Contents

[1. Uvod 1](#_Toc73109201)

[1.1. Zadatak završnog rada 1](#_Toc73109202)

[2. Strojno učenje 2](#_Toc73109203)

[2.1. Nadzirano učenje 2](#_Toc73109204)

[2.2. Nenadzirano učenje 3](#_Toc73109205)

[2.3. Nadzirano učenje – Problem klasifikacije 4](#_Toc73109206)

[2.4. Neuronske mreže 4](#_Toc73109207)

[2.4.1. Aktivacijske funkcije 6](#_Toc73109208)

[2.4.2. Treniranje neuronskih mreža 8](#_Toc73109209)

[2.5. Skupovi podataka i umjetni podatci 11](#_Toc73109210)

[3. [[Treće poglavlje – prvi postupak]] 14](#_Toc73109211)

[3.1. Treniranje modela koristeći umjetne podatke sa lagano uočljivim značajkama 14](#_Toc73109212)

* UVOD
  + Zadatak
* [[KORIŠTENE TEHNOLOGIJE]]
  + Strojno učenje
    - Vrste algoritama (neke navesti, te dovesti u ciljano nadgledano učenje -> klasifikacija)
  + Nadgledano učenje
    - Klasifikacija
  + Modeli strojnog učenja
    - Umjetne neuronske mreže
  + Python
    - Tensorflow  
      [[i njeni slojevi ili da to objasnim kako nailazim na slojeve u "izrada modela"?]]
  + Blender
    - [[Postupak za umjetne podatke]]
* [[ISTRAŽIVANJE PODATAKA]]
  + Skup podataka
    - Pravi i umjetni podatci
  + Istraživanje odabranog skupa podataka koristeći Python
* [[IZRADA MODELA]]
  + [[Primjer jednostavne mreže]]  
      [[Analiza grešaka jednostavne mreže, npr ako ima overfitting]]
  + [[? kontam ići dalje sa kompleksnijim mrežama koje bi riješile po mogućnosti npr. overfitting i slične probleme]]
  + [[?]]
* ZAKLJUČAK
* ...

# Uvod

Cilj ovog završnog rada je istražiti kreiranje modela koji su trenirani za prepoznavanje nekog predodređenog skupa LEGO kocaka. To se može postići na više načina, te većina rada će se vrtjeti u Python programskoj okolini.

TODO

## 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:
* Sigmoid(logistička) funkcija (raspon (0,1)) i njena derivacija:
* Tanh aktivacijska funkcija (raspon (-1, 1)) i njena derivacija:



a) ReLu b) Sigmoid c) Tanh

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*):

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]]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.

## 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 2.5.1 – Uslikana LEGO kocka 3001

Na slici 2.5.1 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 2.5.2 – Umjetno generirana LEGO kocka 3003

Na slici 2.5.2 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 podataka?]]

## 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 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.

[[TODO: Github link na stvari ovog rada ovdje, možda još dodatne stvari ubaciti ali to planiram napraviti u sljedećem poglavlju]]

# [[Treće poglavlje – prvi postupak za state of the art]]

[[Ovdje bi trebao opisati *state of the art* koje su drugi ljudi već izveli]]Klasificiranje

## Treniranje modela koristeći umjetne podatke sa lagano uočljivim značajkama