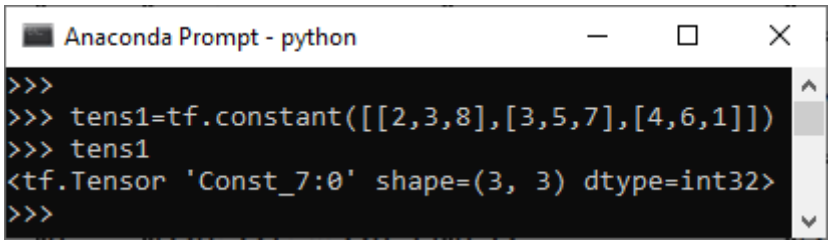
1-tenzora, dok svaki 1-tenzor još sadrži tri 0-tenzora. Zadnje bitno svojstvo je tip, što je tip 0-tenzora u tenzoru. U gornjim tenzorima tip je 32-bitni cijeli broj (eng. *integer*), što se na slikama vidi u polju „*dtype*“. Tenzor, u biblioteci *TensorFlow*, je objekt koji sadrži n-dimenzionalni tenzor određenog tipa (kao što je decimalni broj) kojem su dodane operacije [10].



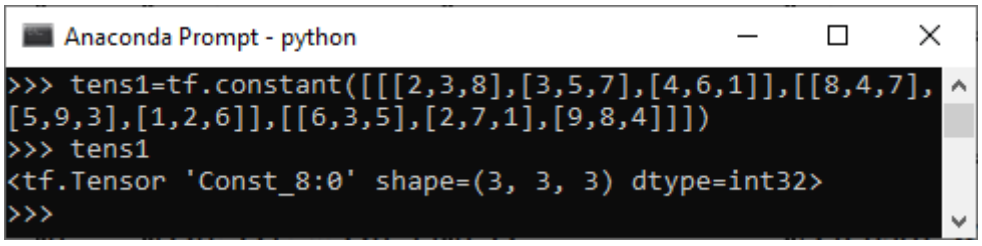
Slika 6: Oblik i tip tenzora t_0



Slika 7:Oblik i tip tenzora t_1



Slika 8: Oblik i tip tenzora t_2



Slika 9: Oblik i tip tenzora t_3

Keras je API visoke razine za različite biblioteke za strojno učenje, kao što su *TensorFlow*, *CNTK* ili *Theano*. Podiže razinu apstrakcije nad navedenim bibliotekama te još više olakšava strojno učenje.

4.3. Postavljanje radnog okruženja

Za početak, potrebno je instalirati potrebne alate. Za ovaj rad, taj alat je *Python* 3.7. Potrebno je i instalirati potrebne module za strojno učenje, uz pomoć sustava za upravljanje modulima ugrađenog u *Python*, *pip*. Prvi modul je *TensorFlow*. Moguće je instalirati „obični" *TensorFlow* koji koristi procesor za trening:

```
pip install --upgrade tensorflow
```

, no moguće je i instalirati tensorflow za rad s grafičkom karticom, jer strojnom učenju jako koristi akceleracija grafičkom karticom:

```
pip install --upgrade tensorflow-gpu
```

Nvidia grafičke kartice sadrže *CUDA* jezgre, koje ubrzavaju trening neuronskih mreži. *CUDA* je *Nvidia* platforma za paralelno računarstvo [11]. *TensorFlow* može koristiti *CUDA* platformu za ubrzanje rada. Na primjer, modelu kojem treba 73 sekunde po epohi s procesorom, potrebno je 5 sekundi po epohi s grafičkom karticom. Za akceleraciju *Nvidia* grafičkim karticama potrebno je instalirati dodatne programe [12]:

- Upravljački program za grafičku karticu
- *CUDA* toolkit
- *CUPTI*, koji dolazi s *CUDA* toolkitom
- *cuDNN* SDK

Grafička kartica koju koristim za ovaj rad je *Nvidia* RTX 2080Ti, sa 11 GB memorije i propusnošću memorije od 512 GB/s.

Koji god modul instalirali, sljedeće moramo instalirati modul *Keras*:

```
pip install --upgrade keras
```

dok je zadnji potrebni modul za ovaj rad *Pillow*, modul za rad sa slikama:

```
pip install --upgrade pillow
```

Zastavica `–upgrade` će instalirati najnoviju verziju modula i njima potrebnih modula.

4.4. Izrada modela

Sada možemo krenuti sa samom izradom neuronskih mreža. Prvo, u *Python* skriptu moramo uvesti potrebne module:

```
import keras
from keras import regularizers
from keras.datasets import cifar10
from keras.layers import Dense, Conv2D, Flatten, BatchNormalization
from keras.models import Sequential
```

Ove linije, redom, rade sljedeće: uvozi modul *keras*, iz modula `keras.datasets` uvozi set podataka CIFAR-10, iz modula `keras.layers` uvozi objekte `Dense` (gusti, potpuno povezani sloj neurona), `Conv2D` (dvodimenzionalni konvolucijski sloj), `Flatten` (prije nego što podatci prijeđu iz predzadnjeg sloja u zadnji sloj, moraju se spljoštiti u jednodimenzionalni sloj, pošto je izlazni sloj 1-tenzor) i `BatchNormalization` (normaliziranje tako da vrijednosti blizu prosjeka budu bliže 0, a standardna devijacija bliža 1), iz modula `keras.models` uvozi objekt `Sequential` (unaprijedni model).

Sljedeće, pripremaju se CIFAR-10 podatci:

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

Prva linija učitava podatke iz CIFAR-10 seta podataka u varijable. Druga i treća linija priprema labele tako da ih pretvara iz oblika skalara ($d = 3$) u oblik binarnog vektora ($[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$, treći element vektora je 1, ostali su 0).

Sljedeće linije su:

```
inputShape=(32,32,3)
brojKlasa=10
```

Prva linija nije potrebna, ali bez nje mreža mora sama izračunati oblik ulaznih podataka. Druga linija jest potrebna, kako bi mreža znala u koliko kategorija mora kategorizirati podatke.

4.4.1. Model 1

Sad napokon možemo definirati model:

```
model=Sequential()
model.add(Conv2D(32,(2,2), activation='relu', input_shape=inputShape,
padding="same"))
model.add(BatchNormalization())
model.add(Flatten(input_shape=inputShape))
model.add(Dense(units=brojKlasa, activation='softmax'))
```

U prvoj liniji koda je definiran unaprijedni model. U drugoj liniji modelu dodajemo konvolucijski sloj sa 32 filtera, veličine 2x2. Ovdje nije obavezno definirati `input_shape`, no ako želimo vidjeti izgled modela prije treninga (niže, linija `print (model.summary())`), moramo ga definirati u prvom sloju. Aktivacijska funkcijska koja se koristi je *ReLU*. `padding="same"` ovdje znači da se rezultat konvolucije neće smanjiti, nego će se pikseli koji nedostaju popuniti s nulama. Nakon konvolucijskog sloja dodajemo sloj normalizacije. Kako je sljedeći sloj izlazni sloj, podatke moramo spljoštiti, pa dodajemo `Flatten` sloj. Zadnji sloj je izlazni sloj, koji je gusti sloj. Aktivacijska funkcija je *softmax*. *Softmax* funkcija je slična sigmoidi,

samo što uzima u obzir rezultate ostalih neurona u sloju, te se koristi ako se podatci klasificiraju u više od dvije kategorije [13].

Sljedeća naredba je:

```
model.compile(loss='categorical_crossentropy',
                        optimizer=keras.optimizers.SGD(),
                        metrics=['accuracy'])
```

koja je zadnji korak prije treninga. Priprema model za trening. Ovdje se postavlja funkcija gubitka, funkcija optimizacije i metrike koje skripta ispisuje tijekom treninga. Funkcija gubitka je kategorička križna entropija (eng. *categorical crossentropy*):

$$E(k) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln (1 - y_n)\}$$

gdje je k pojedini ulaz, N broj kategorija, t_n tražena vrijednost za pojedinu kategoriju (0 ili 1), a y_n dobivena vrijednost za pojedinu kategoriju [13]. Optimizacijska funkcija koju ovdje koristimo je stohastičko spuštanje po gradijentu (eng. *stochastic gradient descent*, ili *SGD*), koja ažurira težinske vrijednosti nakon svakog ulaznog podatka, umjesto da prvo evaluira promjene u težinskim vrijednostima od K ulaza, pa onda mijenja težinske vrijednosti [13]. Metrika koja nas zanima ovdje je točnost modela nakon svake epohe.

```
print (model.summary())
```

Ova linija nije potrebna, no korisna je kako bi vidjeli izgled mreže i broj parametara koji se trebaju izračunati prije nego počnemo s treningom. Ako imamo ovu liniju, obavezno moramo imati navedeni input_shape u prvom sloju mreže. Sažetak mreže za prvi model izgleda kao na slici 10. U sažetku vidimo „*Layer (type)*“, što je *Kerasov* interni naziv tog sloja (npr. „*conv2d_35*“), a iz tog naziva možemo iščitati vrstu sloja, oblik izlaznog tenzora (*None* je rezervirano polje za broj pojedinih podataka, tj. slika koje prolaze kroz mrežu), te broj parametara koji se moraju trenirati (npr. sloj „*conv2d_5*“ ima širinu filtera 2, visinu filtera 2, dubinu boja 3, 32 filtera i 32 praga, pa je $2 \cdot 2 \cdot 3 \cdot 32 + 32 = 416$). Slojevi na slici 10 su konvolucijski sloj, sloj normalizacije, sloj spljoštivanja i gusti sloj rezultata.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_35 (Conv2D)           (None, 32, 32, 32)        416
_________________________________________________________________
batch_normalization_34 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
flatten_15 (Flatten)         (None, 32768)             0
_________________________________________________________________
dense_19 (Dense)             (None, 10)                327690
=================================================================
Total params: 328,234
Trainable params: 328,170
Non-trainable params: 64
```

Slika 10: Sažetak modela 1

Linijom:

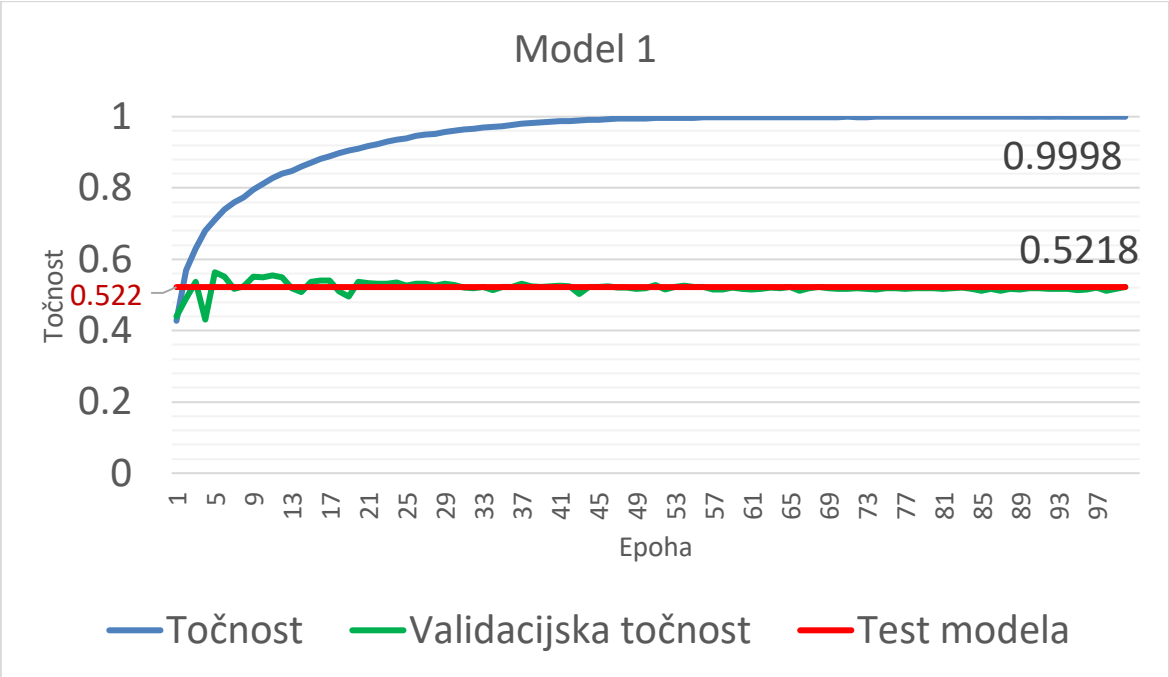
```
model.fit(x_train,y_train,epochs=100, batch_size=100,
validation_split=0.1, shuffle=True)
```

napokon počinje trening mreže. Naredbi se predaju podaci za trening (`x_train`), pripadne labele (`y_train`), broj epoha treninga, koliko je velika jedna hrpa podataka (da mreža odjednom ne trenira na cijelom setu podataka, što bi ju jako usporilo), udio test podataka koji će mreža iskoristiti za validaciju (o tome malo kasnije), te trebaju li se podaci, prije treninga, nasumično izmiješati. Na kraju testiramo model i ispisujemo njegovu točnost:

```
loss_and_metrics=model.evaluate(x_test, y_test)
print(loss_and_metrics)
```

U ovom radu ću razvijati modele postepeno, u svakom koraku dodati dio elemenata kako bi bio vidljiv utjecaj pojedinih elemenata na točnost. Svaki model ima svoje poglavlje.

Rezultat prvog modela vidljiv je na slici 11. Na rezultatu vidimo točnost modela, no vidimo i validacijsku točnost, koja je puno niža od točnosti. Točnost prikazuje koliko dobro mreža prepoznaje slike na kojima je trenirala, ovdje, 99.98%. Ovaj model dobro prepoznaje slike nad kojim trenira: od 45 000 slika, nakon treninga je točno prepoznao 44 991 sliku, ili njih 99.98%. No, dok mreži damo podatke koje nikad nije vidjela, točnost drastično pada, na 52.18%. Kako bi vidjeli kako mreža stvarno napreduje, odvajamo dio trening podataka za validaciju. Točnost nad tim podacima zovemo validacijska točnost. Ovaj model, nakon treninga, točno prepoznaje 2 609 (52.18%) validacijskih slika, od njih 5 000. Na kraju se provodi testiranje sa 10 000 testnih podataka, od kojih je mreža točno prepoznala 5 220 slika, ili 52.20%. Za trening je, u prosjeku, trebalo 4 sekundi po epohi.



Slika 11: Točnost i validacijska točnost modela 1

4.4.2. Model 2

Na modelu 2 dodajemo ispadni (eng. *Dropout*) sloj nakon normalizacije. Prvo ga moramo uvesti iz `keras.layers`:

```
from keras.layers import Dense, Conv2D, Flatten, BatchNormalization,
Dropout
```

pa ga dodajemo nakon normalizacije:

```
model.add(Dropout(0.2))
```

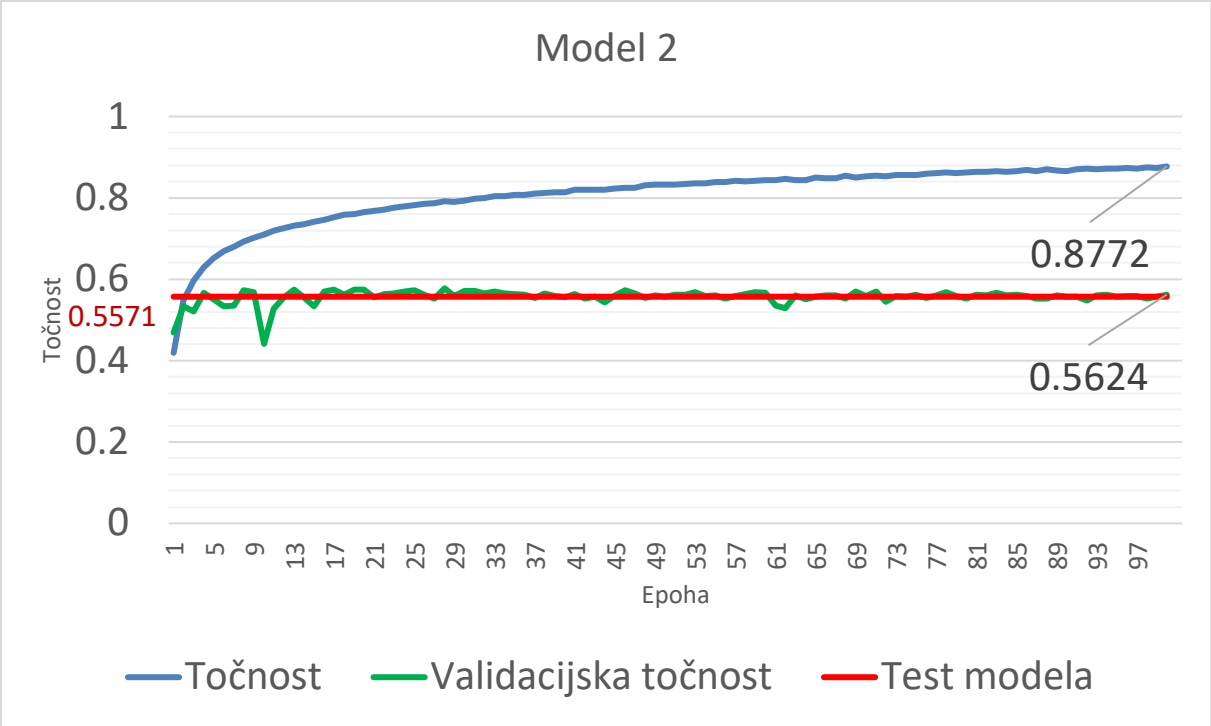
tako da model izgleda kao na slici 12.

Ispadni sloj nasumično „isključuje“ određeni postotak veza između neurona. U našem slučaju, 20% veza se svaki prolaz ignorira, kako se mreža ne bi previše naviknula na iste veze.

Rezultat modela 2 je vidljiv na slici 13. Nakon što smo dodali ispadni sloj, mreža je izgubila na točnosti nad setom za trening (87.72%), no dobila je na točnosti nad podacima koje nikad nije vidjela. Točnost modela je 55.71%, što je poboljšanje od 3.51%. Kao i za prošli model, trebalo je, u prosjeku, 4 sekunde po epohi.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_41 (Conv2D)           (None, 32, 32, 32)        416
_________________________________________________________________
batch_normalization_40 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
dropout_26 (Dropout)         (None, 32, 32, 32)        0
_________________________________________________________________
flatten_21 (Flatten)         (None, 32768)             0
_________________________________________________________________
dense_25 (Dense)             (None, 10)                327690
=================================================================
Total params: 328,234
Trainable params: 328,170
Non-trainable params: 64
_________________________________________________________________
```

Slika 12: Sažetak modela 2



Slika 13: Točnost i validacijska točnost modela 2

4.4.3. Model 3

Na model 3 dodajemo još jedan set konvolucija-normalizacija-ispad nakon prvog, s više filtera i većim filterima:

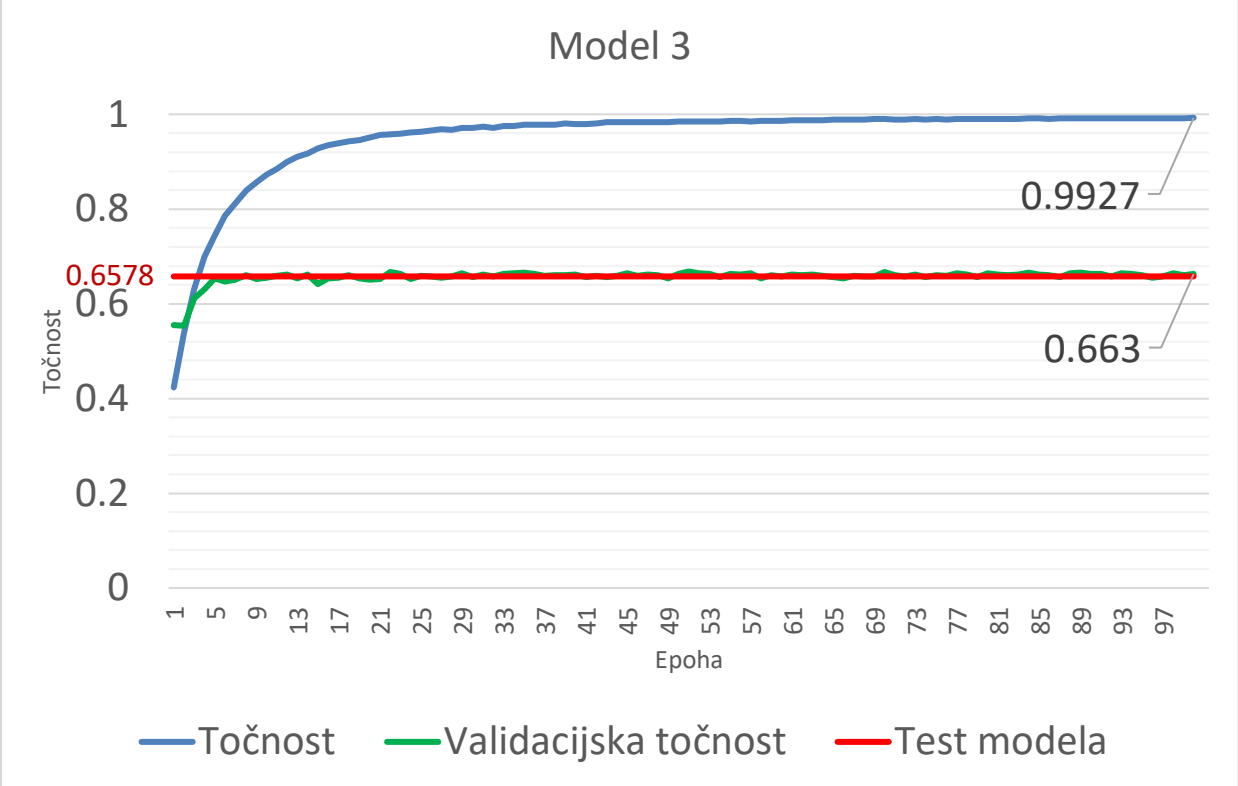
```
model.add(Conv2D(64,(4,4), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
```

Izgled modela 3 je vidljiv na slici 14. Rezultat modela 3 je vidljiv na slici 15. Opet vidimo poboljšanje u modelu. Točnost modela je 65.78%, što je poboljšanje od 10.07%. Za ovaj model je prosječno trebalo 7 sekundi po epohi.

Layer (type)	Output Shape	Param #
conv2d_45 (Conv2D)	(None, 32, 32, 32)	416
batch_normalization_44 (Batc	(None, 32, 32, 32)	128
dropout_30 (Dropout)	(None, 32, 32, 32)	0
conv2d_46 (Conv2D)	(None, 32, 32, 64)	32832
batch_normalization_45 (Batc	(None, 32, 32, 64)	256
dropout_31 (Dropout)	(None, 32, 32, 64)	0
flatten_25 (Flatten)	(None, 65536)	0
dense_29 (Dense)	(None, 10)	655370

Total params: 689,002
Trainable params: 688,810
Non-trainable params: 192

Slika 14: Sažetak modela 3



Slika 15: Točnost i validacijska točnost modela 3

4.4.4. Model 4

Ako dodamo još jedan set konvolucija-normalizacija-ispad, opet s više filtera i većim filterima, i još češćim ispadima:

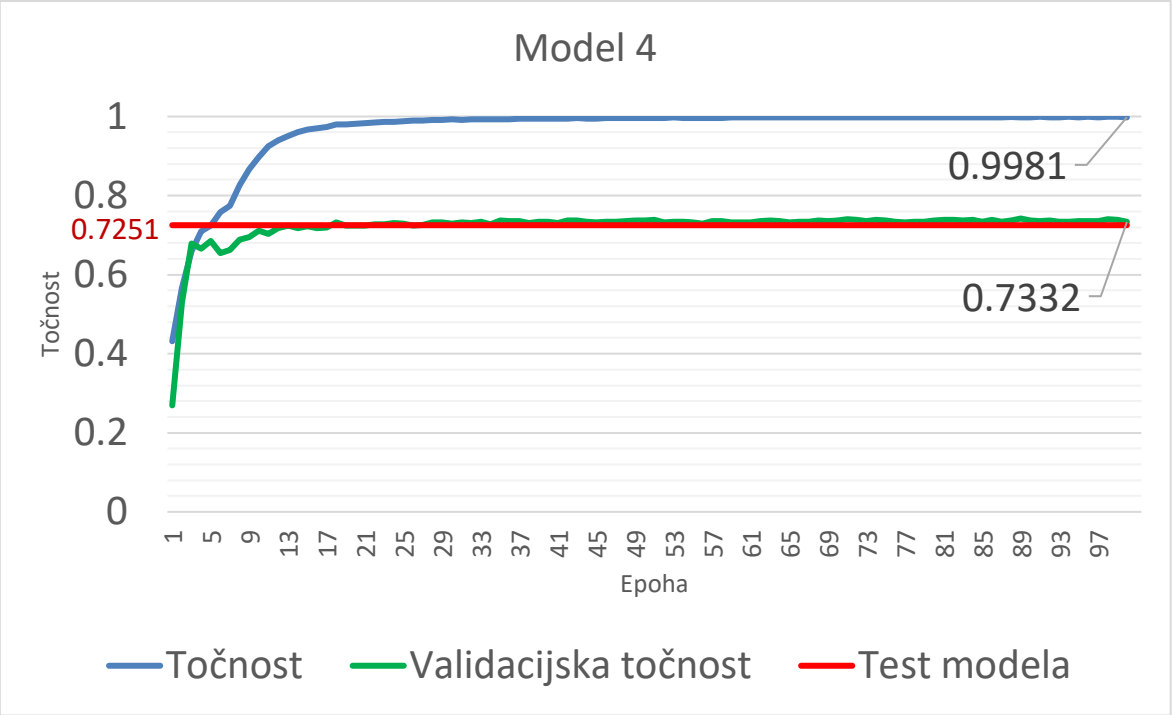
```
model.add(Conv2D(96,(6,6), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
```

tako da model 4 izgleda kao na slici 16, dobijemo rezultate na slici 17. Opet vidimo poboljšanje u točnosti. Točnost modela je 72.51%, što je poboljšanje od 6.73%. Za trening je prosječno trebalo 13 sekundi po epohi.

Layer (type)	Output Shape	Param #
conv2d_53 (Conv2D)	(None, 32, 32, 32)	416
batch_normalization_52 (Batc	(None, 32, 32, 32)	128
dropout_38 (Dropout)	(None, 32, 32, 32)	0
conv2d_54 (Conv2D)	(None, 32, 32, 64)	32832
batch_normalization_53 (Batc	(None, 32, 32, 64)	256
dropout_39 (Dropout)	(None, 32, 32, 64)	0
conv2d_55 (Conv2D)	(None, 32, 32, 96)	221280
batch_normalization_54 (Batc	(None, 32, 32, 96)	384
dropout_40 (Dropout)	(None, 32, 32, 96)	0
flatten_29 (Flatten)	(None, 98304)	0
dense_33 (Dense)	(None, 10)	983050

Total params: 1,238,346
Trainable params: 1,237,962
Non-trainable params: 384

Slika 16:Sažetak modela 4



Slika 17: Točnost i validacijska točnost modela 4

4.4.5. Model 5

Ako opet dodamo set konvolucija-normalizacija-ispad:

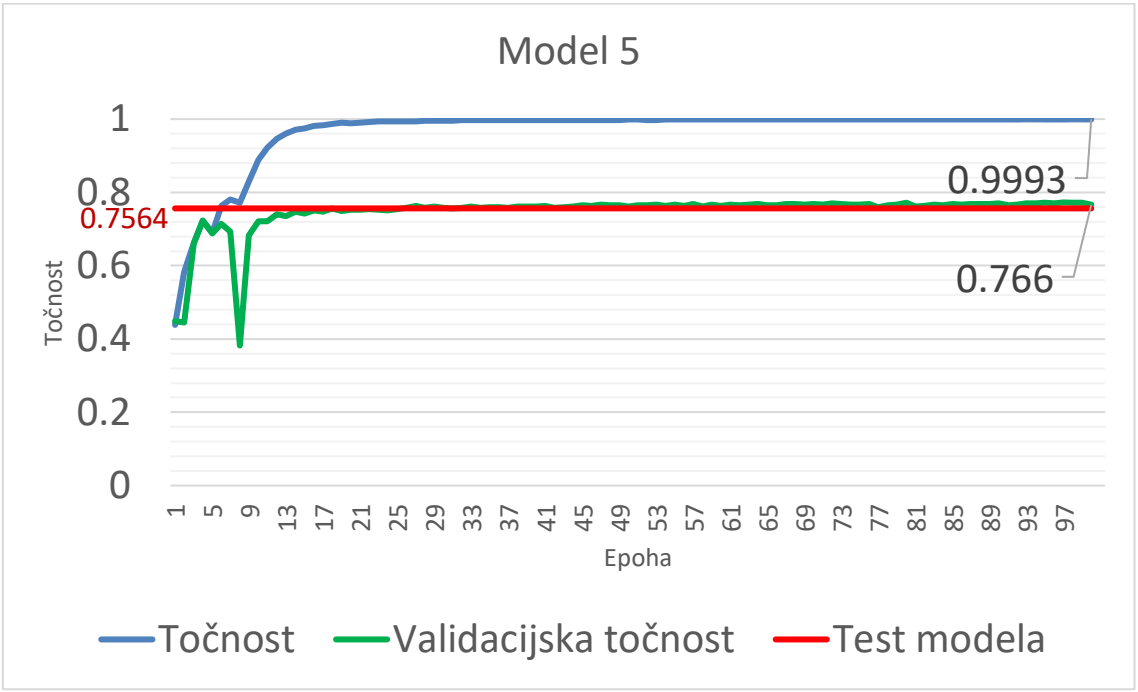
```
model.add(Conv2D(128,(8,8), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
```

tako da model izgleda kao na slici 18, dobijemo rezultat na slici 19. Točnost modela je 75.64%, što je poboljšanje od 3.13%. Za ovaj model nam je trebalo 23 sekunde po epohi. Možemo vidjeti da nam, s dodavanjem slojeva, vrijeme potrebno za trening raste brzo, dok nam točnost raste poprilično sporo.

Layer (type)	Output Shape	Param #
conv2d_56 (Conv2D)	(None, 32, 32, 32)	416
batch_normalization_55 (Batc	(None, 32, 32, 32)	128
dropout_41 (Dropout)	(None, 32, 32, 32)	0
conv2d_57 (Conv2D)	(None, 32, 32, 64)	32832
batch_normalization_56 (Batc	(None, 32, 32, 64)	256
dropout_42 (Dropout)	(None, 32, 32, 64)	0
conv2d_58 (Conv2D)	(None, 32, 32, 96)	221280
batch_normalization_57 (Batc	(None, 32, 32, 96)	384
dropout_43 (Dropout)	(None, 32, 32, 96)	0
conv2d_59 (Conv2D)	(None, 32, 32, 128)	786560
batch_normalization_58 (Batc	(None, 32, 32, 128)	512
dropout_44 (Dropout)	(None, 32, 32, 128)	0
flatten_30 (Flatten)	(None, 131072)	0
dense_34 (Dense)	(None, 10)	1310730

Total params: 2,353,098
Trainable params: 2,352,458
Non-trainable params: 640

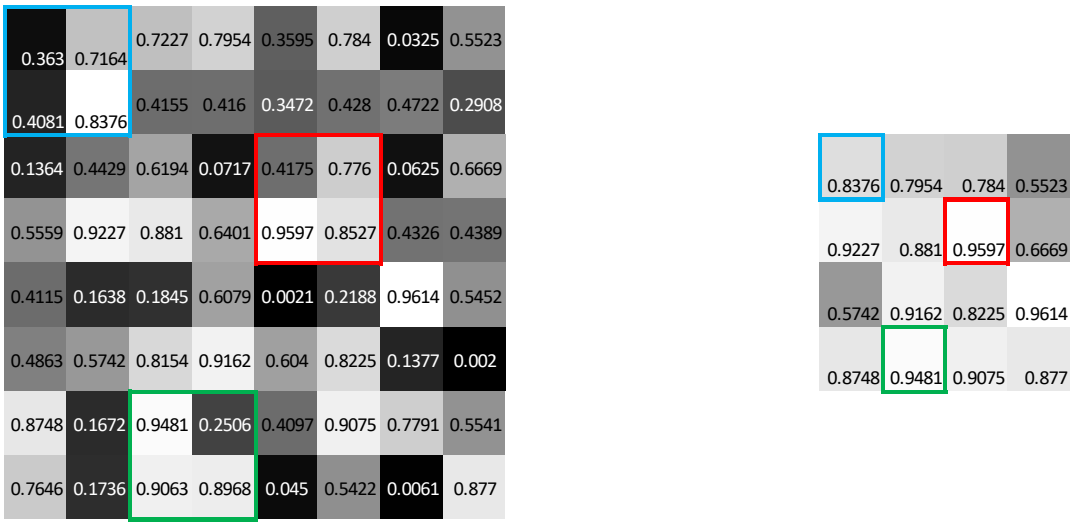
Slika 18: Sažetak modela 5



Slika 19: Točnost i validacijska točnost modela 5

4.4.6. Model 6

S obzirom na to da, s dodavanjem novih slojeva, točnost modela raste sporo, dok vrijeme treninga raste brzo, u model 6 se dodaje sloj združivanja po maksimumu (eng. *max pooling layer*). Sloj združivanja prolazi po podacima s određenom veličinom filtera $N \times N$ (npr. 2×2), uz korak M (npr. 2), nalazi najveću vrijednost u filteru, pomakne se za M i nastavlja po podacima dok ne završi. Kao rezultat imamo sliku koja je slična prošloj, ali je znatno manja. Na slici 20 vidimo primjer, gdje je slika dimenzija 8×8 smanjena na sliku 4×4, koja je 4 puta manja od originala.



Slika 20: Primjer združivanja po maksimumu i pripadni rezultati

Prvo moramo dodati `MaxPooling2D` sloj iz modula `keras.layers`:

```
from keras.layers import Dense, Conv2D, Reshape, BatchNormalization,
Flatten, MaxPooling2D
```

Nakon svakog seta konvolucija-normalizacija-ispad, osim zadnjeg, dodamo sloj združivanja po maksimumu:

```
model.add(MaxPooling2D((2,2),2))
```

Kako se nakon svakog združivanja slike smanjuju, trebamo i urediti veličine filtera na zadnjem konvolucijskom sloju:

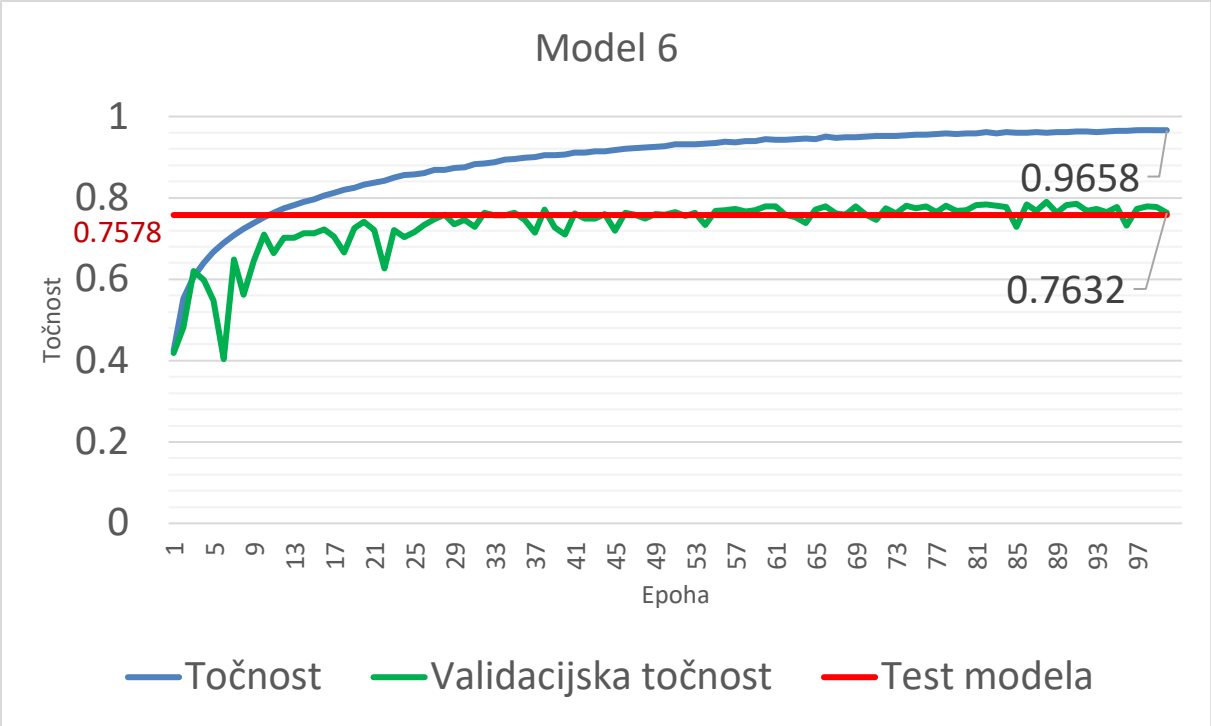
```
model.add(Conv2D(128,(2,2), activation='relu', padding="same"))
```

Nakon svih ovih promjena, model izgleda kao na slici 21. Rezultat ovog modela je vidljiv na slici 22. Ovaj rezultat (75.78%) je malo bolji od modela 5 (75.64%, što je razlika od 0.14%), ali je zato znatno brži. Dok je za model 5 trebalo 23 sekunde po epohi, za model 6 je trebalo 7 sekundi po epohi. Modelu 3 je također trebalo 7 sekundi, no njegova točnost je bila 65.78%.

Layer (type)	Output Shape	Param #
conv2d_68 (Conv2D)	(None, 32, 32, 32)	416
batch_normalization_67 (Batc	(None, 32, 32, 32)	128
dropout_53 (Dropout)	(None, 32, 32, 32)	0
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 32)	0
conv2d_69 (Conv2D)	(None, 16, 16, 64)	32832
batch_normalization_68 (Batc	(None, 16, 16, 64)	256
dropout_54 (Dropout)	(None, 16, 16, 64)	0
max_pooling2d_4 (MaxPooling2	(None, 8, 8, 64)	0
conv2d_70 (Conv2D)	(None, 8, 8, 96)	221280
batch_normalization_69 (Batc	(None, 8, 8, 96)	384
dropout_55 (Dropout)	(None, 8, 8, 96)	0
max_pooling2d_5 (MaxPooling2	(None, 4, 4, 96)	0
conv2d_71 (Conv2D)	(None, 4, 4, 128)	786560
batch_normalization_70 (Batc	(None, 4, 4, 128)	512
dropout_56 (Dropout)	(None, 4, 4, 128)	0
max_pooling2d_6 (MaxPooling2	(None, 2, 2, 128)	0
flatten_33 (Flatten)	(None, 512)	0
dense_37 (Dense)	(None, 10)	5130

Total params: 1,047,498
Trainable params: 1,046,858
Non-trainable params: 640

Slika 21: Sažetak modela 6



Slika 22: Točnost i validacijska točnost modela 6

4.4.7. Model 7

Sljedeći parametri koje možemo promijeniti su stopa učenja i opadanje stope učenja. U modulu *Keras*, u stohastičkom spuštanju po gradijentu, zadana stopa učenja je 0.01. To možemo promijeniti tako da promijenimo:

```
optimizer=keras.optimizers.SGD(),
```

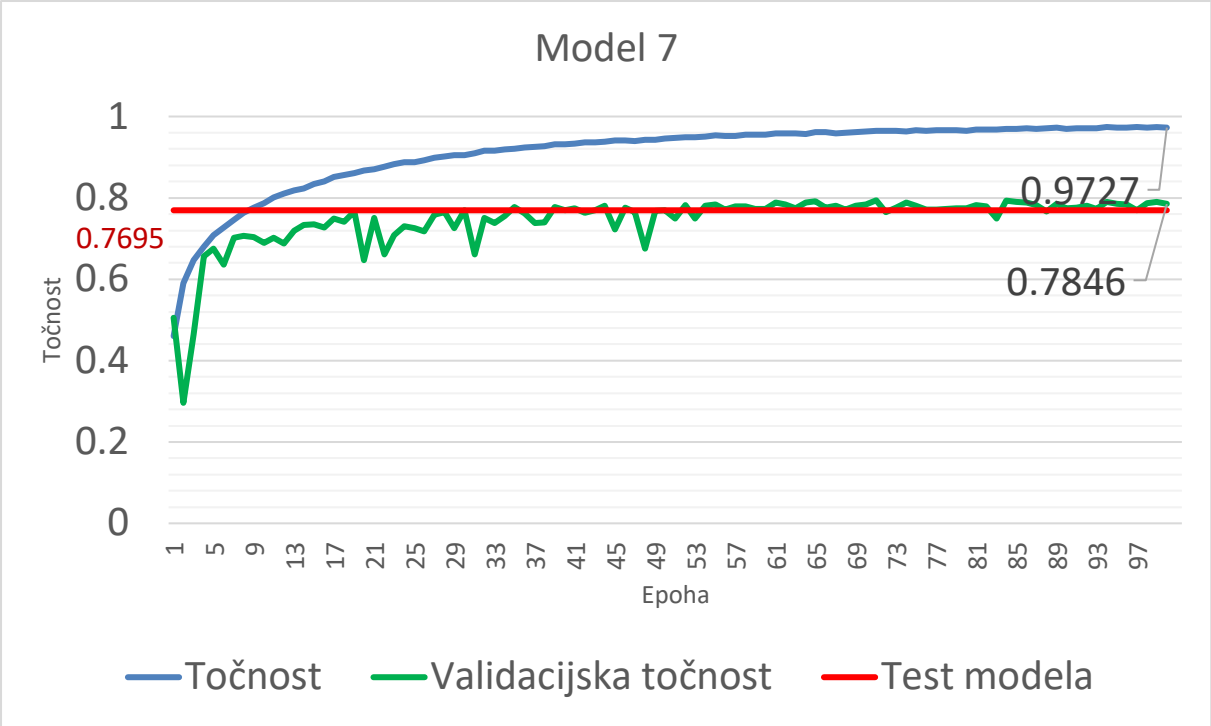
u:

```
optimizer=keras.optimizers.SGD(lr=0.02, decay=0.00001),
```

Stopa učenja opada prema sljedećoj formuli [14]:

$$lr = origlr \cdot \frac{1}{1 + decay * epoch}$$

gdje su $origlr$ originalna stopa učenja i atribut koji u *Python* kodu zovemo lr, $decay$ stopa opadanja stope učenja, a $epoch$ broj epohe. Nakon što ažuriramo kod, model ne mijenja izgled, tj. izgled je jednak kao i izgled modela 6, sa slike 21,a rezultat dobivamo sa slike 23. Točnost modela je 76.95%, što je poboljšanje od 1.17%. Za trening je trebalo 7 sekundi po epohi.



Slika 23: Točnost i validacijska točnost modela 7

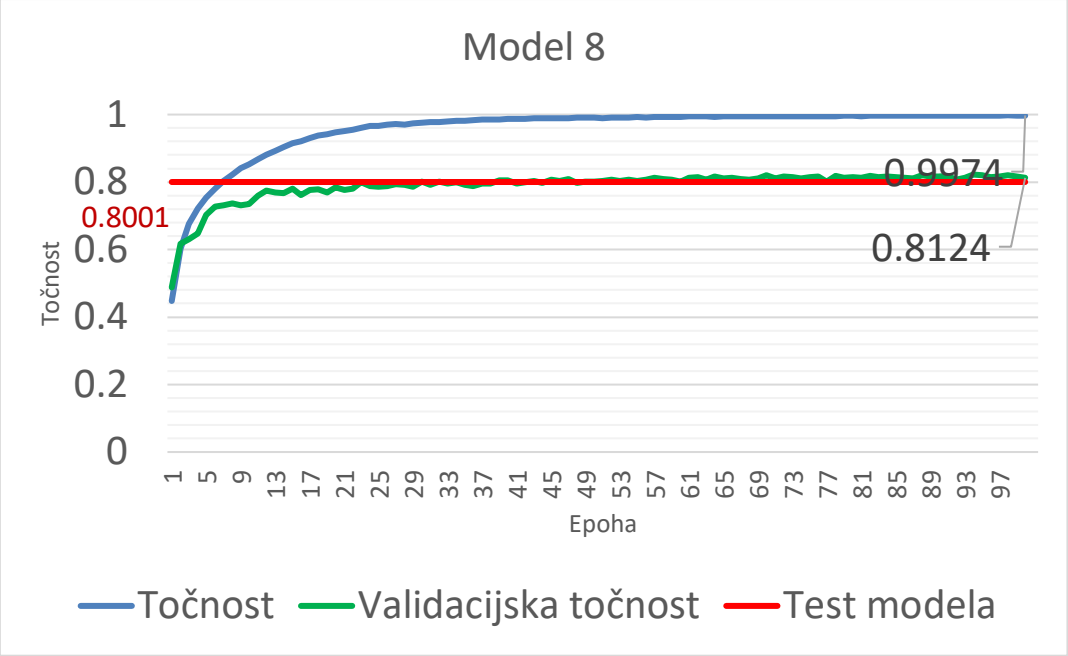
4.4.8. Model 8

Možemo pokušati produbiti mrežu, tako da udvostručimo svaki konvolucijski sloj, tako da slojevi izgledaju ovako:

```
model.add(Conv2D(32,(2,2), activation='relu', input_shape=inputShape, padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(32,(2,2), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(MaxPooling2D((2,2),2))
model.add(Conv2D(64,(4,4), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(64,(4,4), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(MaxPooling2D((2,2),2))
model.add(Conv2D(96,(6,6), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(96,(6,6), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(MaxPooling2D((2,2),2))
```

```
model.add(Conv2D(128,(2,2), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(128,(2,2), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Flatten(input_shape=inputShape))
model.add(Dense(units=brojKlasa, activation='softmax'))
```

Izgled modela 8 je kao na slici 27, te se izgled dalje neće mijenjati. Rezultat ovog modela je vidljiv na slici 24. Točnost modela je 80.01%, što je znatno povećanje točnosti (3.06%), bez utjecaja na prosječno trajanje treninga, koji je još uvijek 7 sekundi po epohi.



Slika 24: Točnost i validacijska točnost modela 8

4.4.9. Model 8 s generatorom podataka

Zadnje što možemo učiniti da poboljšamo točnost modela je povećati broj ulaznih podataka. Jedan od načina da to učinimo je da uvedemo *Keras* generator podataka. Prvo moramo `ImageDataGenerator`:

```
datagen=keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15,
    horizontal_flip=True,
    validation_split=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1
)
```

`ImageDataGenerator`u moramo zadati koje transformacije će raditi nad podacima. `ImageDataGenerator` uzima jednu sliku te nad slikom provodi jednu od navedenih transformacija. U ovom slučaju, `ImageDataGenerator` može zarotirati sliku između -15 i 15 stupnjeva, horizontalno ju zrcaliti, pomaknuti lijevo-desno 10 posto ili pomaknuti gore-dolje 10 posto. Parametar `validation_split` određuje udio slika koje će se koristiti za validaciju. Sljedeće, moramo definirati u kojoj varijabli će se podaci nalaziti:

```
trainDatagen=datagen.flow(x_train, y_train, batch_size=100,
subset="training")
validateDatagen=datagen.flow(x_train, y_train, batch_size=100,
subset="validation")
```

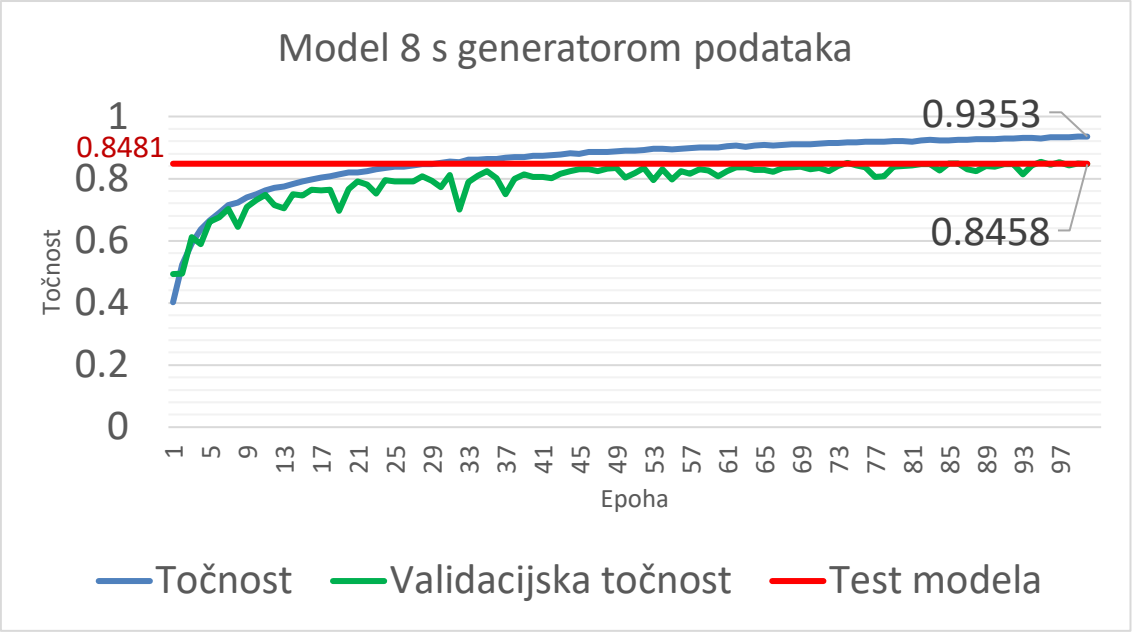
`Flow()` metodi `ImageDataGenerator`a dajemo trening podatke i labele, koliko slika će se generirati za jednu hrpu, te kojem setu pripada. Na kraju, naredbu:

```
model.fit(x_train,y_train,epochs=100, batch_size=100,
validation_split=0.1, shuffle=True)
```

mijenjamo s naredbom:

```
model.fit_generator(trainDatagen,
                    validation_data=validateDatagen,
                   steps_per_epoch=450,validation_steps=50, epochs=100)
```

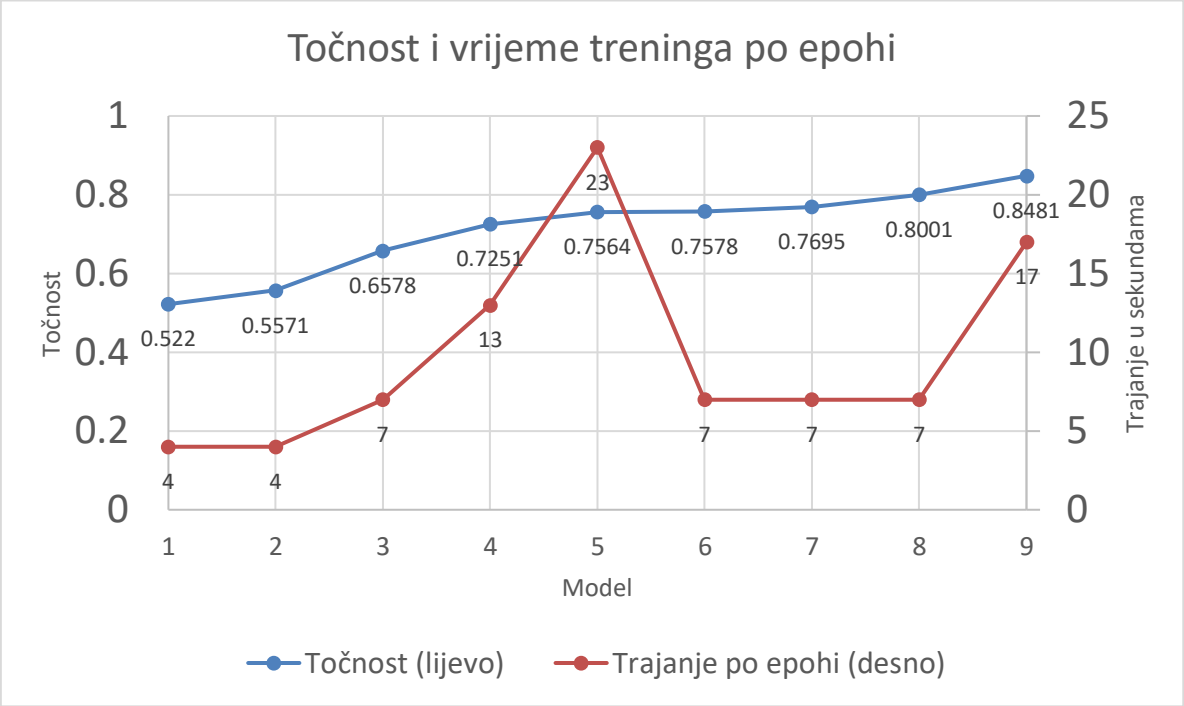
Definiramo trening podatke, validacijske podatke, koliko hrpa slika je potrebno za trening po epohi, koliko hrpa je potrebno za validaciju po epohi, te koliko epoha traje trening. Sama neuronska mreža ostaje ista. Izgled modela se nije mijenjao, te je jednak slici 27. Rezultat s generiranim podatcima je vidljiv na slici 25. Točnost modela je 84.81%, što je povećanje od 4.8%. Točnost je znatno veća od svih prijašnjih modela, ali je i trajanje treninga puno veće, zbog generiranja slika: za trening je potrebno 17 sekundi po epohi.



Slika 25: Točnost i validacijska točnost modela 8 s generatorom podataka

4.5. Usporedba modela

Na kraju možemo usporediti sve modele na jednom grafu, kao na slici 26. Najveći utjecaj na točnost neuronske mreže je imalo dodavanje novih slojeva i generiranje novih podataka. Najveći utjecaj na trajanje su imali slojevi združivanja dodani u modelu 6, koji su smanjili vrijeme treninga po epohi za 16 sekundi, i generiranje novih podataka, koji su povećali vrijeme treninga po epohi za 10 sekundi.



Slika 26: Usporedba modela po točnosti i trajanju treninga po epohi

Dizajn finalne konvolucijske neuronske mreže je vidljiv na slici 27. Mreža sadrži konvolucijski sloj i normalizacijski sloj, te još jedan konvolucijski sloj i normalizacijski sloj. Sljedeći sloj je sloj ispadanja, pa sloj združivanja. Nakon združivanja opet imamo konvoluciju, normalizaciju, konvoluciju i normalizaciju, te još jedno ispadanje i združivanje. Po treći put imamo konvoluciju, normalizaciju, konvoluciju, normalizaciju, ispadanje i združivanje. Sve opet imamo po četvrti put, samo što ovaj put ne dodajemo sloj združivanja. Na kraju se model spljošti, te se dodaje zadnji sloj. Zadnja mreža ima 792 842 parametara koji se mogu podesiti. Njezina preciznost je 84.81% nakon sto epoha, a trening je trajao 17 sekundi po epohi, što je 1700 sekundi, ili 28 minuta i 20 sekundi.

Layer (type)	Output Shape	Param #
conv2d_249 (Conv2D)	(None, 32, 32, 32)	416
batch_normalization_249 (Bat	(None, 32, 32, 32)	128
conv2d_250 (Conv2D)	(None, 32, 32, 32)	4128
batch_normalization_250 (Bat	(None, 32, 32, 32)	128
dropout_122 (Dropout)	(None, 32, 32, 32)	0
max_pooling2d_92 (MaxPooling	(None, 16, 16, 32)	0
conv2d_251 (Conv2D)	(None, 16, 16, 64)	32832
batch_normalization_251 (Bat	(None, 16, 16, 64)	256
conv2d_252 (Conv2D)	(None, 16, 16, 64)	65600
batch_normalization_252 (Bat	(None, 16, 16, 64)	256
dropout_123 (Dropout)	(None, 16, 16, 64)	0
max_pooling2d_93 (MaxPooling	(None, 8, 8, 64)	0
conv2d_253 (Conv2D)	(None, 8, 8, 96)	221280
batch_normalization_253 (Bat	(None, 8, 8, 96)	384
conv2d_254 (Conv2D)	(None, 8, 8, 96)	331872
batch_normalization_254 (Bat	(None, 8, 8, 96)	384
dropout_124 (Dropout)	(None, 8, 8, 96)	0
max_pooling2d_94 (MaxPooling	(None, 4, 4, 96)	0
conv2d_255 (Conv2D)	(None, 4, 4, 128)	49280
batch_normalization_255 (Bat	(None, 4, 4, 128)	512
conv2d_256 (Conv2D)	(None, 4, 4, 128)	65664
batch_normalization_256 (Bat	(None, 4, 4, 128)	512
dropout_125 (Dropout)	(None, 4, 4, 128)	0
flatten_32 (Flatten)	(None, 2048)	0
dense_38 (Dense)	(None, 10)	20490

Total params: 794,122
Trainable params: 792,842
Non-trainable params: 1,280

Slika 27: Finalna konvolucijska neuronska mreža

5. Zaključak

Konvolucijske mreže su danas jako korisne u području računalnog vida. Jedno od područja gdje su danas jako korisne je samovozeći automobili. Samovozeći automobili koriste konvolucijske neuronske mreže kako bi prepoznali objekte na cesti te kako ne bi došlo do sudara [15]. Neke od drugih polja gdje se konvolucijske neuronske mreže koriste su optičko prepoznavanje slova i brojeva (eng. *optical character recognition*), prepoznavanje radnji, prepoznavanje govora, praćenje objekata, itd. [16].

Finalna konvolucijska mreža u ovom radu ima točnost od 84.81%. Kako je najbolji rezultat klasifikacije CIFAR-10 podataka 99% [17] (doduše, uz korištenje naprednijih modela), ovaj model se definitivno može poboljšati. Moguće promjene su podešavanje parametara kao što su stopa učenja ili stopa ispadanja, promjena nekih slojeva, dodavanje gustih slojeva, itd..

Alati koji su korišteni za ovaj rad su *Microsoft Word, draw.io*, *Zotero*, *Microsoft Excel* i *Python* 3.7 i moduli *TensorFlow*, *Keras* i *Pillow*.

Popis literature

[1] A. P. Engelbrecht, *Computational Intelligence. An Introduction*, 2nd ed. Wiley, 2007.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 1st ed. Prentice Hall, 1995.

[3] M. T. Hagan, H. B. Demuth, and M. H. Beale, *Neural Network Design*, Har/Dis. PWS Pub. Co., 1995.

[4] Y. H. Hu and J.-N. Hwang, *Handbook of neural network signal processing*, 1st ed. CRC Press, 2002.

[5] S.-C. Wang, *Interdisciplinary Computing in Java Programming*. Boston, MA: Springer US, 2003.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, Draft. The MIT Press, 2016.

[7] T. Oladipupo, "Types of Machine Learning Algorithms," in *New Advances in Machine Learning*, Y. Zhang, Ed. InTech, 2010.

[8] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*, 2nd ed. 2009. Corr. 3rd printing 5th Printing. Springer, 2009.

[9] "CIFAR-10 and CIFAR-100 datasets." [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html. [Accessed: 18-Aug-2019].

[10] R. Bonnin, *Building Machine Learning Projects with TensorFlow*. Packt Publishing, 2016.

[11] "CUDA Zone," *NVIDIA Developer*, 18-Jul-2017. [Online]. Available: https://developer.nvidia.com/cuda-zone. [Accessed: 20-Aug-2019].

[12] "GPU support," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/install/gpu. [Accessed: 20-Aug-2019].

[13] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. 2006. Corr. 2nd printing. Springer, 2006.

[14] *keras/optimizers.py at master · keras-team/keras · GitHub*. Keras, 2019.

[15] A. Lai, "How do Self-Driving Cars See?," *Medium*, 15-Dec-2018. [Online]. Available: https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503. [Accessed: 20-Aug-2019].

[16] J. Gu *et al.*, "Recent Advances in Convolutional Neural Networks," *arXiv:1512.07108 [cs]*, Dec. 2015.

[17] Y. Huang *et al.*, "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism," *arXiv:1811.06965 [cs]*, Nov. 2018.

Popis slika

Prilog 1: Izvorni kod završne konvolucijske neuronske mreže

```
import keras
from keras.datasets import cifar10
from keras.layers import Dense, Conv2D, BatchNormalization, Flatten, MaxPooling2D, Dropout
from keras.models import Sequential

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

datagen=keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15,
    horizontal_flip=True,
    validation_split=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1
)

trainDatagen=datagen.flow(x_train, y_train, batch_size=100, subset="training")
validateDatagen=datagen.flow(x_train, y_train, batch_size=100, subset="validation")

inputShape=(32,32,3)
brojKlasa=10
model=Sequential()

model.add(Conv2D(32,(2,2), activation='relu',input_shape=inputShape, padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(32,(2,2), activation='relu',input_shape=inputShape, padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(MaxPooling2D((2,2),2))
model.add(Conv2D(64,(4,4), activation='relu', padding="same"))
model.add(BatchNormalization())
```

```
model.add(Conv2D(64,(4,4), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(MaxPooling2D((2,2),2))
model.add(Conv2D(96,(6,6), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(96,(6,6), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(MaxPooling2D((2,2),2))
model.add(Conv2D(128,(2,2), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(128,(2,2), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Flatten(input_shape=inputShape))
model.add(Dense(units=brojKlasa, activation='softmax'))

model.compile(loss='categorical_crossentropy',
                      optimizer=keras.optimizers.SGD(lr=0.02,
decay=0.00001),
                      metrics=['accuracy'])

print (model.summary())

model.fit_generator(trainDatagen,
                     validation_data=validateDatagen,
                    steps_per_epoch=450,validation_steps=50, epochs=20)

loss_and_metrics=model.evaluate(x_test,y_test)
print(loss_and_metrics)
```