Primjena duboke neuronske mreže u raspoznavanju objekata

Žalac, Josip

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij Računarstva

PRIMJENA DUBOKE NEURONSKE MREŽE U RASPOZNAVANJU OBJEKATA

Diplomski rad

Josip Žalac

Osijek, 2019

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 23.09.2019.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Josip Žalac
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 906 R, 28.09.2018.
OIB studenta:	41484786583
Mentor:	Doc.dr.sc. Emmanuel Karlo Nyarko
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Damir Filko
Član Povjerenstva:	Petra Đurović
Naslov diplomskog rada:	Primjena duboke neuronske mreže u raspoznavanju objekata
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Treba istražiti različite primjena duboke neuronske mreže u raspoznavanju objekata. Implementirati jednu takvu mrežu za primjenu na osobnim računalima, optimirati hiperparametre mreže te vrednovati učinak naučene mreže na potpunom označenom skupu različitih objekata.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 1 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	23.09.2019.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 14.10.2019.

Ime i prezime studenta:	Josip Žalac
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 906 R, 28.09.2018.
Ephorus podudaranje [%]:	2%

Ovom izjavom izjavljujem da je rad pod nazivom**: Primjena duboke neuronske mreže u raspoznavanju objekata**

izrađen pod vodstvom mentora Doc.dr.sc. Emmanuel Karlo Nyarko

i sumentora

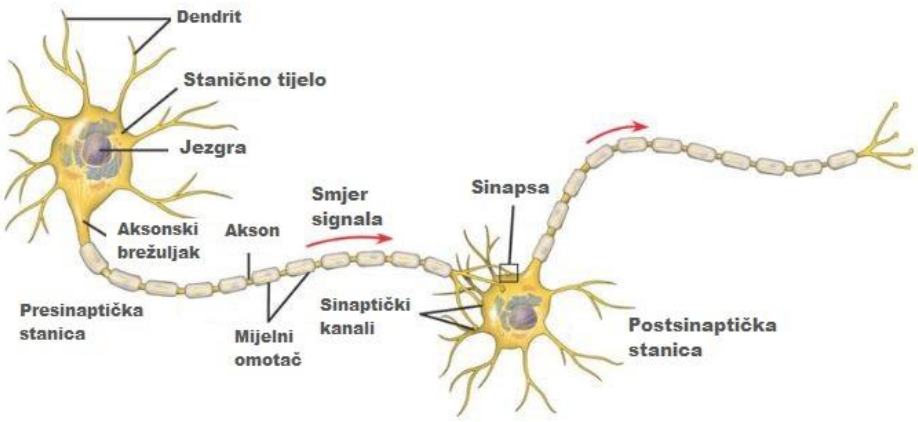
moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Kroz povijest ljudi su bili inspirirani prirodom i stvarima koje nas okružuju, što je dovelo do raznih izuma i napretka ljudske civilizacije. U 21. stoljeću, vremenu inteligentnih strojeva i umjetne inteligencije, ne postoji bolja inspiracija od samog ljudskog mozga. Davne 1943. godine nekoliko je znanstvenika došlo na ideju napraviti matematički model čija je zadaća bila upravo to, imitirati rad ljudskog mozga. Naravno, taj model, kao ni modeli koji se upotrebljavaju u današnje vrijeme, se ne može u potpunosti usporediti sa ljudskim mozgom, iako se znanstvenici polako približavaju tome cilju. Neuronske mreže su algoritmi inspirirani upravo radom ljudskog mozga, sa zadatkom učenja određenih značajki ili uzoraka. U suštini, neuronske mreže su skup umjetnih neurona modeliranih na sliku biološkog neurona, gdje je svaki neuron povezan putem sinapsi u svrhu propagacije signala.



Slika 1.1. *Prikaz biološkog neurona [1]*

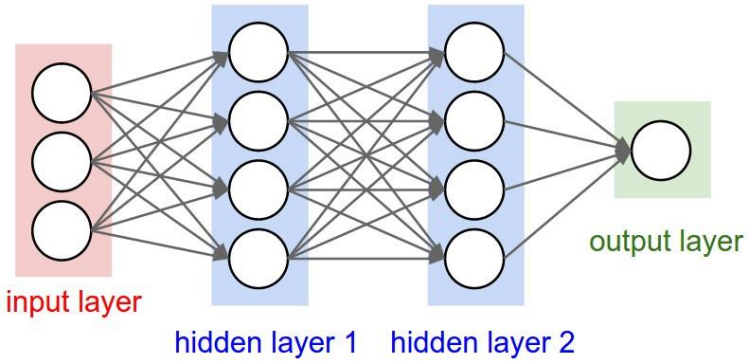
Neuronske mreže se upotrebljavaju za rješavanje raznih problema u poljima računalnog vida, prepoznavanja govora, strojnog prevođenja i sl. S vremenom, problemi su postali sve zahtjevniji te se iz tog razloga počinju upotrebljavati dublje i kompleksnije mreže, s velikim brojem slojeva i neurona kako bi pratili razvoj kompleksnosti problema.

U svrhu upoznavanja s dubokim mrežama, u nastavku ovoga rada su pomno objašnjene njihove arhitekture, način rada te njihova implementacija u praksi. U sljedećem poglavlju će se teorijski obraditi konvolucijske neuronske mreže, njihovu arhitekturu i način rada te usporediti s klasičnim unaprijednim mrežama. U nastavku se govori o optimizaciji mreža te transfer učenju kao jednom od bitnijih polja dubokog učenja. Eksperimentalni dio ovoga rada se sastoji od implementacije već prethodno naučenih modela i evaluacije na skupu podataka, te kreiranje novog modela u svrhu evaluacije i usporedbe s već prethodno naučenim modelima.

2. KONVOLUCIJSKE NEURONSKE MREŽE

Konvolucijske neuronske mreže su podvrsta neuronskih mreža koje se upotrebljavaju u dubokom učenju najčešće za analizu i obradu slike. Također, poput unaprijednih mreža, inspiracija dolazi s biološke strane, točnije, organizacija neurona unutar mreže slična je onoj u životinjskom vizualnom korteksu. Arhitektura mreže te način rada biti će detaljnije objašnjeni u nastavku ovog poglavlja. No prije toga, kako bi se dobio bolji uvid u rad konvolucijske neuronske mreže, radi usporedbe napravit će kratka digresija o unaprijednim neuronskim mrežama.

Ideja iza neuronske mreže oduvijek je bila ista. Na temelju nekog skupa podataka cilj je napraviti predikcijski model za prepoznavanje uzoraka ili slika. Prilikom treniranja modela na ulaz mreže se dovede slika u obliku vektora te kao takva prosljedi kroz cijelu mrežu. Klasična unaprijedna mreža se sastoji od neurona i raznih aktivacijskih funckija unutar neurona, te različitih slojeva koji su međudosbno povezani kao što je prikazano na slici 2.1.

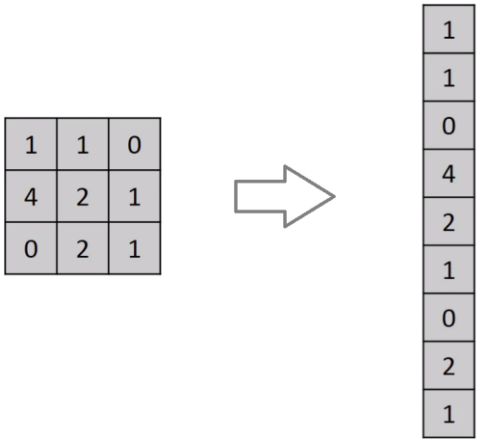


Slika 2.1. *Prikaz troslojne unaprijedne mreže [1]*

Kada se govori o konvolucijskim neuronskim mrežama, može se reći da su u posljednjih nekoliko godina najzaslužnije za napredak u poljima računalnog vida i obradi prirodnih jezika. Usporede li se s klasičnim unaprijednim mrežama, primjeti se niz sličnosti, od samih elemenata pa do principa rada mreže. Međutim, razlika nastaje u arhitekturi konvolucijske mreže koja je prilagođena obradi slika i samim time puno efikasnija s manjim brojem parametara. Koncept konvolucijske mreže te njena arhitektura objašnjeni su detaljnije u sljedećem poglavlju.

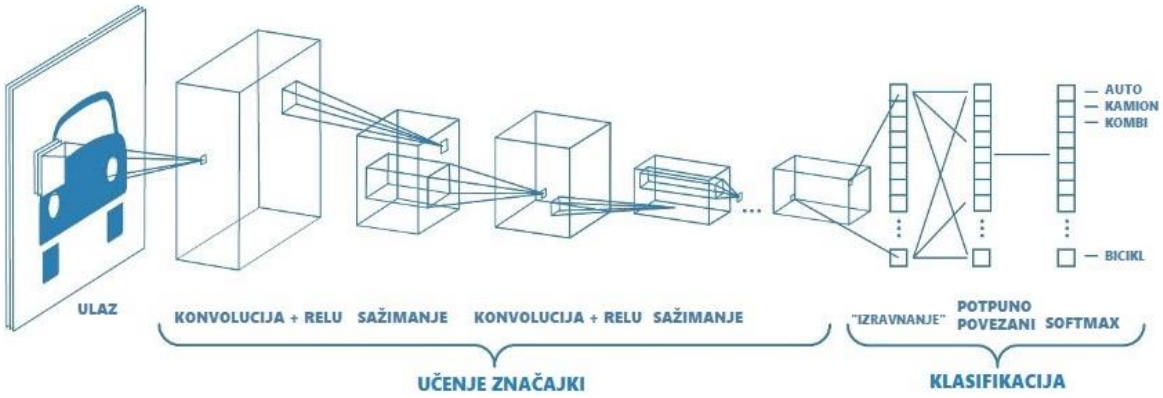
2.1. Arhitektura konvolucijske neuronske mreže

U prijašnjem poglavlju je spomenuto da unaprijedne mreže prime neki ulazni podatak u obliku vektora te ga kroz niz skrivenih slojeva mreže transformiraju. Pretpostavit će se da je na ulaz dovedena slika dimenzija 3x3x1 (3 piksela širine i visine, 1 kanal za boju), ta slika se prilikom dovođenja na ulaz mreže mora pretvoriti u vektor dimenzija 9x1 kao što je prikazano na slici 2.2.



Slika 2.2. *Pretvaranje matrice ulaznog podatka u vektor [4]*

Problem nastaje ukoliko se upotrebljava slika puno većih dimenzija jer tada vektor postaje prevelik i svaki neuron mreže je opterećen s velikim brojem parametara. Za sliku dimenzija 200x200x3 bi to značilo da svi neuroni u prvom skrivenom sloju imaju 120,000 parametara koji se moraju podešavati, što dovodi do velikog računalnog opterećenja i gubitka vremena.



Slika 2.3. *Prikaz arhitekture konvolucijske neuronske mreže [4]*

Upravo je problem skaliranja na slike većih dimenzija riješen upotrebom konvolucijskih mreža. Na slici 2.3. je prikazana arhitektura konvolucijske mreže gdje se vidi da je svaki neuron mreže

povezan s malim dijelom prijašnjeg sloja mreže ili izvorne slike. Takva arhitektura dopušta određenim slojevima mreže sposobnost učenja različitih značajki slike, što kao rezultat daje modele s puno boljim i točnijim predikcijama.

2.2. Slojevi konvolucijske neuronske mreže

Upotrebljavaju se tri glavna sloja za izgradnju konvolucijske neuronske mreže: Konvolucijski sloj, sloj sažimanja te potpuno-povezani sloj. U nastavku poglavlja je detaljnije opisan svaki sloj te njegova zadaća unutar mreže.

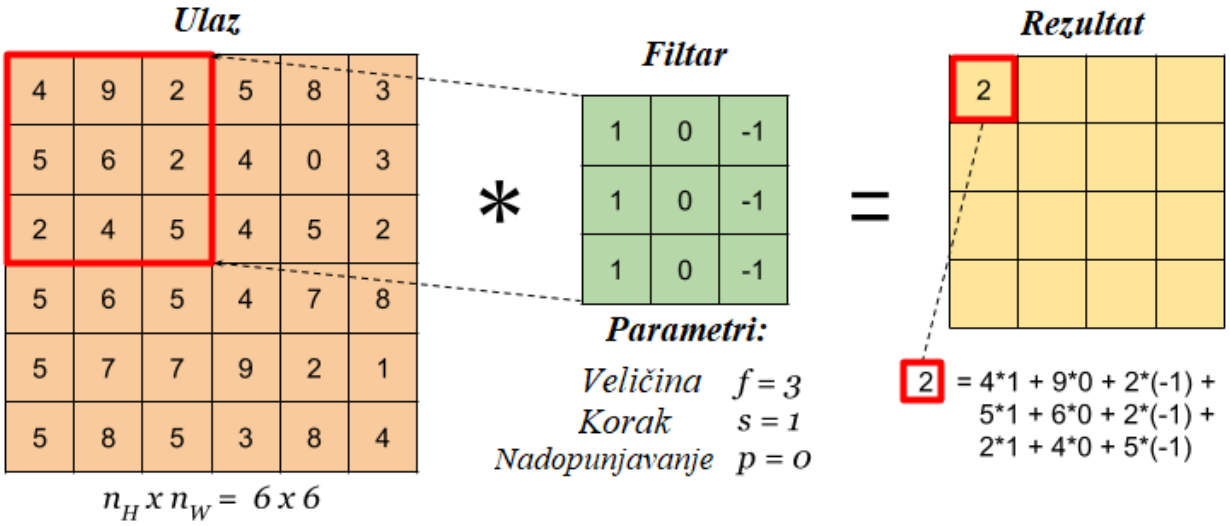
2.2.1. Konvolucijski sloj

Konvolucijski sloj je glavni sloj za izgradnju konvolucijske neuronske mreže. Zadaća konvolucijskog sloja je izvući određene značajke slike ovisno o trenutnoj dubini mreže. Taj proces se odvija upotrebljavajući filtar (engl. *kernel*) koji se pomiče uzduž slike i vrši konvoluciju nad određenim dijelom slike. Svaki filtar je relativno mali u odnosu na ulaznu sliku te mora imati isti broj kanala poput ulaznog podatka. Cilj konvolucije u nižim slojevima mreže je pamćenje jednostavnih značajki (rubovi, boja, orijentacija linija), dok se u kasnijim slojevima mreže pamte kompleksnije i specifičnije značajke vezane za pojedinu sliku. Način rada konvolucijskog sloja je definiran parametrima koji se upotrebljavaju prilikom odabira filtra. Ulazna veličina je dana dimenzijama $W_1 \; x \; H_1 \; x \; D_1$, gdje W_1 i H_1 predstavljaju širinu i visinu, a D_1 dubinu, tj. broj kanala slike. Parametri koji se upotrebljavaju prilikom konvolucije sadržavaju broj filtara koji se upotrebljava K, veličinu filtra F (dimenzija FxF), korak konvolucije S te nadopunjavanje P. Izlazna veličina ima dimenzije $W_2 \; x \; H_2 \; x \; D_2$ koju dobijemo sljedećim formulama:

$$W_2 = \frac{(W_1 - F + 2P)}{S} + 1 \tag{2-1}$$

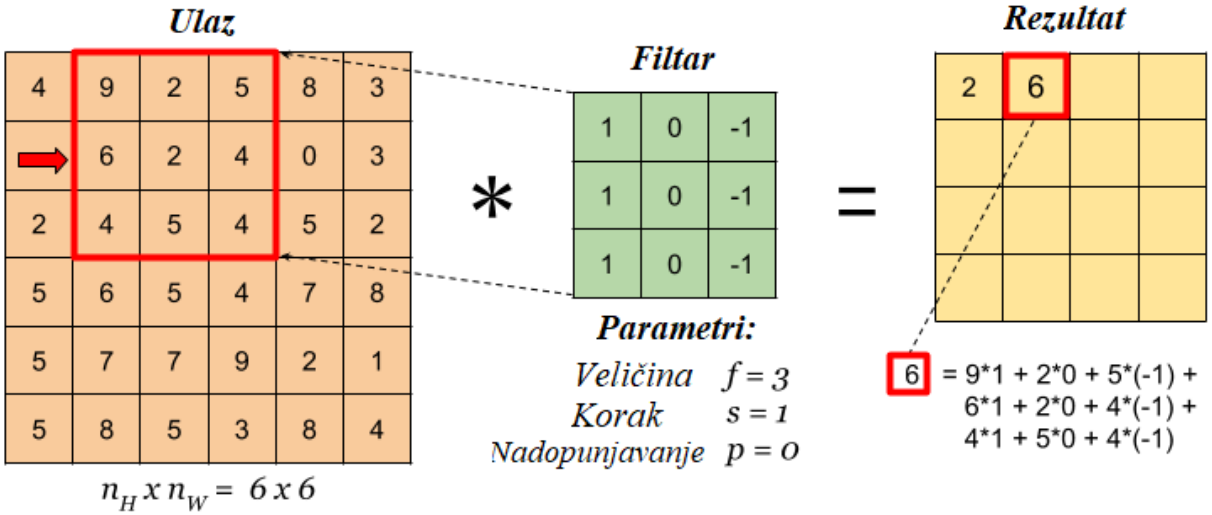
$$H_2 = \frac{(H_1 - F + 2P)}{S} + 1 \tag{2-2}$$

$$D_2 = K \tag{2-3}$$



Slika 2.4. *Prikaz prvog koraka konvolucije u konvolucijskom sloju [6]*

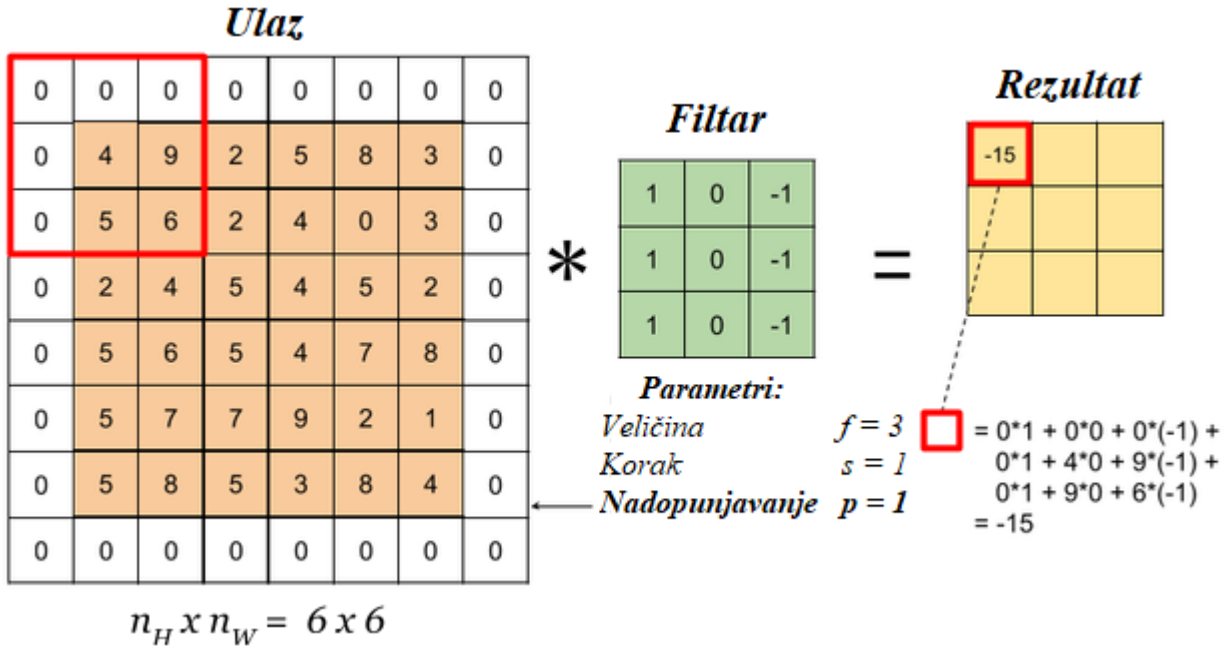
Na slici 2.4. je prikazan primjer konvolucijskog sloja gdje se kao ulaz upotrebljava slika dimenzija 6x6x1, te filtar dimenzija 3x3x1 koji obavlja konvoluciju nad tom slikom. Tijekom prvog koraka konvolucije, filtar se pomiče udesno obavljajući konvoluciju nad dijelom slike te se proces ponavlja dok filtar ne dođe do ruba slike kao što vidimo na slici 2.5. Kada filtar dosegne rub slike, vraća se na početak te se pomiče dolje kako bi ponovio proces. Rezultat konvolucije je jednodimenzionalna aktivacijska mapa, smanjenih dimenzija.



Slika 2.5. *Prikaz drugog koraka konvolucije u konvolucijskom sloju [6]*

Korak koji filtar uzima dok se pomiče udesno naziva se korak konvolucije te se označava sa S i može se regulirati radi smanjenja dimenzija izlaza i ubrzavanja procesa konvolucije. Na slici 2.5. korak konvolucije iznosi 1.

Uzevši u obzir da proces konvolucije smanjuje dimenzije slike, u dubokim mrežama s mnogo konvolucijskih slojeva to rezultira prevelikim smanjenjem slike i gubljenjem značajki. Problem se rješava nadopunjavanjem nula (engl. *padding*) rubova originalne slike. Osim rješavanja problema dimenzionalnosti, nadopunjavanjem nula se također zadržava veći broj informacija s rubova slika.



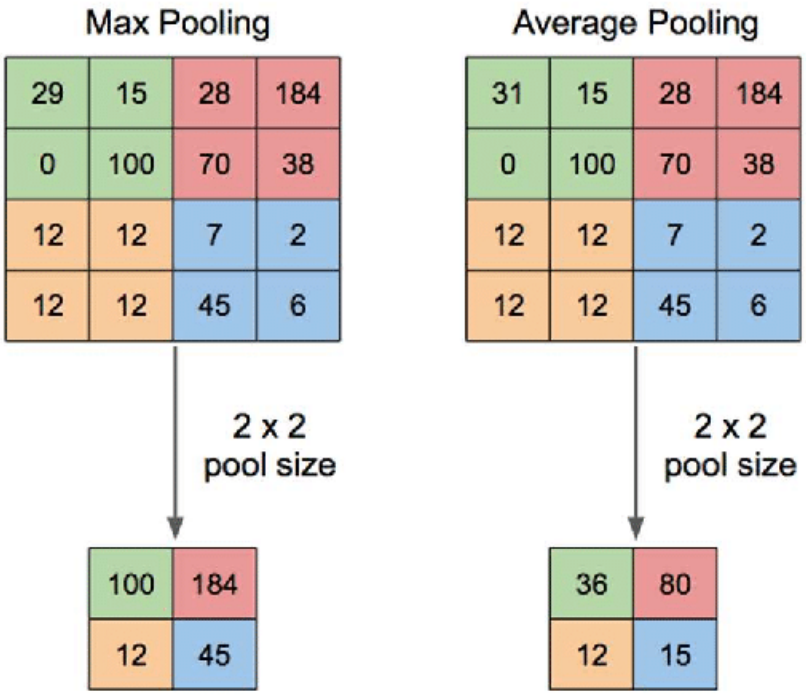
Slika 2.6. *Primjer nadopunjavanja nulama na ulaznom podatku [6]*

2.2.2. Sloj sažimanja

U prijašnjem poglavlju je spomenuto nadopunjavanje nula, čija je zadaća zadržati dimenzije slike nakon procesa konvolucije. Radi smanjenja dimenzija te ubrzanja samog procesa konvolucije, upotrebljava se sloj sažimanja. U ovom sloju se također upotrebljava filtar, najčešće dimenzija 2x2 te korak konvolucije 2 koji vrši operacije ovisno o vrsti sloja sažimanja. Postoje dvije vrste sloja sažimanja koji se upotrebljavaju u praksi:

- Sažimanje maksimalnom vrijednošću
- Sažimanje usrednjavanjem

Sažimanje maksimalnom vrijednošću vraća maksimalnu vrijednost nad dijelom ulaza kojeg jezgra pokriva, dok sažimanje usrednjavanjem vraća srednju vrijednost. Na slici 2.7. je prikazan način rada obje vrste sloja sažimanja.



Slika 2.7. *Prikaz sažimanja maksimalnom vrijednošću (a) odnosno usrednjavanjem (b) [7]*

Konvolucijski sloj i sloj sažimanja zajedno formiraju *i*-ti sloj konvolucijske mreže te se u praksi često na ta dva sloja mreže gleda kao na jedan sloj. Nakon izvlačenja značajki iz slike putem konvolucijskih slojeva, one se „izravnaju“ te u obliku vektora dovode na izlaz mreže upotrebljavajući potpuno-povezane slojeve.

2.2.3. Potpuno povezani sloj

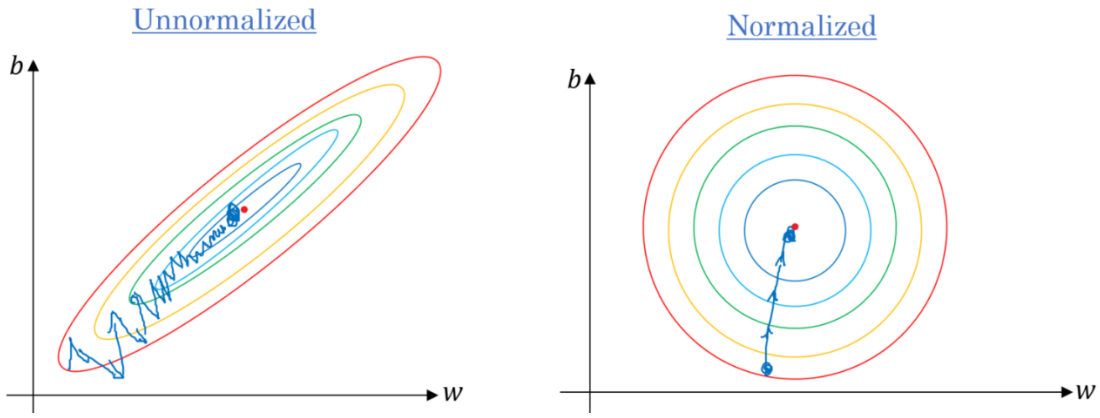
Dodavanje potpuno-povezanog sloja na kraju mreže je vrlo jeftin način učenja nelinearnih kombinacija kompleksnih značajki slike koja prolazi kroz mrežu. Na kraju prijašnjeg potpoglavlja je spomenuto da se izlaz zadnjeg konvolucijskog sloja „izravna“ u vektor dimenzija i kao takav proslijedi kroz unaprijednu mrežu gdje se upotrebljava unazadna-propagacija. Taj proces omogučuje mreži da razlikuje dominante, kompleksne značajke od onih jednostavnih, nižerazrednih.

3. OPTIMIZACIJA MREŽE

Kako bi se poboljšala svojstva i učinkovitost mreže, upotrebljavaju se različiti načini optimizacije poput primjene različitih optimizacijskih algoritama, načina inicijalizacije te regularizacije. Jedan od načina koji je izuzetno bitan prilikom optimizacije mreže je predobrada podataka. U nastavku poglavlja se spominju dvije metode za predobradbu podataka.

3.1. Predobradba ulaznih podataka

Kako bi se rad mreže ubrzao, prilikom dovođenja podataka na ulaz mreže treba napraviti predobradbu nad danim podatcima, drugim riječima, podatke treba standardizirati. Razlog tome što je vrijednosti značajki ulaznih podataka mogu biti različito skalirane što dovodi do neželjenog izgleda funkcije gubitka i samim time otežava ažuriranje parametara prilikom unazadne-propagacije.



Slika 3.1. *Prikaz kriterijske funkcije prije i poslije normalizacije [3]*

Proces standardizacije se odvija u dva koraka. Ako se uzme u obzir mreža sa dva ulazna podatka x_1 i x_2. Vrijednosti x_1 variraju u opsegu od 0 do 1, dok vrijednosti $x_2$variraju u opsegu od 0 do 0.01. S obzirom da je zadatak mreže pronaći kombinaciju tih dvaju ulaza kroz serije linearnih i nelinearnih kombinacija, parametri koji su dodijeljeni tim ulaznim podatcima će također biti različito skalirani. Prvi korak pri standardizaciji je oduzimanje srednje vrijednosti, što znači

da se od svake vrijednosti ulaznog podatka, u ovom slučaju x_1 i x_2, oduzima srednja vrijednost i ažurira s novim dobivenim vrijednostima.

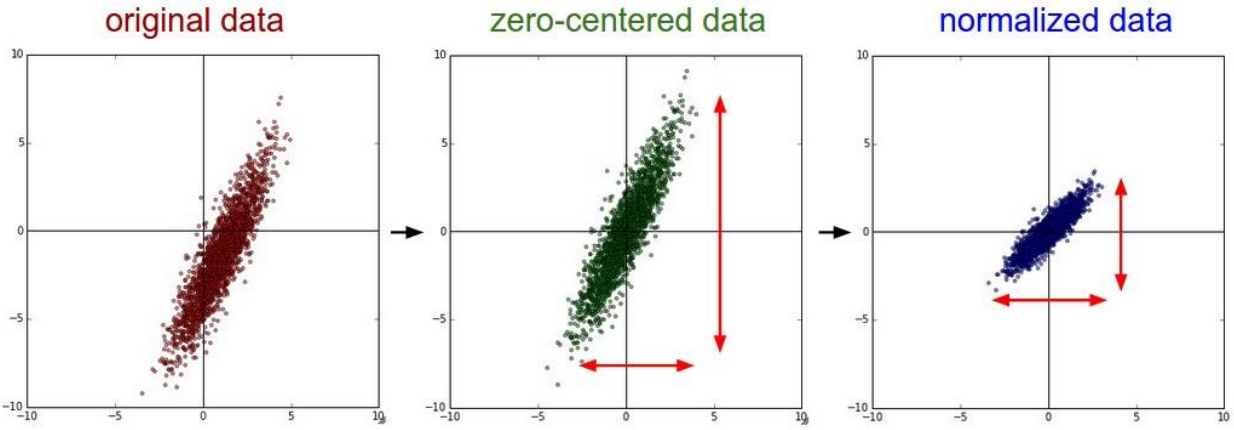
$$\mu = \frac{1}{m}\sum_{i=1}^{m} X^{(i)} \tag{3-1}$$

$$x := x - \mu \tag{3-2}$$

Drugi korak standardizacije je normalizacija podataka, odnosno skaliranje dimenzija na jednaku veličinu. Nakon što se podatci centriraju na nulu, svaka dimenzija podatka se podijeli njegovom standardnom devijacijom. Prilikom izvršenja oba koraka, na slici 3.2. je prikazan izgled podatkovnog skupa nakon predobradbe.

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} X^{(i)} ** 2 \tag{3-3}$$

$$x := \frac{x}{\sigma^2} \tag{3-4}$$

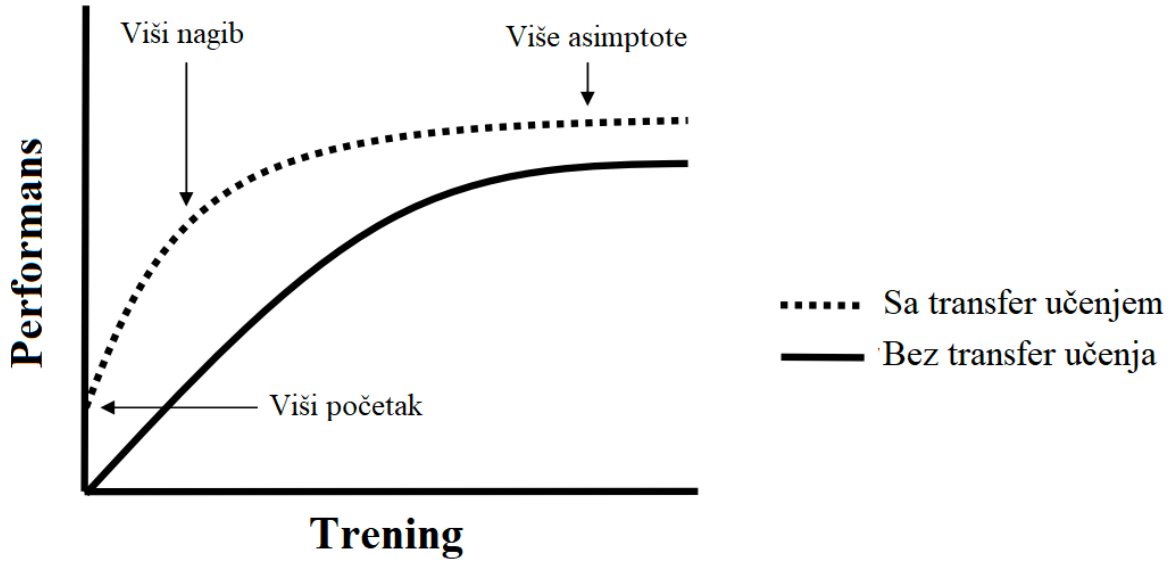


Slika 3.2. *Grafički prikaz predobradbe podataka [2]*

4. TRANSFER UČENJE

U doba kada se vrijeme smatra jednim od najvrijednijih aspekata ljudskog života, pronalazak rješenja za uštedu vremena prilikom rješavanja nekog problema je vrlo bitna stvar. Upravo to je jedna od glavnih motivacija transfer učenja. Transfer učenje se definira kao metoda gdje se prethodno naučeni modeli napravljeni s ciljem rješavanja nekog problema, upotrebljavaju kao početna točka pri rješavanju nekog drugog problema. Glavni razlog tome je što izgradnja modela od samog početka, kao i podešavanje parametara istog, zahtijeva puno vremena i nije optimalno rješenje. Transfer učenje je jedno od popularnijih pristupa u dubokom učenju te je najzastupljeniji u području računalnog vida i obradi prirodnih jezika.

Iako je transfer učenje idealno rješenje mnogih problema u polju dubokog učenja, ponekad njegova implementacija ne daje željene rezultate. Na slici 4.1. su prikazana moguća poboljšanja modela ukoliko je transfer učenje uspješno implementirano.



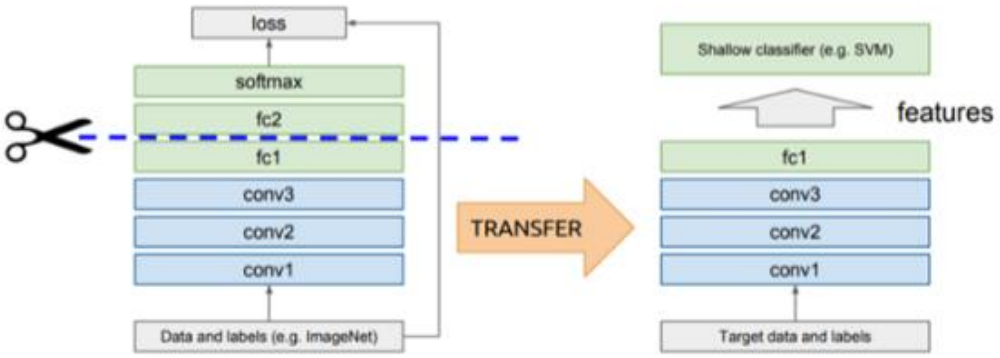
Slika 4.1. *Usporedba normalnog treniranja s transfer učenjem*

4.1. Metode transfer učenja za duboko učenje

U prijašnjem potpoglavlju je spomenuto da duboko učenje zahtijeva jako puno vremena i resursa te ponavljanje tog procesa nije optimalno rješenje. U većini slučajeva znanstvenici i istraživači dijele svoje modele koji imaju vrhunske rezultate na specifičnim zadatcima u svrhu transfer učenja. Da bi se transfer učenje implementiralo, upotrebljavaju se dvije metode koje su objašnjene u nastavku.

4.1.1. Prethodno naučeni modeli za izlučivanje značajke

U poglavlju 2., rečeno je da modeli imaju sposobnost učenja različitih značajki u različitim slojevima mreže. Takvi slojevi su spojeni u zadnji, potpuno-povezani sloj, da bi se dobila željena predikacija. Slojevita arhitektura tih mreža dopušta upotrebljavanje prethodno naučenih modela, poput VGG16 [9] ili InceptionV3 [10] modela, bez zadnjeg sloja mreže, radi izlučivanja značajki prilikom rješavanja drugih problem.



Slika 4.2. *Uklanjanje zadnjeg sloja i dodavanje jednostavnijeg klasifikatora [8]*

4.1.2. Fino podešeni prethodno naučeni modeli

Druga metoda transfer učenja osim zamjene klasifikatora u zadnjem sloju mreže, također omogućuje ponovno učenje određenih dijelova mreže radi podešavanja parametara. Moguće je podesiti parametre u svim slojevima mreže. U praksi se niži slojevi mreže uglavnom „zelede“ (zbog pretjeranog usklađivanja na podatke) te se podešavaju samo viši slojevi mreže. Razlog tome je što niži slojevi detektiraju rubove i neke općenite značajke slika podatkovnog skupa, dok oni viši slojevi detektiraju specifičnije, kompleksnije značajke vezane uz određene klase.

5. EKSPERIMENTALNI DIO

Programski kod je napisan u programskom jeziku Python na Anaconda distribuciji. Izvršavao se uz pomoć središnje procesorske jedinice (engl. *Central processing unit*), tzv. CPU, što je uvelike otežalo učenje modela. Prilikom implementacije modela upotrebljava se Keras, biblioteka otvorenog izvora (engl. *Open-source*) neuronskih mreža napisana u Pythonu. Osim Kerasa, upotrebljava se i Tensorflow koji služi za rješavanje kompleksnih numeričkih problema.

Eksperimentalni dio ovog rada se dijeli na dva zadatka:

- Aplicirati transfer učenje već gotovih mreža na skupu podataka
- Napraviti novi model konvolucijske mreže i testirati na istom skupu podataka

Prilikom implementacije transfer učenja upotrebljavaju se dva prethodno trenirana modela koje se poziva iz Keras API-ja. Da bi se uspješno implementirali, modelima se uklanja zadnji, klasifikacijski sloj mreže te se „zaleđuju“ preostali slojevi radi sprječavanja ponovnog treniranja parametara. Naposljetku, modelu se dodaje novi klasifikacijski sloj s dva kanala koji predstavljaju dvije klase koje se klasificiraju u ovome radu. Modeli će biti testirani na kaggle skupu podataka mačaka i pasa [11] koji sadrži 25,000 jedinstvenih slika. Podatkovni skup koji se upotrebljava prilikom transfer učenja je smanjen zbog računalnih ograničenja te sadrži 500 slika iz kaggle skupa mačaka i pasa podijeljenih u skupu za učenje, validaciju i test (400/50/50).

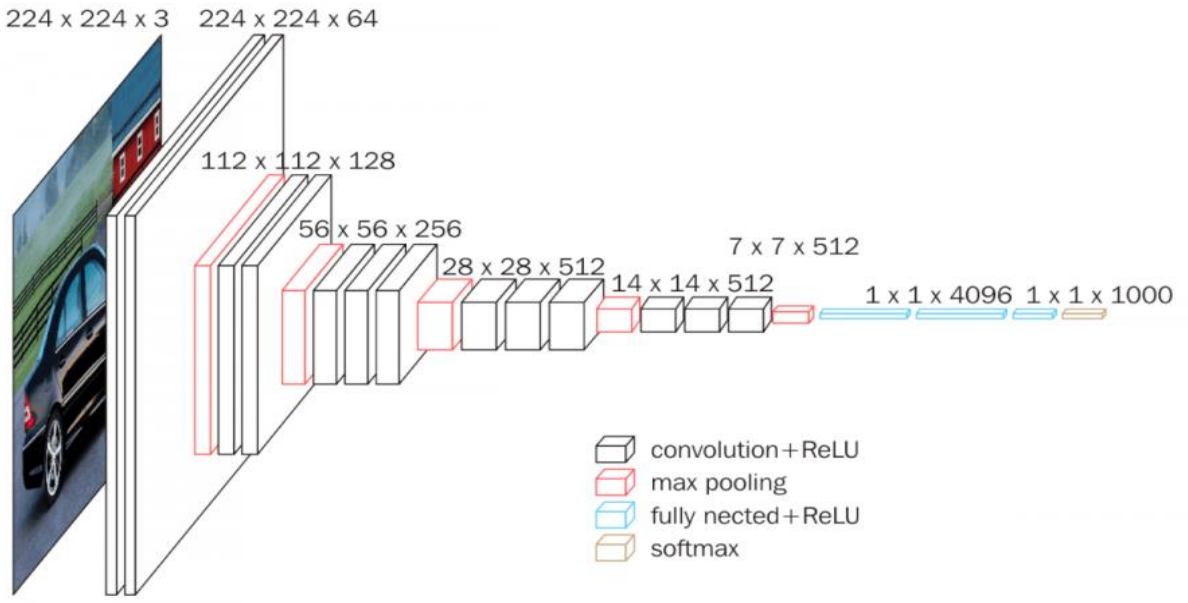
5.1. VGG16 konvolucijska neuronska mreža

VGG16 je model konvolucijske neuronske mreže namijenjen za klasifikaciju slika velikih dimenzija [9]. Model postiže preciznost klasifikacije od 92.7% na ImageNet-ovom skupu podataka koji sadrži preko 14 miliona slika koje pripadaju 1000 klasa.

5.1.1. Arhitektura i način rada mreže

Na ulaz VGG16 mreže se dovodi RBG slika fiksnih dimenzija 224x224x3. Slika prolazi kroz red konvolucijskih slojeva koji sadrže filtre vrlo malih dimenzija (3x3). Korak konvolucije je fiksiran kroz cijelu mrežu te iznosi 1. Nadopunjavanje nula svakog konvolucijskog sloja je takav da dimenzije ulaza ostanu iste nakon procesa konvolucije. Na slici 5.1. je također vidljivo da mreža

sadrži 5 Max-pool slojeva koji se ne nalaze iza svakog konvolucijskog sloja. Red konvolucijkih slojeva završava s 3 potpuno-spojena sloja od kojih prva dva predstavljaju vektore dimenzija 4096x1. Treći sloj je softmax sloj te služi za klasifikaciju slika i sadrži 1000 kanala, po jedan za svaku klasu. Treba napomenuti da mreža u svim skrivenim slojevima upotrebljava ReLU aktivacijsku funkciju.



Slika 5.1. *Prikaz arhitekture VGG16 modela [9]*

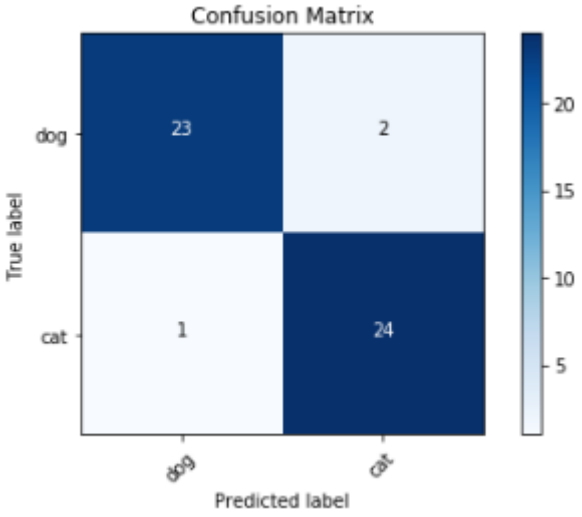
5.1.2. Evaluacija VGG16 modela

Prilikom treniranja modela, računa se funkcija gubitka (engl. *Loss*) te preciznost (engl. *Accuracy*) upotrebljavanjem kategoričke unakrsne entropije (engl. *Categorical crossentropy*). Također, tijekom treniranja se upotrebljava optimizacijski algoritam Adam, koji je u suštini unaprijeđena verzija stohastičkog gradijentnog spusta. Model se trenira u trajanju 5 epoha gdje se prilikom svake iteracije propagira cijeli podatkovni skup kroz mrežu u obliku malog batcha (engl. *Mini batch*). Veličina batcha koja se propagira kroz mrežu određena je parametrom batch_size te ovisi o veličini podatkovnog skupa te za učenje, validacijski i testni skup u našem slučaju iznosi 50/10/50. U nastavku je prikazana simulacija modela nad skupom podataka za učenje i validaciju, te evaluacija testnog skupa upotrebljavajući matricu zbunjenosti.

Tablica 5.1. *Simulacija VGG16 modela*

Loss (Trening)	Accuracy (Trening)	Val_Loss (Validacija)	Val_Accuracy (Validacija)
0.0048	100%	0.1046	94%

U tablici 5.1. je prikazan rezultat učenja prethodno naučenog modela na skupu podataka od 500 slika. Iz tablice se vidi da je preciznost skupa za učenje savršena u odnosu na skup za validaciju. Također, iz razlike preciznosti skupa podataka za učenje i validaciju, može se zaključiti da dolazi do pretjeranog usklađivanja podataka nad skupom za učenje. U nastavku ovog poglavlja model će biti evaluiran na testnom skupu podataka od 50 slika podijeljenih u dvije kategorije.



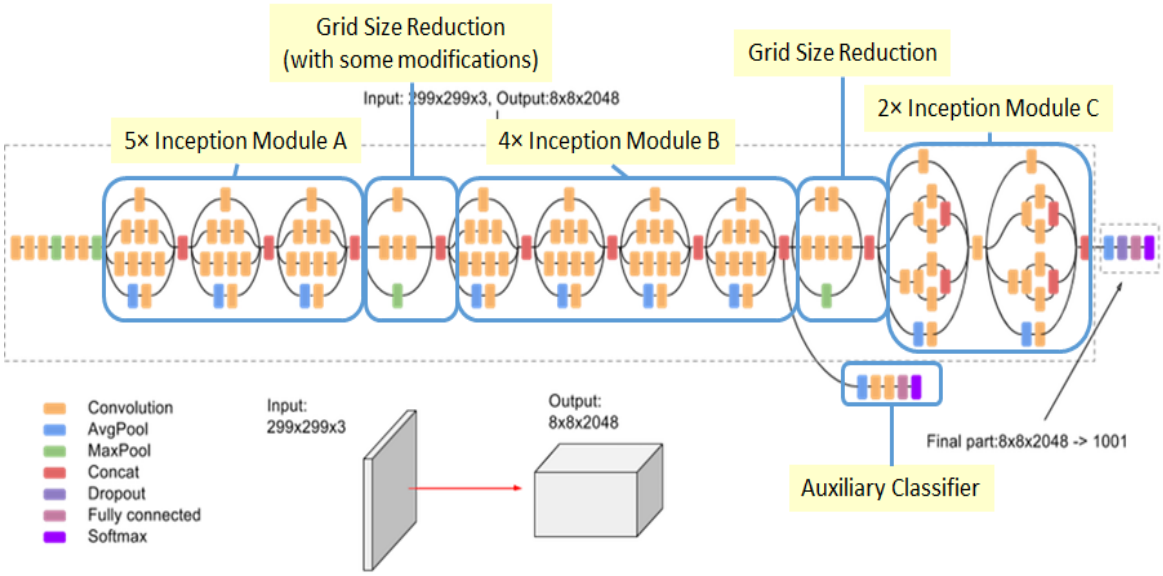
Slika 5.2. *Matrica zbunjenosti za VGG16 model*

Na slici 5.2. je prikazana matrica zbunjenosti koja služi za evaluaciju modela nad testnim skupom podataka. Iz slike se može zaključiti da je točnost klasifikacije 94%, što je isto poput validacijskog skupa prilikom treniranja modela. Također, primijeti se da je model krivo klasificirao mačku 1 put te psa 2 puta.

5.2. InceptionV3 konvolucijska neuronska mreža

Ideja mreže unutar mreže je nastala s ciljem smanjenja ukupnog broja parametara i povećanja računalne učinkovitosti mreže [10]. Prije Inception modela, konvolucijske mreže su postajale sve dublje i dublje, sa većim brojem parametara, nadajući se sve boljem rezultatu predikcije. U odnosu na VGG16 koji sadrži 140 milijuna parametara, InceptionV3 iako ima 42 sloja sadrži 7 puta manje parametara te je puno brža i učinkovitija. U nastavku poglavlja se detaljnije ulazi u arhitekturu i način rada mreže , te se naposljetku prethodno naučeni model evaluira na testnom skupu podataka.

5.2.1. Arhitektura i način rada mreže



Slika 5.3. *Prikaz arhitekture InceptionV3 modela [10]*

Glavni dio arhitekture se sastoji od tri modula: A, B i C. Svaki od navedenih modula implementira faktorizirajuće konvolucije na različite načine. Modul A smanjuje dimenzije filtara koje upotrebljava radi smanjenja ukupnog broja parametara mreže. Dva filtra 3x3 dimenzija daju manji broj parametara od jednog 5x5 filtra dok mreža i dalje zadržava učinkovitost. Modul B upotrebljava asimetrične konvolucije. Drugim riječima, ovaj modul zamjenjuje jedan 3x3 filtar s 3x1 filtrom i 1x3 filtrom, što dodatno smanjuje ukupni broj parametara. Kao i prijašnja dva modeua, ideja iza modula C je smanjenje broja parametara mreže kroz veliki broj filtara manjih dimenzija.

Osim modula koji su spomenuti, InceptionV3 upotrebljava pomoćni klasifikator. Također, InceptionV3 pronalazi računalno učinkovit način smanjenja dimenzija aktivacijskih mapa u odnosu na druge konvolucijske mreže.

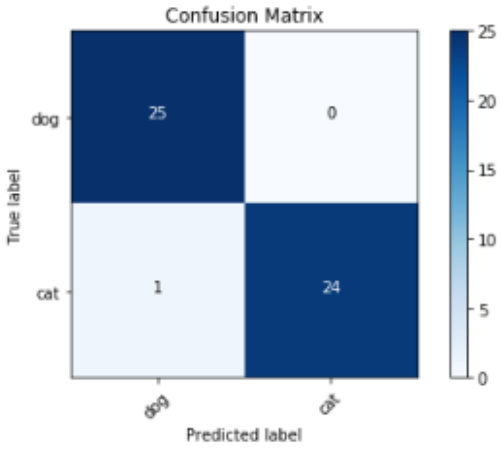
5.2.2. Evaluacija InceptionV3 modela

Poput VGG16 modela, model se evaluira računajući funkciju gubitka i preciznost nad skupom za učenje i validacijskom skupu upotrebljavajući kategoričku unakrsnu entropiju. Prilikom treniranja modela također je korišten Adam optimizacijski algoritam. Broj epoha tijekom treniranja, kao i veličina batcha podataka koji su se upotrebljavali je također ostao isti. U nastavku je prikazan rezultat treniranja u obliku tablice te evaluacija nad testnim skupom podataka upotrebljavanjem matrice zbunjenosti.

Tablica 5.2. *Simulacija InceptionV3 modela*

Loss (Trening)	Accuracy (Trening)	Val_Loss (Validacija)	Val_Accuracy (Validacija)
0.2319	93.25%	0.1743	94%

Iz tablice 5.2. se vidi da je preciznost skupa za učenje lošija u odnosu na validacijski skup. Također, ukoliko se usporedi s VGG16 modelom, vidi se razlika u preciznosti skupa podataka za učenje između dva modela. Način poboljšanja preciznosti trening skupa se može riješiti povećanjem broja epoha tijekom treniranja modela. U nastavku ovog poglavlja model će biti evaluiran na testnom skupu podataka od 50 slika podijeljenih u dvije kategorije. Treba napomenuti da slike koje se koriste prilikom evaluacije modela su iste slike korištene prilikom evaluacije VGG16 modela.



Slika 5.4. *Matrica zbunjenosti za InceptionV3 model*

Iz slike 5.4. se može zaključiti da je točnost klasifikacije 98%, što je bolje u odnosu na validacijski skup podataka. Također, primjećuje se da je model krivo klasificirao psa samo jednom. Krivo klasificirana slika nije ista kod oba modela te ne sadrži neke distorcije, dakle ne postoji jasan razlog krive klasifikacije.

5.3. Testiranje vlastitog modela

Cilj ovog poglavlja je kreirati u potpunosti novi model s jedinstvenom arhitekturom za klasifikaciju dviju klasa. Upotrebljen je isti skup podataka međutim zbog resursa i memorijskog ograničenja, slike su modificirane. Dimenzija slika je smanjenja na 70x70 te su slike isključivo sivih nijansi. Ideja je napraviti modele različitih arhitektura, evaluirati ih na validacijskom skupu te model sa najboljom preciznošću predikcije maksimalno podesiti. Naposljetku, podešeni model će biti evaluiran na testnom skupu od 2,500 slika podijeljenih u dvije klase. Izmjene u samoj arhitekturi se svode na različiti broj konvolucijskih i potpuno-povezanih slojeva, te na broj neurona u svakom od tih slojeva.

5.3.1. Arhitektura i evaluacija modela

Prilikom učenja upotrebljen je cijeli kaggle skup podataka mačaka i pasa od 25,000 slika podijeljenih u skup za učenje, validaciju i test (20,000/2,500/2,500). Kao i kod prethodna dva modela, tijekom učenja se računa funkcija gubitka i preciznost upotrebljavajući kategoričku unakrsnu entropiju. U tablici 5.3. je prikazana arhitektura svakog modela te njegova evaluacija na validacijskom skupu podataka.

Tablica 5.3. *Prikaz arhitektura i simulacija napravljenih modela*

Arhitektura	Loss (Trening)	Accuracy (Trening)	Val_Loss (Validacija)	Val_Accuracy (Validacija)
1-Conv64—0-Dense	0.4203	80.77%	0.5309	74.33%
2-Conv64—0-Dense	0.4034	81.46%	0.4396	80.05%
3-Conv64—0-Dense	0.4022	81.64%	0.4045	81.72%
1-Conv128—0-Dense	0.4080	81.50%	0.5510	73.55%
2-Conv128—0-Dense	0.4409	79.75%	0.4987	76.31%
3-Conv128—0-Dense	0.3522	84.41%	0.3827	82.99%
1-Conv64—1-Dense64	0.3355	85.51%	0.5441	74.37%
2-Conv64—1-Dense64	0.3352	85.48%	0.4560	79.98%
3-Conv64—1-Dense64	0.3664	83.71%	0.3957	82.21%
1-Conv128—1-Dense128	0.2221	90.89%	0.6458	73.73%
2-Conv128—1-Dense128	0.3072	86.52%	0.4848	78.16%
3-Conv128—1-Dense128	0.3874	82.47%	0.3994	81.52%

U tablici 5.3. su prikazane različite arhitekture mreža koje su trenirane u ovom poglavlju. Ukoliko se uzme u obzir „1-Conv64—0-Dense“ mreža iz tablice, iz imena se može primjetiti da mreža sadrži 1 konvolucijski sloj od 64 neurona, te niti jedan potpuno-povezani sloj. Isto tako, ukoliko se uzme „1-Conv128—1-Dense128“ mreža iz tablice, iz imena se može zaključiti da mreža sadrži 1 konvolucijski sloj od 128 neurona te 1 potpuno-povezani sloj također od 128 neurona. Cilj evaluacije modela je istovremeno pronaći model s najvećom preciznošću predikcije validacijskog skupa te minimalnom razlikom u preciznosti predikcije između trening i validacijskog skupa. Iz Tablice 5.3. se zamjećuje da model s 3 konvolucijska sloja od 128 neurona te bez potpuno-povezanog sloja ima najbolju preciznost predikcije. Isto tako, vidi se da modeli s potpuno povezanim slojem i manjim brojem konvolucijskih slojeva imaju puno veću preciznost nad skupom za učenje u odnosu na validacijski skup, drugim riječima, u nekom trenutku dolazi do pretjeranog usklađivanja na skupu podataka za učenje. Također treba napomenuti da su svi modeli trenirani u trajanju od 5 epoha, uz veličinu batcha od 32 slike. U nastavku se mreža sa najboljom predikcijom na validacijskom skupu podataka, dakle „3-Conv128—0-Dense“ model, testira upotrebljavajući različite optimizacijske algoritme.

Tablica 5.4. *Rezultati različitih optimizacijskih algoritama za „*3-Conv128—0-Dense“ model

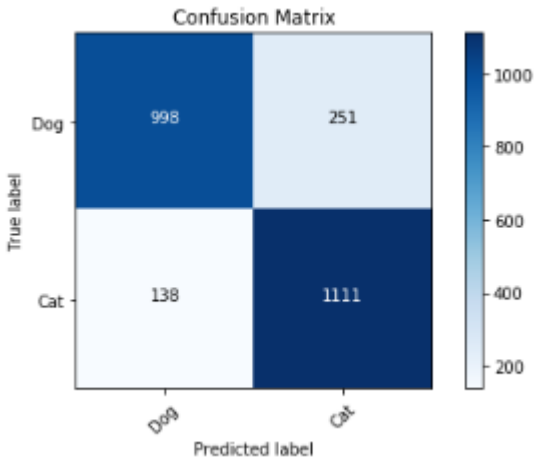
Optimizacijski algoritam	Loss (Trening)	Accuracy (Trening)	Val_Loss (Validacija)	Val_Accuracy (Validacija)
SGD	0.6381	63.40%	0.6373	61.95%
Momentum	0.4120	81.50%	0.4244	80.02%
RMSprop	0.3148	86.22%	0.6146	76.09%
Adam	0.3522	84.48%	0.3827	82.99%

Iz tablice 5.4. se vidi kako najbolju preciznost na skupu za validaciju daje optimizacijski algoritam Adam. Također, prilikom upotrebljavanja stohastičkog gradijentnog spusta, preciznost skupa za učenje i validaciju se sporo povećavala. U odnosu na druge algoritme, može se zaključiti da je algoritam relativno spor te da je potreban veći broj epoha učenja kako bi se postigla bolja preciznost. Isto tako, iz tablice se vidi da algoritam RMSprop u jednom trenutku počinje pretjerano usklađivati podatke za učenje što dovodi do smanjenja preciznosti na validacijskom skupu podataka. Treba napomenuti da je učenje modela za svaki algoritam trajalo 5 epoha. U talici 5.5. je prikazano učenje modela u trajanju 10 epoha upotrebljavajući optimizacijski algoritam Adam.

Tablica 5.5. *Rezultati za optimizaciski algoritam Adam u trajanju 10 epoha*

Broj Epoha	Loss (Trening)	Accuracy (Trening)	Val_Loss (Validacija)	Val_Accuracy (Validacija)
1	0.5856	68.25%	0.4856	76.22%
2	0.4636	78.23%	0.4498	79.50%
3	0.3908	82.41%	0.4017	81.97%
4	0.3339	85.42%	0.3858	82.66%
5	0.2840	87.97%	0.3586	83.51%
6	0.2400	89.95%	0.3566	85.41%
7	0.1902	92.42%	0.3701	85.49%
8	0.1550	93.83%	0.4173	84.40%
9	0.1180	95.46%	0.4679	84.72%
10	0.0850	96.79%	0.4986	85.21%

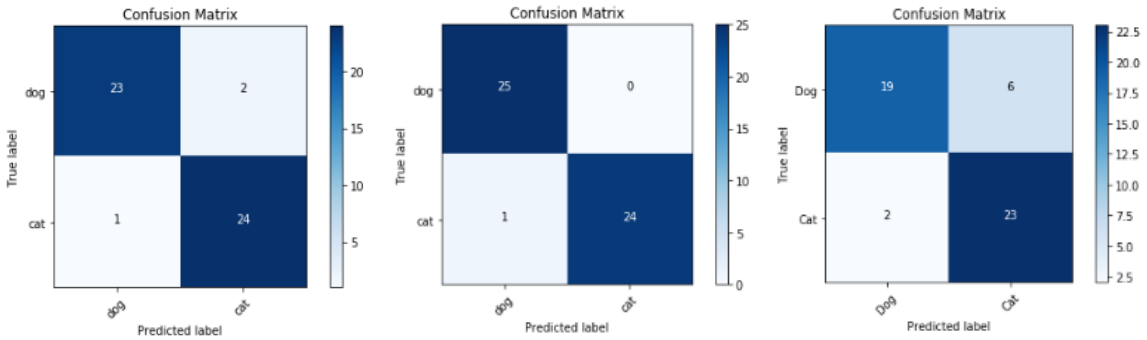
Iz tablice 5.5. je vidljivo da se povećanjem broja epoha učenja također povećala preciznost validacijskog skupa podataka. Također, ukoliko se uzme u razmatranje 7. epoha pa nadalje, vidljivo je da gubitak validacijskog skupa počinje rasti a preciznost stagnirati. Drugim riječima, model nakon 6. epohe treniranja počinje pretjerano usklađivati podatke na skupu za učenje. Upotrebljavajući „ModelCheckpoint“, jednu od Kerasovih funkcija uzvratnog poziva (engl. *Callback*), spremljena je najbolja verzija modela. Iz tablice 5.5. se može zaključiti da je najučinkovitija verzija modela nakon 6. epohe treniranja, te se upravo ta verzija koristi prilikom evaluacije na testnom skupu podataka. Na slici 5.5. je prikazana matrica zbunjenosti za testni skup od 2,500 podataka. Treba spomenuti da su neki podatci iz testnog skupa oštećeni te su uklonjeni prilikom evaluacije modela.



Slika 5.5. *Matrica zbunjenosti vlastitog modela*

Iz slike 5.5. se može zaključiti da je točnost klasifikacije 84.36% što je lošije od predikcije na validacijskom skupu podataka. Također, vidljivo je da model skoro dvostruke lošije procjenjuje pse u odnosu na mačke.

Kako bi se usporedili rezultati vlastitog modela sa prethodno naučenim modelima, u nastavku će se prilikom predikcije upotrijebiti isti testni skup podataka kao i kod prethodno naučenih modela. Na slici 5.6. su prikazane matrice zbunjenosti za sva tri modela testiranih na istom skupu podataka.



Slika 5.6. *Matrice zbunjenosti za VGG16 (a), InceptionV3 (b) i vlastiti model (c)*

Na slici 5.6. je prikazana usporedba predikcije sva tri modela na istom skupu podataka. Iz slike se vidi da prethodno naučeni modeli imaju puno bolju preciznost predikcije, što je očekivano. Također, treba napomenuti da dvije krivo klasificirane slike psa VGG16 modelom su također krivo klasificirane upotrebom vlastitog modela. Na slikama ne postoje distorcije te nema vidljivih razloga krive klasifikacije.

6. ZAKLJUČAK

Tema ovog diplomskog rada je bila upoznavanje s dubokim neuronskim mrežama, s naglaskom na konvolucijske neuronske mreže. Također, u teorijskom dijelu je obrađen pojam transfer učenja, metode koje se koriste prilikom transfer učenja te način implementacije. U eksperimentalnom dijelu trebalo je implementirati prethodno naučene modele upotrebljavajući jednu od metoda transfer učenja te napraviti vlastiti model i testirati ih na skupu podataka.

Za implementaciju prethodno naučenih mreža upotrebljavani su VGG16 i InceptionV3 modeli. Modeli su trenirani na istim skupovima podataka od 500 slika podijeljenih u skup za učenje, validaciju i test. Iz postupka učenja VGG16 modela vidi se da dolazi do pretjeranog usklađivanja podataka na skupu za učenje, što dovodi do lošije preciznosti na validacijskom skupu. Također, to se odrazilo prilikom testiranja te se postiže preciznost od 94%. Problem pretjeranog usklađivanja se mogao rješiti povećanjem skupa za učenje, dodavanjem distorcija podatcima skupa za učenje kako bi se indirektno povećao skup ili dodavanjem regularizacije uz klasifikacijski sloj koji je dodan na vrhu mreže. Iz postupka učenja InceptionV3 modela zamjećuje se manja preciznost na skupu za učenje što se može rješiti dodatnim epohama učenja modela. Iako je lošija preciznost predikcije na skupu za učenje, prilikom evaluacije modela na testnom skupu podataka model je postigao preciznost od 98%.

Prilikom implementacije vlastitog modela korišten je isti skup podataka međutim zbog računalnog ograničenja, slike su smanjenih dimenzija i sivih nijansi. Arhitektura modela sa najvećom preciznosti predikcije se sastoji od 3 konvolucijska sloja od 128 neurona te bez potpuno poveznaog sloja. Također, prikazano je da modeli sa potpuno povezanim slojevima te sa manjim brojem konvolucijskih slojeva relativno brzo počinju pretjerano usklađivati podatke za učenje što rezultira u velikoj pogrešci na validacijskom i testnom skupu. Taj problem bi se mogao riješiti proširenjem skupa za učenje iz razloga što bi model prestao pamtiti specifične značajke slika, ili dodavanjem „Dropout“ regularizacije još jednom sloju mreže. Preciznost na validacijskom skupu postignuta prilikom treniranja vlastitog modela je manja u odnosu na prethodno naučene modele, kao i prezicnost na testnom skupu koja iznosi 84.36%. Povećanje preciznosti bih se moglo postići povećanjem skupa podataka, kompleksnijom arhitekturom mreže ili uvođenjem kvalitetnijih ulaznih podataka. Zahtjevi za računalnim resursima su veliki te je upravo to glavno ograničenje prilikom treniranja dubokih mreža.

LITERATURA

[1] Aktivacijske funkcije i Klasične feed-forward mreže

http://cs231n.github.io/neural-networks-1/

[2] Preprocesiranje podataka

http://cs231n.github.io/neural-networks-2/

[3] Gradient descent i kriterijska funkcija

https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

[4] CNN

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[5] CNN

http://cs231n.github.io/convolutional-networks/

[6] Konvolucijski sloj

https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

[7] Pooling sloj

https://knowridge-i.com/knowledge/pooling-the-ordinary-disaster-in-convolutional-network/

[8] Transfer učenje

https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[9] VGG16

https://neurohive.io/en/popular-networks/vgg16/

[10] InceptionV3

https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

[11] Kaggle skup podataka

https://www.kaggle.com/c/dogs-vs-cats/data

SAŽETAK

U ovom radu opisane su arhitekture i način rada konvolucijskih neuronskih mreža te je detaljno objašnjena zadaća svakog sloja unutar konvolucijske neuronske mreže. Također, opisana je metoda transfer učenja i načini upotrebljavanja u praksi. Detaljno je prikazana arhitektura i način rada modela upotrebljavanih za transfer učenje, te njihova implementacija i testiranje na skupu podataka dviju klasa. Obavljeno je učenje vlastitiog modela te na temelju rezultata, najbolji model se upotrebljavao za daljnje podešavanje da se dobije što veća preciznost predikcije. Model je također testiran na skupu podataka te su rezultati uspoređeni.

Ključne riječi: Neuronske mreže, Konvolucijske neuronske mreže, Transfer učenje, model, VGG16, InceptionV3

ABSTRACT

In this master thesis, an architecture and the way convolutional neural networks work are described in detail. Also, each layer of Convolutional neuaal network and it's task within network is shown. Aside from that, method of Transfer learning is introduced, and it's usage in practice. Description of one of the covered methods and it's implementation is shown on datasets used throughout this thesis, containing two classes. Also, training of our own models is shown with variety of architectures using the same dataset. Model with best performanse was later used to fine-tune and to try different optimizing algorithms. After fine-tuning, results were compared to pre-trained models

Key words: Neural netowrks, Convolutional neural networks, Transfer learning, model, VGG16, InceptionV3

Životopis

Josip Žalac rođen je 17.12.1992. u Osijeku. 2007. godine završava osnovnu školu „Retfala“ te iste godine upisuje srednju školu „Elektronička i prometna škola Osijek“, u Osijeku, smjer elektrotehničar. Završava ju 2011. godine. Tijekom cijelog srednjoškolskog obrazovanja ostvaruje dobar uspijeh. Po završetku srednje škole, iste godine upisuje stručni studij elektrotehnike, smjer informatika na „Elektrotehnički fakultet Osijek“ u Osijeku. Po završetku stručnog studija, 2015. godine upisuje razlikovnu godinu za prekvalifikaciju na smjer Računarstva.

Razlikovnu godinu završava 2016. godine te iste godine upisuje diplomski studij računastva Elektrotehničkog fakulteta Osijek, smjer računarstvo, izborni blok procesno računarstvo.

Žalac Josip

PRILOG

U prilogu ovog rada se nalazi kod koji je upotrebljavan prilikom učenja mreža.

- Funkcije

```
def my_gen(gen):
    while True:
        try:
            data, labels = next(gen)
            yield data, labels
        except:
            pass
```

```
def plots(ims, figsize=(12,6), rows=2, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if(ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows +1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize=12)
        plt.imshow(ims[i], interpolation=None if interp else 'none')
```

```
def plot_confusion_matrix(cm, classes, normalize = False, title ='Confusion Matrix', cmap=plt.cm.Blues):


    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)


    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                horizontalalignment='center',
                color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

- VGG16

```
import tensorflow as tf
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense, GlobalAveragePooling2D, Dropout, Flatten
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import *
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt

%matplotlib inline
```

```
train_path = 'cats-and-dogs/train'
valid_path = 'cats-and-dogs/valid'
test_path = 'cats-and-dogs/test'
```

```
train_datagen = ImageDataGenerator(rescale=1. / 255)

test_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
train_batches = ImageDataGenerator().flow_from_directory(train_path,
                                                         target_size=(224,224),
                                                         classes=['dog','cat'],
                                                         color_mode='rgb',
                                                         shuffle=True,
                                                         batch_size=50)

valid_batches = ImageDataGenerator().flow_from_directory(valid_path,
                                                         target_size=(224,224),
                                                         classes=['dog','cat'],
                                                         color_mode='rgb',
                                                         shuffle=True,
                                                         batch_size=10)

test_batches = ImageDataGenerator().flow_from_directory(test_path,
                                                        target_size=(224,224),
                                                        classes=['dog','cat'],
                                                        color_mode='rgb',
                                                        shuffle=False,
                                                        batch_size=50)
```

```
vgg16_model = keras.applications.vgg16.VGG16()
```

```
# Functional API > Sequential model

model = Sequential()

for layer in vgg16_model.layers[:-1]:  # Uklanjanje zadnjeg sloja
    model.add(layer)
```

```
# "Zaleđivanje" svih slojeva

for layer in model.layers:
    layer.trainable = False
```

```
# Dodavanje klasifikacijskog sloja

model.add(Dense(2, activation="softmax"))
```

```
# Kompajliranje i treniranje modela

model.compile(loss="categorical_crossentropy", optimizer = 'adam', metrics=['accuracy'])

model.fit_generator(train_batches, validation_data=valid_batches, validation_steps=5, epochs=5, steps_per_epoch=8, verbose=1)
```

```
# Spremanje labela testnog skupa za matricu zbunjenosti

test_labels = test_labels[:,0]
test_labels
```

```
# Pravljenje predikcije nad batchom podataka (Cijeli testni skup)

predictions = model.predict_generator(test_batches, steps=1, verbose=1)
```

```
# Spremanje labela i predikcija te plotanje matrice zbunjenosti

cm = confusion_matrix(test_labels, np.round(predictions[:,0]))

cm_plot_labels = ['dog','cat']
plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')
```

- InceptionV3

```
import tensorflow as tf
import numpy as np
import keras
from keras.applications.inception_v3 import InceptionV3
from keras.models import Model
from keras.models import Sequential
from keras.layers import Activation
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.normalization import BatchNormalization
from keras.layers import Dense, GlobalAveragePooling2D, Dropout, Flatten
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt

%matplotlib inline
```

```
train_path = 'cats-and-dogs/train'
valid_path = 'cats-and-dogs/valid'
test_path = 'cats-and-dogs/test'
```

```
train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2, zoom_range=0.2,
                                   horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
train_batches = train_datagen.flow_from_directory(train_path,
                                                  target_size=(299, 299),
                                                  classes=['dog','cat'],
                                                  color_mode='rgb',
                                                  shuffle=True,
                                                  batch_size=50)

valid_batches = test_datagen.flow_from_directory(valid_path,
                                                 target_size=(299, 299),
                                                 classes=['dog','cat'],
                                                 color_mode='rgb',
                                                 shuffle=True,
                                                 batch_size=10)

test_batches = test_datagen.flow_from_directory(test_path,
                                                target_size=(299, 299),
                                                classes=['dog','cat'],
                                                color_mode='rgb',
                                                shuffle=False,
                                                batch_size=50)
```

```
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299, 299, 3))
```

```
# Dodavanje AvgPool i Dropout sloja na "vrh" InceptionV3 modela

x = base_model.output
x = GlobalAveragePooling2D(name='avg_pool')(x)
x = Dropout(0.3)(x)
classifier = Dense(2, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=classifier)
```

```
# "Zaleđivanje" svih slojeva

for layer in base_model.layers:
    layer.trainable = False
```

```
# Dodavanje klasifikacijskog sloja

model.add(Dense(2, activation="softmax"))
```

```
# Kompajliranje i treniranje modela

model.compile(loss="categorical_crossentropy", optimizer = 'adam', metrics=['accuracy'])

model.fit_generator(train_batches, validation_data=(my_gen(valid_batches)), validation_steps=5, epochs=5, steps_per_epoch=8, verl
```

```
# Spremanje labela testnog skupa za matricu zbunjenosti

test_labels = test_labels[:,0]
test_labels
```

```
# Pravljenje predikcije nad batchom podataka (Cijeli testni skup)

predictions = model.predict_generator(test_batches, steps=1, verbose=1)
```

```
# Spremanje labela i predikcija te plotanje matrice zbunjenosti

cm = confusion_matrix(test_labels, np.round(predictions[:,0]))

cm_plot_labels = ['dog','cat']
plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')
```

- Vlastiti model

```
import tensorflow as tf
import numpy as np
import keras
from keras.applications.inception_v3 import InceptionV3
from keras.models import Model
from keras.models import Sequential
from keras.layers import Activation
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.normalization import BatchNormalization
from keras.layers import Dense, GlobalAveragePooling2D, Dropout, Flatten
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt

%matplotlib inline
```

```
train_path = 'cats-and-dogs/train'
valid_path = 'cats-and-dogs/valid'
test_path = 'cats-and-dogs/test'
```

```
train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2, zoom_range=0.2,
                                   horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
train_batches = train_datagen.flow_from_directory(train_path,
                                                  target_size=(299, 299),
                                                  classes=['dog','cat'],
                                                  color_mode='rgb',
                                                  shuffle=True,
                                                  batch_size=50)

valid_batches = test_datagen.flow_from_directory(valid_path,
                                                 target_size=(299, 299),
                                                 classes=['dog','cat'],
                                                 color_mode='rgb',
                                                 shuffle=True,
                                                 batch_size=10)

test_batches = test_datagen.flow_from_directory(test_path,
                                                target_size=(299, 299),
                                                classes=['dog','cat'],
                                                color_mode='rgb',
                                                shuffle=False,
                                                batch_size=50)
```

```
conv_layers = [1, 2, 3]
dense_layers = [0, 1]
layer_sizes = [65, 128]

for dense_layer in dense_layers:
    for layer_size in layer_sizes:
        for conv_layer in conv_layers:
            NAME = f"{conv_layer}-Conv-x{layer_size}-neurona-{dense_layer}-Dense-x{layer_size}-neurona-{int(time.time())}"
            tensorboard = TensorBoard(log_dir='logs/{}'.format(NAME))
            filepath = "/Users/Nuxx/checkpoint-{epoch:02d}-{val_loss:.2f}.hdf5"
            checkpoint = keras.callbacks.callbacks.ModelCheckpoint(filepath,
                                                                   monitor='val_loss',
                                                                   verbose=0,
                                                                   save_best_only=True,
                                                                   mode='min')

            print(NAME)

            model = Sequential()

            #Conv1 Layer, input = image
            model.add(Conv2D(layer_size, (3,3), input_shape = (70,70,1)))
            model.add(Activation("relu"))
            model.add(MaxPooling2D(pool_size=(2,2)))

            for l in range(conv_layer-1):
                model.add(Conv2D(layer_size, (3,3)))
                model.add(Activation("relu"))
                model.add(MaxPooling2D(pool_size=(2,2)))

            model.add(Flatten())

            for l in range(dense_layer):
                model.add(Dense(layer_size))
                model.add(Activation("relu"))
                model.add(Dropout(0.2))

            model.add(Dense(2))
            model.add(Activation("softmax"))


            model.compile(loss="categorical_crossentropy",
                          optimizer=adam,
                          metrics=['accuracy'])


            model.fit_generator(my_gen(train_batches),
                                validation_data=(my_gen(valid_batches)),
                                validation_steps=78,
                                epochs=5,
                                steps_per_epoch=625,
                                verbose=1, callbacks=[tensorboard, checkpoint])
```

```
# Dohvaćanje test batcha podataka

test_imgs, test_labels = next(test_batches)
```

```
# Spremanje labela testnog skupa za matricu zbunjenosti

test_labels = test_labels[:,0]
test_labels
```

```
# Pravljenje predikcije nad batchom podataka (Batch od 50 slika)

predictions = model.predict_generator(test_batches, steps=50, verbose=1)
```

```
# Spremanje labela i predikcija te plotanje matrice zbunjenosti

cm = confusion_matrix(test_labels, np.round(predictions[:,0]))

cm_plot_labels = ['dog','cat']
plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')
```