**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKUFAKULTET ELEKTROTEHNIKE, RAČUNARSTVA IINFORMACIJSKIH TEHNOLOGIJA OSIJEK**

**Sveučilišni studij**

**Primjena neuronskih mreža u klasificiranju LEGO kocaka**

**Diplomski rad**

**Anto Tufeković**

**Osijek, 2021.**

SADRŽAJ

[1. UVOD 1](#_Toc76978903)

[1.1. Zadatak završnog rada 1](#_Toc76978904)

[2. STROJNO UČENJE 2](#_Toc76978905)

[2.1. Nadzirano učenje 2](#_Toc76978906)

[2.2. Nenadzirano učenje 3](#_Toc76978907)

[2.3. Nadzirano učenje – Problem klasifikacije 4](#_Toc76978908)

[2.4. Neuronske mreže 4](#_Toc76978909)

[2.4.1. Aktivacijske funkcije 6](#_Toc76978910)

[2.4.2. Treniranje neuronskih mreža 8](#_Toc76978911)

[2.5. Proširenje skupa podataka 10](#_Toc76978912)

[2.6. Skupovi podataka i umjetni podatci 13](#_Toc76978913)

[2.7. Python, Keras, PlaidML 16](#_Toc76978914)

[3. REZULTATI MODELA, USPOREDBE 17](#_Toc76978915)

[3.1. [[SOTA – sintetički podatci]] 17](#_Toc76978916)

[3.2. [[SOTA – pravi podatci]] 18](#_Toc76978917)

[4. [[PYTHON KOD]] 18](#_Toc76978918)

[5. ZAKLJUČAK 18](#_Toc76978919)

[LITERATURA 18](#_Toc76978920)

[SAŽETAK 18](#_Toc76978921)

[ABSTRACT 18](#_Toc76978922)

[ŽIVOTOPIS 18](#_Toc76978923)

[PRILOG A: Popis korištenih modela i njihovi prikazi 19](#_Toc76978924)

# UVOD

Cilj ovog završnog rada je istražiti problem klasifikacije LEGO kocaka. U drugom poglavlju rada će se objasniti osnove strojnog učenja i neuronskih mreža, te osnovni podatci o korištenim ulaznim podatcima. Treće poglavlje služi kao prikaz rezultata modela dobiveni nad podatcima iz drugog poglavlja, te sadrži paralele kakve rezultate daje taj model naspram drugih modela iz područja prepoznavanja slika. Četvrto poglavlje se većinom bavi objašnjenjem korištenog programskog koda te pojašnjenje prijenosnog učenja (engl. *transfer learning*). [[nisam siguran dali je potrebno četvrto poglavlje, tamo sam kontao pojasniti transfer learning]]

## Zadatak završnog rada

Proučiti i opisati problem klasificiranja LEGO kocaka. Predložiti model i dizajnirati sustav za klasificiranje te ga ispitati na odgovarajućem podatkovnom skupu.

# STROJNO UČENJE

Strojno učenje je disciplina proučavanja korištenja neke vrste algoritma koja može sebe modificirati kroz neke određene akcije da daje traženi izlaz na neki dani ulaz. Te algoritme je potrebno trenirati tj. potrebno ih je učiti prije nego se mogu koristiti za svoje svrhe, te po tome ih dijelimo na dvije većinske grupe: nadzirano učenje i nenadzirano učenje.

## Nadzirano učenje

Algoritmi koji pripadaju nadziranom učenju se moraju učiti nad određenim skupom podataka, svaki komad informacije u podatcima mora imati svoju oznaku na kojoj piše šta je točno. Kad se algoritam trenira, on se uči nad ovim podatcima sa oznakama, te time može namještati svoje unutrašnje parametre na takav način da određeni podatci utječu na određen način na parametre, te algoritam to pamti preko danih oznaka, pa nakon treniranja kada se pokušava validirati ispravnost algoritma daju mu se podatci bez oznaka, te algoritam mora odrediti pripadajuću oznaku za dane ulazne podatke.

Neki od problema koji spadaju u nadzirano učenje su:

* Klasifikacija – za određen broj ulaznih podataka algoritam mora moći odrediti njihove pripadnosti u kategorija, npr. na temelju dane slike životinje pripadno trenirani algoritam mora moći odrediti kojoj klasi životinja pripada, tipa ako je dana slika psa algoritam određuje koja je sorta pasmine i slično. Ovaj rad za Lego kocke je tipa klasifikacije, za određene slike Lego kocaka određuje se koje su točno model
* Regresija – na temelju ulaznih podataka pokušava se dobiti pripadajući realni broj koji opisuje najbolje ulazne podatke (za razliku od klasifikacije, regresija daje realan broj dok klasifikacija daje jedan od mogućih izlaznih oznaka), npr. za dane podatke o rabljenom automobilu kao što su kilometraža, marka, model, postotak štete na autu i slično, algoritam može nam dati procjenu vrijednosti automobila
* Prijevod jezika – algoritam prima riječi ili tekst, te njihove prijevode za učenje, te kad se nakon treniranja da neki novi tekst ono mora moći odrediti prijevod danog teksta
* Prepoznavanje grešaka – algoritmu se daju podatci normalnog ponašanja za treniranje, te pri validaciji nad abnormalnim podatcima algoritam mora moći prepoznati visoko odstupanje. Npr. algoritmu se daju slike ispravnih mehaničkih komponenti za trening, te na novim slikama mora uočiti razlike kao što su pukotine ili iskrivljenje materijala[<https://global.canon/en/technology/crack2019.html>]
* Uklanjanje smetni – algoritmu se daju podatci koji ga upućuju na ciljane smetnje te trenira se tako da izmjenjuje podatke dok se smetnje ne uklone. Npr. algoritam se uči na slikama sa pomućenjem, npr. od kamera koje su se kretale pri snimanju, te trenira se tako da smanji utjecaj pomućenja mijenjanjem slike

## Nenadzirano učenje

Algoritmi za nenadzirano učenje se treniraju na drugačiji način naspram algoritama za nadzirano. Oni za trening primaju podatke bez oznaka, te moraju onda pronaći neku vrstu strukture u danim podatcima, kao što su grozdovi ili skupovi podataka ovisno o informacijama u podatcima, te za validaciju onda moraju točno svrstati nove podatke u dane skupove, iako točnost je ovisna o ljudskoj osobi koja procjenjuje kvalitetu kreiranih grupa.

Glavna korist nenadziranog učenja nad skupovima podatka je procjena gustoće (engl. *density estimation*), algoritam prema podatcima kreira polja koja sadrže neki postotak vjerojatnosti da podatak pripada tom skupu, što je bliže „centru“ polja to je veća vjerojatnost da pripada tom skupu.

Neki od problema koji spadaju u nenadzirano učenje su:

* Grupiranje – algoritmu se daju podatci sa određenim informacijama, algoritam pokušava te podatke grupirati u svoje grupe ovisno o njihovim informacijama, postoji više metoda za grupiranje
* K-means – metoda grupiranja gdje algoritam koristi informacije u podatcima da odredi skupova podataka, isprva odabere nasumičnih podataka te onda iterativno traži prosječnu udaljenost prema svim podatcima koji se smatraju pod tim skupom, nakon toga ponovno izračuna središnje točke svih skupova i ponavlja algoritam dok se ne konvergira u mirno stanje (konačno konvergirano stanje ne mora biti optimalno stanje)
* [[TODO plus i primjeri za potkrijepiti]]

## Nadzirano učenje – Problem klasifikacije

Klasifikacija je problem iz područja statistike, u problemu se pokušava doći do strukturiranog rješenja gdje nekoj promatranoj jedinici damo jednu oznaku iz skupa oznaka. Jedan jednostavan primjer bi bila klasifikacija elektronske pošte u ulazni sandučić ili u neželjenu poštu (engl. *spam*).Takvi ulazi moraju u sebi sadržavati neku informaciju (engl. *feature*) koja algoritmu pomaže odrediti kojoj od klasa ulazni podatak pripada, takve informacije mogu biti same po sebi kategorične (npr. osoba može imati krv tipa „A“, „AB“, „A+“, itd.), mogu biti bazirane na brojevima (npr. slike imaju piksele koje sadrže tri kanala za boje, najčešće sa vrijednostima od 0 do 255) ili jednostavno uspoređivati dolazeće podatke sa prijašnje treniranim podatcima tako da uspoređuje udaljenost ili sličnost novih podataka od treniranih.

Algoritam koji obavlja klasifikaciju se naziva klasifikator, ako postoje samo dvije klase za klasificirati onda se metoda zove „binarna klasifikacija“, ako imamo više klasa metoda se zove „klasifikacija s više klasa“ a ako imamo metodu gdje neki izlaz može imati više klasa pridruženo onda se radi o „klasifikaciji s više oznaka“. Ovaj rad će se baviti sa „klasifikacijom s više klasa“ te morat će pridružiti ulaznim slikama izlazne oznake.

Zato što se koriste slike kao ulazni podatci u algoritam klasifikacije s više oznaka, potrebno je odabrati pripadajuću strategiju s kojom se može problem riješiti, te odabrano rješenje ovog rada su umjetne neuronske mreže (engl. *Artificial neural networks*).

## Neuronske mreže

Umjetne neuronske mreže (engl. *Artificial neural networks – ANNs*) su algoritmi koji se baziraju na neuronskim mrežama životinja iz pravog svijeta, neuronska mreža se sastoji od kolekcije spojenih jedinica koje se nazivaju neuroni. Spoj među neuronima je sličan spoju kojeg prave pravi živci u obliku sinapsa, samo za olakšano i sigurnije programiranje tih spojeva modelira se sve u obliku slojeva neurona, slojevi su međusobno spojeni i tako se signal na ulazu propagira prema izlazu, kao što je prikazano na slici 2.4.3.



**Slika 2.4.1.** Uzor neurona je prava biološka živčana stanica [[izvor](https://commons.wikimedia.org/wiki/File:Neuron3.png) slike]

Na slici se vidi da svaki neuron po uzoru akson – sinapsa – dendrit (izlaz – prijenos – ulaz) ima svoje ulaze () na dendritima, neka funkcija stanice u tijelu, te onda preko aksona ima izlaz() kojim se dalje propagira do kraja sustava. Prethodno spomenuta funkcija stanice neurona određuje kakav će izlaz biti na svakom izlazu ovisno o ulazu, funkcija sadrži u sebi težine koje se pri treningu mijenjaju ovisno o stimulusu te onda odlazi u aktivacijsku funkciju. Najjednostavnije objašnjenje je da svaki izlaz neurona je realna funkcija, te težina pojačava ili oslabljuje „signal“ na danom izlazu.



**Slika 2.4.2.** Matematički model neurona[[izvor slike](https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc)]

Osnovni matematički model neurona je da svaki ulaz ima svoju težinu [[izvor za ovu sekciju](http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf)], težine se postavljaju tijekom treninga ovisno o važnosti ulaza. Na kraju neuron izvršava neku funkciju (često zvana aktivacijska funkcija) nad sumom svih ulaza množeni svojim težinama te to predstavlja kao izlaz svim sljedećim spojenim neuronima. [[Ovaj dio bilo bi potrebno uklopiti sa ostatkom teksta]]



**Slika 2.4.3.** Primjer neuronske mreže sa jednim skrivenim slojem

Kao što je na početku poglavlja spomenuto, neuroni su često poslagani u slojevima zbog jednostavnosti kreiranja modela, to ne znači da signal putuje jednom kroz model i završava odmah na izlazu, nego u modelu se mogu nalaziti razne funkcionalnosti između slojeva koje pomažu modelu postići veći uspjeh te glavni cilj tih slojeva je izbjegavanje mrtvih neurona. [[Prvo moramo pregledati kakve aktivacijske funkcije postoje i kako mogu pomoći. OVO možda pomaknuti, pokazati treniranje prvo i zapisati tekst za taj dio pa micati okolo stvari]]

### Aktivacijske funkcije

Aktivacijske funkcije korištene u umjetnim neuronima djeluju nad sumom ulaza množeni sa njihovim pripadajućim težinama. Te funkcije daju drugačije izlaze ovisno o treniranim parametrima težine u neuronima.[[izvor za ovu sekciju](https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/)]

Najčešće korišteni u praksi su:

* ReLu aktivacija (ispravljena linearna jedinica) (raspon ([0, ∞)) i njena derivacija:

|  |  |
| --- | --- |
|  | (2-1) |

* Sigmoid(logistička) funkcija (raspon (0,1)) i njena derivacija:

|  |  |
| --- | --- |
|  | (2-2) |

* Tanh aktivacijska funkcija (raspon (-1, 1)) i njena derivacija:

|  |  |
| --- | --- |
|  | (2-3) |



a) ReLu (2-1) b) Sigmoid (2-2) c) Tanh (2-3)

**Slika 2.4.1.1.** Prikaz aktivacijskih funkcija a), b), c)

Kod ove tri aktivacijske funkcije važno je napomenuti da se može dogoditi problem sa nestajućim gradijentom prilikom treniranja (više objašnjeno u sljedećem podpoglavlju). Glavni princip problema je da kod Sigmoid i Tanh funkcija imamo lokalni maksimum i minimum (Sigmoid ima raspon (0,1) dok Tanh ima (-1,1)), ako se preko SGD i povratnog razmnožavanja naiđu brojevi veći od maksimuma i minimuma, oni se prevedu natrag direktno u maksimum/minimum, te time se onemogućava treniranje neurona. ReLu ima sličan problem ali samo kod minimuma, zato je njegov raspon [0, ∞) te to uvelike pomaže kod treniranja dubokih mreža jer mreže koje se zasnivaju na Sigmoid ili Tanh funkcijama često naiđu na problem nestajućeg gradijenta, jer gradijent se kreće u nekom smjeru pa ako naiđe na minimum ili maksimum pojavljuje se onda problem. ReLu je otporan na maksimume te nema takve probleme.

Ako se trenira mreža sa ReLu i naiđe se na problem nestajućeg gradijenta, moguće je koristiti njegovu alternativu, parametarski ReLu (engl. *parametric ReLu*).



**Slika 2.4.1.2.** Prikaz aktivacijske funkcije parametarski ReLu

Karakteristika ove aktivacijske funkcije je da više nema raspon od 0 do beskonačno, nego mu je modificiran negativan dio koristeći parametar (često je jednak 0.01 pod nazivom „propusni“ tj. engl. *leaky ReLu*):

|  |  |
| --- | --- |
|  | (2-4) |

Na ovaj način se može izbjeći problem nestajućeg gradijenta jer više ne može zapeti na nuli, nego se može kretati slobodno pa postoji šansa da se težina oživi natrag iznad nule. Neuron kojemu su sve težine zapele na negativnoj strani se naziva mrtav neuron, te propusni ReLu služi kao mogućnost oživljavanja takvih neurona.

### Treniranje neuronskih mreža

[[namjestiti ovo i aktivacijske funkcije da ima smisla, potrebno je možda odčlaniti i ubaciti dio za ADAM]]Treniranje neuronskih mreža se izvodi preko stohastičkog gradijentnog spusta (engl. *stochastic gradient descend*) iliti SGD, SGD je iterativna metoda za optimizaciju funkcije gubitka treninga, sa dobivenim gradijentom ono može zajedno sa algoritmom povratnog razmnožavanja [[izvor](http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf) 9-48] prolaziti unatrag kroz sve slojeve neuronske mreže, mijenjajući težine na ulazu neurona. Za ovu svrhu je potrebno da je aktivacijska funkcija neurona derivabilna jer algoritam kako prolazi unatrag mora moći proći i kroz aktivacijsku funkciju neurona da može nastaviti.

Ovdje se pojavljuje problem u treniranju neuronskih mreža, naime problemi nestajućeg i eksplodirajućeg gradijenta [[izvor za ovu sekciju](https://books.google.hr/books/about/Deep_Learning.html?id=Np9SDQAAQBAJ&printsec=frontcover&source=kp_read_button&redir_esc=y#v=onepage&q&f=false)]. Kako se trenira vrlo duboka mreža sa mnogo slojeva, postoji vjerojatnost da će gradijent, kako prilazi početku neuronske mreže, biti toliko malen da će imati neznatan utjecaj na težine, u najgorem slučaju može se dogoditi da se neuronska mreža kompletno prekine trenirati zbog ovog problema. Primjer događanja bi bio da neka aktivacijska funkcija, primjerice Tanh koja ima gradijente samo u rasponu od 0 do 1, kroz povratno razmnožavanje smanji svoj gradijent na eksponencijalno male brojeve kako prolazi kroz slojeve, time uzrokuje da rani slojevi dobiju vrlo malo ili ništa treninga. Sličan problem se ukazuje kod eksplodirajućeg gradijenta kada derivacije aktivacijskih funkcija primaju vrlo visoke vrijednosti, što uznemiruje ravnotežu treniranja sa divljim mijenjanjem težina.

Ciklusi treniranja nad skupom podataka se nazivaju epohe, svaka epoha se provodi nad ulaznim skupom podataka iz kojeg se odabire mini-šarža (engl. *mini-batch*) podataka, koristeći SGD i povratno razmnožavanje ovaj se postupak iterativno ponavlja dok se ne iskoriste sve mini-šarže u skupu podataka. Na kraju epohe se na validacijskom skupu podataka provjeravaju performanse mreže, ti parametri se onda mogu koristiti u sljedećim epohama za namještanje parametara treninga. Kad završi trenutna epoha pokreće se sljedeća dok ne prođu sve zadane epohe. Algoritam treniranja može rano zaustaviti treniranje ako vidi da se funkcija greške ne mijenja znatno preko vremena, ovo je naznaka da je algoritam treniranja došao u lokalni minimum za moguću grešku, te potrebno je dalje namještati parametre ili ako su zadovoljavajući rezultati spremiti model za korištenje u produkciji.

Daljnji problemi pri treniranju se mogu pojaviti u obliku pretreniranja (engl. *overfitting*). Pretreniranje je pojava u kojoj trenirani model postaje previše naviknut na skup podataka na kojem je treniran. To se događa ako je skup podataka za treniranje vrlo malen, ili ako su podatci vrlo slični jedni drugom. Prilikom treniranja mreže potrebno je obratiti pažnju na parametre koje se vraćaju na kraju epohe, koje govore kako model u toj epohi djeluje nad validacijskim skupom. Ako model ima dobre performanse nad trening skupom ali loše performanse nad validacijskim skupom može se reći da pati od pretreniranja nad trening skupom.



**Slika 2.4.2.1.** Grafički prikaz za utjecaj parametara treninga na pretreniranje[[izvor slike](https://datascience.foundation/sciencewhitepaper/underfitting-and-overfitting-in-machine-learning)]

Na slici 2.4.2.1. se vide primjeri tri modela sa drugačijim parametrima treniranja. Prvi model je oblika polinomna regresija sa jednim stupnjem, zbog toga se ne može točno podatcima namjestiti (engl. *underfit*). Drugi model ima četiri stupnja, što je dostatno da se model namjesti trening podatcima i da liči originalnoj funkciji. Treći model ima previše stupnjeva te pokušava se namjestiti na takav način da kroz sve podatke pokušava proći. Tu se nalazi problem pretreniranja.

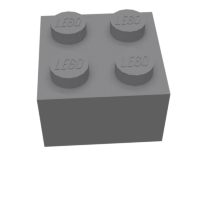
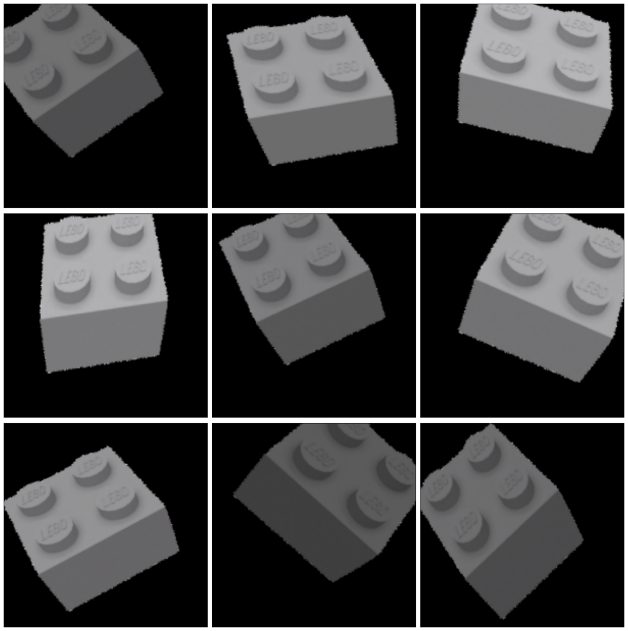
Sličan koncept se može dogoditi i kod neuronskih mreža, ako preko SGD i povratnog razmnožavanja konstantno dolaze slični gradijenti, neuroni će biti naučeni na te specifične podatke, te kad dođu neki drugi dovoljno drugačiji podatci, neuroni neće biti u mogućnosti dati kvalitetan odgovor prema sljedećem sloju, što uzrokuje grešku u npr. predviđanjima i pogoršava parametre kod validacije. Otkrivanje podtreniranja i pretreniranja je jednostavno ako imamo parametre sa treniranja, ako model ima dobre performanse prilikom treniranja i dobre performanse pri validaciji, onda se može reći da je model kvalitetan. Ako model ima loše performanse kod validacije onda je moguće pretreniran, dok loše performanse kod treniranja ukazuju da nije dovoljno treniran, tj. podtreniran je i nedostaje više epoha treniranja.

## Proširenje skupa podataka

Za potrebe treniranja neuronskih mreža može se dogoditi situacija da ne postoji dovoljno dostupnih podataka nad kojima se može trenirati, razni su razlozi kao npr. poštivanje privatnosti u medicini (naprimjer rendgenske, mamografske, ultrazvučne slike i slično) zbog čega gubimo pristup vrijednim podatcima za treniranje, a potrebna je velika količina da se može trenirati kvalitetna mreža, pogotovo ako je duboka mreža, što je dublja to je više raznolikosti u trening podatcima potrebna da se kvalitetno trenira bez problema pretreniranja i podtreniranja.

Za tu svrhu postoji proširenje skupa podataka (engl. *Data augmentation*). To je svaka vrsta tehnike proširenja postojećeg skupa podataka tako da se uvede nasumično transformirana kopija nekog originala iz skupa ili se generira sintetički podatak od originalnih[[izvor](https://www.researchgate.net/profile/Agnieszka-Mikolajczyk-3/publication/325920702_Data_augmentation_for_improving_deep_learning_in_image_classification_problem/links/5d5d5569458515210257607c/Data-augmentation-for-improving-deep-learning-in-image-classification-problem.pdf) za ovo cijelo podpoglavlje? sam izvor ima svoje izvore, možda izvucem nest]. Za kreiranje nasumično transformiranje kopije originala često se koriste jednostavne funkcije kao što su Afine transformacije (rotacija, skaliranje, translacija, refleksija, smicanje), transformacije nad histogramom (npr. ujednačenje), te mijenjanje svojstva slike kao što su kontrast, svjetlina ili transformacije kao povećanje oštrine, pomućenje slike i izjednačavanje svjetline.

Razlog za proširenjem skupa podataka je poboljšanje generalizacijskog svojstva mreže. Ako se kompleksna mreža trenira nad malim brojem podataka za normalan broj epoha, dogoditi će se podtreniranje mreže jer mreža nije dovoljno trenirana da uspješno prepoznaje objekt na slici, ali ako se trenira nad malim brojem podataka za više epoha mreža će početi pamtiti ulazne podatke i dogoditi će se pretreniranje što opet smeta generalizaciji, jer pri treniranju će imati visoku točnost, a pri validiranju nisku točnost.

a) b)

**Slika 2.5.1.** a) Slika modela 3003 LEGO kocke, b) Slika modela 3003 LEGO kocke proširena raznim tehnikama [[staviti sliku novih modela]]

a) b)

**Slika 2.5.2.** a) Uslikana LEGO kocka 3001, b) Uslikana LEGO kocka 3001 proširena raznim tehnikama

Na [sinteticko a)] prvoj slici imamo sintetičku sliku modela 3003 LEGO kocke, takva kocka je slikana iz raznih kutova te nije potrebno koristiti transformacije kao što su refleksija, međutim sve ostale se mogu iskoristiti, na slici [sinteticko b)] se vidi polje istog modela ali prošireno raznim tehnikama. Za uslikane [uslikano a)] se mogu koristiti iste tehnike ali se mogu koristiti i refleksije za dodatno proširenje podataka.

Na [drugoj] slici je prikazan skup od 9 slika nad kojima se izvela nasumična transformacija smicanja, rotiranja, translacije i mijenjanja svjetline. Zato što je izvorna slika iz skupa sintetičkih podataka ona je siva te uslikana je iz svih mogućih smjerova, zato nije poželjno koristiti transformacije kao što su rotacija i refleksija, jer takve transformacije će rezultirati slikama generiranih iz drugih kutova što može slučajno povećati nadtreniranje.

[[Za sintetičke podatke koji su uvijek centrirani koristiti će se sljedeće transformacije]]:

* Nasumično smicanje, maksimalno ±10°
* Nasumično povećanje (engl. *zoom*) slike, maksimalno ±10%
* Nasumična translacija horizontalno, maksimalno ±10%
* Nasumična translacija vertikalno, maksimalno ±10%
* Nasumično mijenjanje svjetline, od 50% do 100%
* [[promijenjeno sa novim modelima]]

[[Poredati ovo i sintetičke podatke da ima više smisla, dodati još teksta]]

## Skupovi podataka i umjetni podatci

Skupovi podataka korišteni u ovom radu se sastoje od uslikanih LEGO kocaka ili umjetno generiranih slika LEGO kocaka. Prilikom treniranja neuronskih mreža za klasifikaciju slika važno je odrediti kakve podatke će mreža primati za treniranje i validiranje.

Prije treniranja neuronskih mreža nad pravim slikama ako je moguće za što veću točnost je potrebno što više standardizirati pozadinu uslikane LEGO kocke. Ako je pozadina neka vrsta reljefa ili jednostavna tekstura, neuronska mreža može sama sebe trenirati da takve značajke ignorira, nego da se koncentrira na detekciju rubova, detekciju oblika, detekciju broja dugmadi na LEGO kockama i drugo potrebno za klasifikaciju. Neuronske mreže su posebne na taj način da im mi ne moramo govoriti šta da traže, one će preko trajanja treninga naučiti same kako otkriti željene značajke.



**Slika 2.6.1.** Uslikana LEGO kocka 3001

Na slici 2.6.1 vidimo jednostavnu sliku narančaste LEGO 3001 kocke. Zadatak mreže bi bio ignorirati pozadinu, te pravilno klasificirati kocku kao 3001 kocku. Kod mreža treniranim nad umjetnim podatcima to je drugačije.



**Slika 2.6.2** Umjetno generirana LEGO kocka 3003

Na slici 2.6.2 se vidi slika umjetno generirane LEGO kocke koristeći program Blender[[izvor](https://www.kaggle.com/joosthazelzet/lego-brick-images)]. Odmah su uočljive razlike kao što su drugačija pozadina i drugačija tekstura LEGO kocke. U ovom koraku bi osoba trebala odabrati jedno od dva puta: generirati podatke na takav način da su značajke modela što vidljivije ili generirati umjetne podatke da što više liče pravim podatcima (lažni podatci).

Prvi slučaj se koristi za istraživanje mogućnosti mreže, kada osoba ima želju istražiti koristi mreže koja je trenirana nad nekim objektom, za početak može generirati jednostavne umjetne podatke da pregleda koliko je mreža efektivna u predviđanju klase. Nakon toga bi se postupilo nabava slika ciljanog objekta te može se prijeći na treniranje za pravi svijet ili se može postupiti drugim slučajem ako nedostaje dovoljno slika za treniranje.

Drugi slučaj bi se koristio kada imamo jako malen broj uzoraka iz pravog svijeta, ali želja je da se mreža koristi u pravom svijetu. Osoba mora pažljivo generirati slike sa raznim parametrima postavljenim da što bolje oponašaju „oko“ u pravom svijetu, pazeći na pozadinu slike, izvore svijetla, materijal i teksturu LEGO kocke i slično.

Umjetni podatci imaju općenito široki raspon svrha za koje se mogu koristiti [[izvor](https://research.aimultiple.com/synthetic-data/)] te najčešće pojednostavljuju i pojeftinjuju operacije za koje su vezane, u slučaju ovog rada se ne mora par dana slikati par LEGO kocaka, nego je moguće generirati sve slike u raznim uvjetima, mogu biti sa raznim pozadinama, mogu dijelovi kocaka biti skriveni, svijetlo na kockama se može namještati, slike se mogu generirati oštećene kao simulacija slikanja u pravom svijetu (primjer je generirana slika sa pomućenjem u nekom smjeru, kao da je osoba koja je slikala micala kameru pri slikanju). [[Umetnuti dodatne sitnice oko podataka?]]



**Slika 2.6.3.** Isječak iz Blenderovog 3D ViewPort-a za prikaz lokacija objekta, svjetla i kamere

Sintetički podatci za ovaj rad se generiraju u Blenderu koristeći skriptu koja uslika zadani model iz raznih kutova, te slike imaju dva svjetlosna izvora u blizu sredine ali su za jednu jedinicu odmaknutu svaka u svoju stranu (na Slici 2.6.3. se vidi da su odmaknute od centra 4 jedinice, u stranu pomaknute za 1 jedinicu te visoko/duboko postavljeno 6 jedinica). LEGO modeli se skaliraju tako imaju točan odnos veličine međusobno (npr. vizualno pregledati da su na svim modelima dugmadi iste veličine, da su sami modeli iz istog izvora, te time garantirano iste skale, da su tehničke kocke iste dužine (2M, 3M, itd.)) i uslikane su sa jednakom udaljenošću od kamere. LEGO modeli su nasumično obojani od par odabranih boja, kao što su plava, ružičasta, siva, itd., neuronske mreže moraju biti otporne na različite boje u slikama, iako jedan mogući zaobilazak problema je da se slike prije treniranja pretvore u slike sivih tonova.



**Slika 2.6.4.** Prikaz načina kretanja kamere u Blenderu

Kamera slika model počevši od skroz gore (slika 2.6.4.), te u zadanim koracima se kreće sferično oko modela prema dolje. Kako prolazi tako kamera „slika“ model, te generirane slike sprema za daljnje korištenje u treniranju.

Za rad su korišteni modeli koji su ilustrirani u Prilogu A. Cilj je bio koristiti modele koji imaju različite stupnjeve sličnosti, više o tome u podpoglavlju (3.1.).

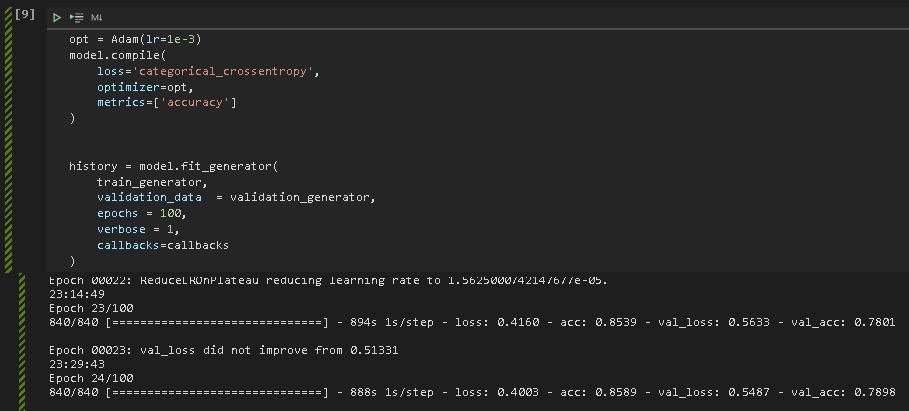
## Python, Keras, PlaidML

Modeli ovoga rada su pisani u *Python* [[izvor](https://www.python.org/)] programskom jeziku koristeći biblioteke *Tensorflow* i *Keras* koje služe za programiranje strojnog učenja [[izvor](https://keras.io/)]. Strojno učenje preko spomenutih biblioteka se može provoditi preko procesora ili grafičke kartice, te korištenje grafičke kartice za treniranje modela je brža opcija naspram procesora (deset puta brže ili čak više). Međutim, treniranje preko grafičke kartice se tradicionalno odvijalo na Nvidia grafičkim karticama jer biblioteke su pisane sa naredbama specifično za te kartice, te za to služi *PlaidML*.

*PlaidML* je biblioteka koja se veže za *Keras* biblioteku na takav način da dopušta raznim grafičkim karticama, kao što su Intel-ove integrirane kartice ili AMD kartice, da preuzmu istu funkciju kao i Nvidia grafičke kartice te ono nam prevodi i kompilira sve njihove naredbe u naredbe koje ostale grafičke kartice razumiju.

*Keras* je biblioteka koja je nastavak na *Tensorflow* biblioteku, ono služi kao posrednik između programera i *Tensorflow* biblioteke te time uvelike olakšava programiranje i namještanje modela te snalaženje u kodu.

Cijelokupno izvođenje *Python* koda se provodi unutar *Jupyter* bilježnica[[izvor](https://jupyter.org/)]. Takve bilježnice se koriste za brzo izvođenje promjena bez potrebe ponovnog paljenja i gašenja *Python* jezgre, te služi nam kao spremnik izlaza *Python* kôda za daljnju analizu (slika 2.7.1.).



**Slika 2.7.1.** Prikaz dijela *Jupyter* bilježnice u kojem se izvodi trening mreže

*Blender* je program[[izvor](https://www.blender.org/)] za 3D modeliranje, te sadrži mogućnosti za izradu slika i animacija modela. Koristi se u ovom radu za generiranje slika sintetičkog skupa.

Cijeli projekt za ovaj rad se nalazi na sljedećoj github poveznici: [[kad postavim public stavit ću]]

Projekt sadrži dva direktorija, *docs* i *py*. Unutar *docs* se nalaze svi materijali vezano za pisani rad i ovaj sam rad. Unutar *py* se nalazi više poddirektorija, svaki poddirektorij sadrži svoju *Jupyter* bilježnicu unutar koje se izvršavao *Python* kôd. Direktoriji sa prefiksom „test“ su direktoriji koji sadrže neuronske mreže nad kojima se treniralo, te sve funkcije vezane za njih. Direktorij „image\_generation“ sadrži *Blender* skriptu i *.blend* datoteku za koju je skripta namijenjena. Datoteka sadrži sve modele i okolinu za slikanje slika, dok skripta služi za automatizaciju generiranja slika. Direktorij „image\_sourcing“ sadrži *Jupyter* bilježnicu za generiranje slika za korištenje u ovom dokumentu.

# REZULTATI MODELA, USPOREDBE

[[Ovdje bi trebao krenuti opisati *state of the art*, na koji nacin?]]Klasificiranje slika su ljudi izvodili već duže vrijeme, AlexNet[[izvor](https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf)] je jedan primjer neuronske mreže koja je 2012. godine u ImageNet natjecanju široko pojasnog prepoznavanja postigla prvo mjesto, pobijedivši drugo mjesto za više od 10% manje grešaka u 5 najvećih kategorija koje predstavljaju greške[[top-5 error prevesti ljepše?]], najviše jer su koristili treniranje preko grafičkih kartica. Ovo je privuklo globalnu pozornost prema koristi područja umjetnog učenja i umjetne inteligencije te ubrzo je krenulo napredovanje tih tehnologija. Dan danas se rezultati tehnologija nalaze u raznim poljima svakodnevnice, kamere koriste tehnologije prepoznavanja lica i aktivnog stabiliziranja videosnimke, doktori sa modelima mogu lakše uočiti bolesti na raznim vrstama snimaka, od EKG do rendgenskih slika te moguće je lakše prevesti pokrete tijela u pokrete objekata virtualne stvarnosti uz pomoć neuronskih mreža (npr. za koristi u arhitekturi[[izvor](https://www.davidpublisher.com/Public/uploads/Contribute/5d5e0535ad919.pdf)]).

## [[SOTA – sintetički podatci]]

U poglavlju [[stagod za sintetičke](#_Skupovi_podataka_i)] su pojašnjene koristi i mane sintetički generiranih podataka, za slučaj u radu znamo da LEGO kocke imaju oštro definirane oblike (nikada se dvije kocke istog modela ne razlikuju, osim možda po boji), kocke imaju rubove i dugmadi, ploče imaju glatke površine bez dugmadi, štapići za povezivanje su cilindričnog oblika sa specifičnim reljefima na krajevima, kotači su široki cilindri sa reljefima na kontaktnoj površini, itd.

Za grafički prikaz LEGO kocaka je potreban model[[sekcija za primjer renderiranja, potrebno?]]

**Tablica 3.1.1.** Prikaz rezultata za korištene modele

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | loss | acc | val\_loss | val\_acc |
| test | 2.8129 | 0.1097 / 10.97% | 2.6149 | 0.1384 / 13.84% |
| test\_complex | 1.3817 | 0.5200 / 52% | 1.1055 | 0.5796 / 57.96% |
| test\_vgg16 | 0.3197 | 0.8916 / 89.16% | 0.4053 | 0.8440 / 84.40% |
| test\_vgg19 | 0.3107 | 0.8903 / 89.03% | 0.4430 | 0.8230 / 82.30% |

Nad sintetičkim podatcima jednostavne mreže mogu postići visok rezultat kao što je vidljivo na tablici 3.1.1. Problem koji je spomenut na kraju podglavlja 2.6. je vezan za sličnosti među modelima, uzmimo za primjer model 3001 2x4 kocku i model 3020 2x4 ploču (slike se nalaze u prilogu A). Ako se trenira mreža sa slikama modela od skroz gore ili dolje, model će naučiti da su to značajke ta dva modela, ali kad dođe u pitanje predviđanje modela takve slike, mreža neće moći sa potpunom sigurnošću reći koji je to model osim ako nije slikano dovoljno u stranu, da se vidi debljina modela. Slično se može produžiti na model 3037 2x4 kocke za krov, ako se da slika od dolje to je još jedan sličan model za mrežu, te tu nije teško da se događaju greške, zbog toga nije moguće da mreža postigne neku vrlo visoku točnost. Paralelno tome je odabrano još par modela koji su ili slični ili jedinstveni u izgledu, naprimjer model 2340 kormilo i 4083 ograda, ti modeli su iz skoro svih pogleda relativno jedinstveni, dok postoje dodatni parovi za otežavanje predviđanja mreže kao što su model 2357 1x2x2 rub kocka i model 2420 1x2x2 rub ploča, gdje je jedina razlika u debljini modela.

[[Ovaj paragraf možda izostaviti? ili dalje doraditi/premjestiti ili obraditi kasnije kao zasebnu mrežu]]Jedno moguće rješenje ovog problema je uvođenje „druge kamere“, tj. dodati još jedan unos u neuronsku mrežu[[izvor](https://www.kaggle.com/joosthazelzet/one-versus-two-camera-setup-to-recognize-lego/log)]. Ako mreža ima pristup dvjema slikama istog modela, može postići bolju točnost prilikom predviđanja modela u slikama.

Ako se uzme u obzir nesigurnost predviđanja mreže iz prošlog paragrafa, očekuje se predviđena maksimalna točnost od oko 90-93% za 25 klasa, te najbolje aktualno trenirane mreže su postigle točnost od oko 90%, što podupire raspon postotka.

Gore spomenuti raspon postotaka je razlog poteškoća uspoređivanja mreže sa ostalim neuronskim mrežama, zajedno sa problemom generalne klasifikacije: mreža iz rada klasificira specifične modele LEGO kocaka, dok mreže iz općih radova klasificiraju nad nekim poznatim podatkovnim skupovima kao što su *imagenet* [[izvor](https://image-net.org/)]. Time ostale mreže mogu najčešće klasificirati LEGO kocku samo kao „LEGO kocka“, umjesto „LEGO 3003 2x2 kocka“.

Ipak se može usporediti u nekim pogledima modificirana VGG16 mreža sa ostalim mrežama iz istog područja, kao što je mreža Jacques Mattheija [[izvor](https://jacquesmattheij.com/sorting-lego-many-questions-and-this-is-what-the-result-looks-like/#accuracy)] gdje se mreža koristi da klasificira tip modela (dali je uslikan model kocka, ograda, *Technic*, vegetacija, itd.), na 60000 primjera podatkovnog skupa za treniranje ima točnost od oko 95%, više za češće modele kao kocke, manje za rjeđe modele kao što su npr. rotacijska ploča (engl. *turntable*). Razlika u točnosti među mrežama dolazi od toga da Mattheijova mreža ne mora točan model odrediti, makar mu se da slika kocke sa strane ili od gore, ono će samo razmišljati kojem skupu pripada.

Još jedan primjer je stroj kojeg je izradio Daniel West [[izvor](https://towardsdatascience.com/a-high-speed-computer-vision-pipeline-for-the-universal-lego-sorting-machine-253f5a690ef4)] iz inspiracije prethodno spomenutog Mattheijovog stroja. West spominje da je njegov stroj „sljedeći korak u evoluciji“, po tome što ima sposobnost prepoznavanja LEGO kocaka po modelu. To znanje se koristi u stroju za sortiranje tako da se modeli grupiraju po tipu objekta i zajedno svi spreme (slično kao Jacquesov stroj) u zajedničke kante (kocke u kocke, grede sa gredama, komadi figurica sa ostalim komadima, itd.). Uspjeh stroja leži u načinu kako je koncipiran. Originalno je treniran nad sintetičkim podatcima, te kad mu se daju stvarne slike iz pravog svijeta, ima lošu točnost predviđanja. Koristeći nasumično unaprjeđenje skupa podataka, West je prestao pokušavati generirati što sličniju sliku realnome nego je krenuo generirati slike koje se više koncentriraju na specifične detalje modela. Sa povećanjem točnosti na kraju je West počeo koristiti bilježenje oznake uz pomoć njegove mreže, mreža mu je ponudila što ono misli da je najbliže predviđanje pa West mora odabrati koji je to točno model, te na tom odabiru mreža se uči. West tvrdi da je oko 100,000 slika uspješno označio na ovaj način. Jedini problem je što West nije objavio nikakve podatke, metrike ili označene slike u javnosti, te zbog toga se ne može pravilno uspoređivati sa našom mrežom.

## [[SOTA – pravi podatci]]

# [[PYTHON KOD, potrebno?, tu sam kontao transfer learning obraditi]]

# ZAKLJUČAK

# LITERATURA

# SAŽETAK

# ABSTRACT

# PRILOG A: Popis korištenih modela i njihovi prikazi

|  |  |  |  |
| --- | --- | --- | --- |
| 2340 Rudder 1x4x3 |  | 3040 Roof tile 1x2 45° |  |
| 2357 Brick corner 1x2x2 |  | 3298 Roof tile 2x3 33° |  |
| 2420 Plate corner 2x2 |  | 4083 Hanger 1x4x2 |  |
| 3001 Brick 2x4 |  | 4276 Worm |  |
| 3003 Brick 2x2 |  | 4286 roof tile 1x3 33° |  |
| 3004 Brick 1x2 |  | 4864 Wall element 1x2x2 |  |
| 3005 Brick 1x1 |  | 6143 Brick D16 with cross |  |
| 3010 Brick 1x4 |  | 6632 Lever 3M |  |
| 3020 Plate 2x4 |  | 18575 Double conical wheel Z20 1M |  |
| 3022 Plate 2x2 |  | 32140 Technic angled beam 4x2 90° |  |
| 3023 Plate 1x2 |  | 41678 Cross block fork 2x2 |  |
| 3024 Plate 1x1 |  | 99301 Roof tile inside 3x3 33° |  |
| 3037 Roof tile 2x4 |  |  |  |