Korištenje otvorenih biblioteka za strojno učenje

Črnčec, Patrik

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Patrik Črnčec

KORIŠTENJE OTVORENIH BIBLIOTEKA ZA STROJNO UČENJE

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Patrik Črnčec

Matični broj: 45770/17–R

Studij: Informacijski sustavi

KORIŠTENJE OTVORENIH BIBLIOTEKA ZA STROJNO UČENJE

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Darko Andročec

Varaždin, lipanj 2020.

Patrik Črnčec

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj rad se bavi strojnim učenjem i njegovim korištenjem u svrhu izrade aplikacije za kolorizaciju crno-bijelih slika. Kao pomoć pri izradi aplikacije koristile su se otvorene biblioteke za strojno učenje. U prvom dijelu rada je objašnjeno strojno učenje, njegova podjela i njegovi glavni algoritmi. Također su objašnjene i međusobno uspoređene neke od najpopularnijih otvorenih biblioteka za strojno učenje. U drugom dijelu je kroz programski kod objašnjen rad izrađene aplikacije te algoritma strojnog učenja koji se je koristio pri njezinoj izradi. Za kraj su prikazani rezultati aplikacije zajedno s evaluacijom implementiranog modela.

Ključne riječi: strojno učenje; biblioteke; programiranje; crno-bijele slike; kolorizacija; autoenkoder

Sadržaj

1. Uvod

Strojno učenje je postalo u posljednjih nekoliko godina veoma popularno zbog svojih brojnih primjena. Njezina velika primjena je moguća zahvaljujući sve boljim računalnim performansama te sve većoj količini podataka koje je moguće upotrijebiti za treniranje modela strojnog učenja. Kako su često modeli strojnog učenja složeni, razvijene su otvorene biblioteke koje znatno olakšavaju i ubrzavaju implementaciju algoritama strojnog učenja.

Cilj ovog rada je objasniti strojno učenje i njegove glavne algoritme te navesti i opisati otvorene biblioteke za strojno učenje koje se danas najčešće koriste. Osim toga, cilj je uz pomoć otvorenih biblioteka za strojno učenje izraditi aplikaciju za kolorizaciju crno-bijelih slika. Ideja ove aplikacije je proizašla iz uočavanja da prethodne metode za kolorizaciju crno-bijelih slika su bile vremenski neefikasne te su zahtijevale stručno znanje u obradi slika. Jedna od takvih neefikasnih metoda je bojanje slika u popularnom alatu Adobe Photoshop, gdje čovjek kolorizira sliku na temelju njegove vlastite percepcije i mišljenja kako bi kolorizirana verzija takve slike mogla izgledati. Za razliku od toga, algoritam strojnog učenja je u stanju s određenom mjerom preciznosti naučiti preslikavanje crno-bijelih piksela u RGB piksele te tako automatizirati proces kolorizacije.

Ovaj se završni rad sastoji od sedam poglavlja, gdje se od drugog do četvrtog poglavlja opisuje teorijska podloga strojnog učenja i njezinih otvorenih biblioteka, u petom poglavlju se opisuju metode strojnog učenja koje su korištene pri izradi aplikacije, dok se u šestom poglavlju opisuje rad izrađene aplikacije te donose njezini rezultati. Rad završava sa sedmim poglavljem u kojem se izvodi konačni zaključak.

2. Osnove strojnog učenja

Strojno učenje je područje umjetne inteligencije koje pruža sustavima svojstvo da uče kroz iskustvo bez da su eksplicitno programirani. Takvi sustavi koriste velike količine podataka kako bi kroz proces učenja naučili u njima prepoznati uzorke koje će utjecati na donošenje odluka u budućnosti [1].

Strojno učenje omogućuje rješavanje problema koji su prije njegove pojave bili nerješivi. Primjerice, prije nije bilo moguće napraviti program koji bi uspješno mogao raspoznati da li se na nekoj slici nalazi pas ili ne, te ako se nalazi, u kojem se to području slike nalazi. Pisanje koda koji bi provjeravao boju svakog piksela i na temelju toga donio odluku bilo bi besmisleno i nemoguće jer sigurno ne bismo mogli pokriti sve slučajeve kao što su različite pasmine, različite pozadine (npr. snijeg, livada, unutrašnjost kuće…), kut slikanja, rezolucije slika, crno-bijele slike, osvjetljenje… Strojno učenje ovaj problem može riješiti tako što će algoritam uz pomoć odabranih podataka odnosno slika kroz vrijeme naučiti koje sve elemente slika mora sadržavati da bi se na njoj potencijalno mogao nalaziti pas. Neki od elemenata mogu biti primjerice pasje uši, rep, oblik tijela, proporcije tijela i slično.

2.1. Područja primjene strojnog učenja

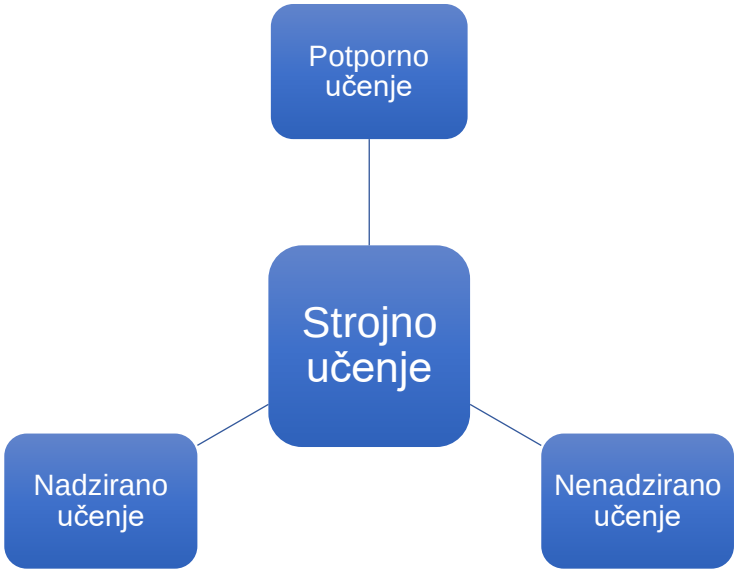
Prethodni primjer je samo jedan od mnogobrojnih primjena strojnog učenja u stvarnom svijetu. Zbog njegove velike moći, strojno učenje se upotrebljava u gotovo svim industrijama, te vjerojatno postoji manji broj područja u kojima se strojno učenje ne upotrebljava. U nastavku će biti nabrojena neka od područja njegove primjene te će se za pojedino područje objasniti na koji se način primjenjuje.

- Trgovine mogu koristiti strojno učenje za analizu njihovih podataka o prodaji kako bi saznali ponašanje njihovih kupaca te tako poboljšali njihov odnos. Osim toga, moguća je i upotreba chatbotova kao pomoć kupcu pri kupovini proizvoda kroz pružanje informacija o trgovini i njezinim proizvodima ili uslugama.
- Financijske institucije analiziraju prošle transakcije kako bi predvidjele kreditne rizike klijenata i tako odbile ili prihvatile zahtjev za kredit klijenta.
- Roboti uče optimizirati svoje ponašanje kako bi mogli završiti neki zadatak koristeći što manje resursa. Također, roboti mogu koristiti tehnike strojnog učenja kako bi se mogli snaći u prostoru.
- U bioinformatici se strojno učenje koristi za analizu bioloških podataka.

- U automobilskoj industriji strojno učenje se koristi za implementaciju autonomnih vozila.
- Digitalni asistenti mogu prepoznati pitanje koje je postavio čovjek te na to pitanje dati smislen odgovor ili izvršiti odgovarajuću akciju [2].

3. Podjela strojnog učenja

Strojno učenje nije samo jedan algoritam koji omogućava prethodno navedene funkcionalnosti, već je to jedna velika skupina algoritama koja se dijeli na tri glavna područja. Svaki od tih algoritama ima svoju svrhu, a odabir algoritma, tj. metode koja će se koristiti pri implementiranju rješenja ovisi ponajviše o zadanom problemu. Prema tome, strojno učenje se dijeli na: **nadzirano učenje, nenadzirano učenje i podržano učenje.** U nastavku će se detaljnije objasniti svako područje te će se objasniti njihovi glavni algoritmi.



Slika 1: Podjela strojnog učenja

3.1. Nadzirano učenje

Glavna zadaća algoritama nadziranog učenja je generiranje funkcije koja preslikava ulaze na željene izlaze u svrhu predviđanja (regresija) ili klasificiranja podataka. Izlazni podaci iz klasifikacijskih modela su diskretne vrijednosti dok su kod prediktivnih kontinuirane. Kod klasifikacijskog modela postoje klase koje se moraju odrediti prije procesa treniranja zbog toga što su one ujedno i izlaz na temelju kojeg algoritam uči. Klase su konačan skup podataka koje definira čovjek tako da svaka klasa sadrži one podatke koji su međusobno slični ili identični po jednom ili više obilježja. Kod prediktivnih modela se uz pomoć obilježja ulaznog podatka predviđa izlazna vrijednost, konkretnije neki realan broj [3].

Proces učenja kod nadziranog strojnog učenja se dijeli na dvije faze a to su treniranje i testiranje. Kako bi to bilo moguće, potrebno je naš skup podataka podijeliti u dvije zasebne cjeline, u podatke za treniranje i testiranje gdje će najveći udio poprimiti podaci za treniranje. U fazi treniranja se podaci za treniranje uzimaju kao ulazni te se njihove karakteristike uče uz pomoć odgovarajućeg algoritma. Glavni zadatak algoritma je predvidjeti izlazni podatak na temelju karakteristika ulaznog podatka kako bi onda svoju predviđenu vrijednost mogao usporediti sa stvarnim izlaznim podatkom i pritom otkriti greške. Nakon toga se na temelju pronađene greške model modificira kako bi buduće predikcije mogle biti preciznije, što zapravo daje svojstvo učenja. U fazi testiranja se uzima istrenirani model te se uz pomoć njega rade predikcije nad podacima za testiranje. S obzirom na to da podaci za testiranje nisu bili korišteni prilikom treniranja modela, rezultat predikcije nad njima će nam poslužiti za evaluaciju, odnosno dobit ćemo informaciju o tome da li je naš istrenirani model dovoljno dobar za naše potrebe ili ga je potrebno dodatno usavršiti [3].

Primjer predviđanja uz pomoć nadziranog učenja je predviđanje cijene kuće na temelju njezine lokacije, veličine, starosti (godine izgradnje), interijera i slično. Primjer klasifikacije može biti prepoznavanje i klasifikacija rukom napisanih brojeva gdje sustav na temelju slike određuje koji se broj nalazi na slici. U ovom su slučaju klase znamenke u dekadskom brojevnom sustavu (0-9).

3.1.1. Linearna regresija

Linearna regresija je jednostavan algoritam potpornog učenja čija je glavna zadaća predikcija podataka, što znači da se ne koristi za klasifikaciju podataka. Ovaj algoritam predstavlja modeliranje odnosa između kontinuirane skalarne varijable (izlazna vrijednost) i jedne ili više nezavisnih varijabli (ulazne vrijednosti) koje ujedno mogu i biti višedimenzionalne [3].

Naše ulazne podatke možemo označavati s $x_1, x_2, \dots, x_n$, međutim jednostavnije je sve ulazne podatke označiti s jednom varijablom X koja će predstavljati skup svih ulaznih podataka odnosno vektora. Također, moramo znati razlikovati predviđenu izlaznu vrijednost algoritma i stvarnu izlaznu vrijednost. Predviđene izlazne vrijednosti se najčešće označavaju s $\hat{y}$, dok se stvarne označavaju jednostavno s y. U linearnoj regresiji, ali i u mnogim drugim algoritmima nadziranog učenja su nam potrebni parametri koji se kroz rad algoritma modificiraju. Drugi naziv za njih su i težine te se one najčešće označavaju sa slovom w [4].

Model linearne regresije se definira uz pomoć jednadžbe:

$$\hat{\boldsymbol{y}} = \boldsymbol{w}^{\boldsymbol{T}} \cdot \boldsymbol{x} + \boldsymbol{b}$$

Težine w služe za kontroliranje ponašanja sustava. Svaka pojedina težina govori o tome koliko svaka karakteristika ulaznog podatka utječe na finalnu predikciju $\hat{y}$. Ukoliko se neka karakteristika x_i pomnoži s pozitivnom težinom w_i, onda će povećanje vrijednosti te karakteristike povećati i vrijednost predikcije. Vrijedi i suprotno, ako se karakteristika pomnoži s negativnom težinom, onda će se smanjiti vrijednost predikcije. Što je veća vrijednost neke težine to ona više utječe na konačan ishod predikcije. U jednadžbi linearne regresije također imamo i varijablu b koja označava odsječak funkcije što osigurava funkciji da ne mora nužno proći kroz ishodište [4].

Prethodna jednadžba nam služi za izračun predikcije linearnom regresijom, no kao što je već to prije objašnjeno, najprije je potrebno istrenirati model i testirati njegove rezultate. U nastavku će se objasniti način na koji se može efikasno izračunati pogreška linearne regresije koja govori koliko predikcije algoritma griješe u odnosu na stvarne podatke za treniranje ili testiranje. Uz pomoć toga će se i konstruirati algoritam za učenje gdje će se ažurirati težine na način da se što više smanji pogreška modela.

Postoji više načina na koji se može izračunati pogreška modela, a jedan od njih je izračun srednje (prosječne) kvadratne pogreške. Ona se izračunava slično kao i varijanca, samo što se ovdje izračunava prosječna kvadratna razlika između predviđene vrijednosti i stvarne vrijednosti, tj. $\hat{y}$ i y. Prema tome, srednja kvadratna pogreška se izračunava preko formule:

$$SKP = \frac{1}{n}\sum_{i}(\hat{y} - y)_i^2$$

S obzirom na to da želimo imati što manju grešku, cilj algoritma je minimalizirati tu grešku da bude što bliža nuli. Ona se može minimalizirati promjenom težina na odgovarajuće vrijednosti, a to ćemo realizirati tako što ćemo gradijent srednje kvadratne pogreške izjednačiti s nulom. Iz te se jednadžbe onda može dobiti jednadžba za izračun težina. Napomenimo i da se ovdje kao ulazni podaci X uzimaju samo podaci za treniranje, dok se podaci za testiranje koriste samo kod evaluacije modela linearne regresije kao što je to već i ranije objašnjeno.

$$\nabla_w SKP = 0$$

$$\nabla_w \frac{1}{n}\|\hat{y} - y\|_2^2 = 0$$

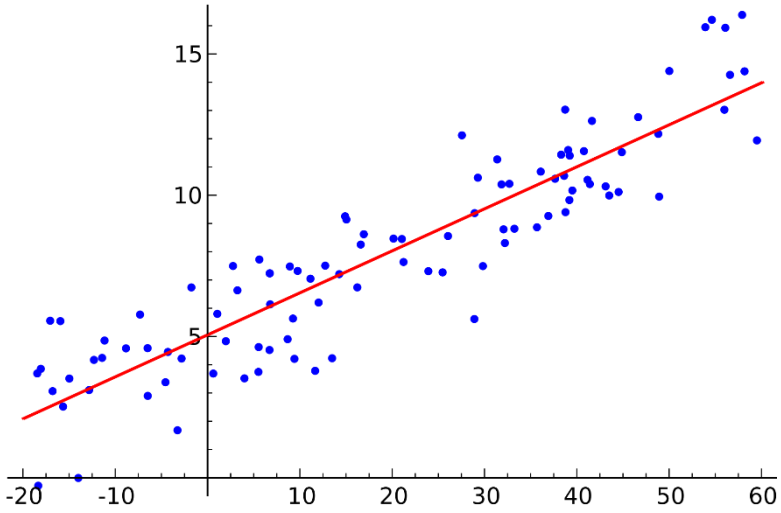
$$\nabla_w (X \cdot w - y)^T (X \cdot w - y) = 0$$

$$2 \cdot X^T \cdot X \cdot w - 2 \cdot X^T \cdot y = 0$$

$$\boldsymbol{w = \left(X^T \cdot X\right)^{-1} X^T \cdot y}$$

Ovakav sustav jednadžbi se naziva normalne jednadžbe [4].

Na sljedećoj slici se nalazi jedan grafički prikaz linearne regresije u koordinatnom sustavu. Možemo primijetiti da je linearna regresija naučila da dvodimenzionalni podaci koji su označeni s plavom bojom budu što bliže linearnom pravcu, što je rezultat minimaliziranja srednje kvadratne pogreške.



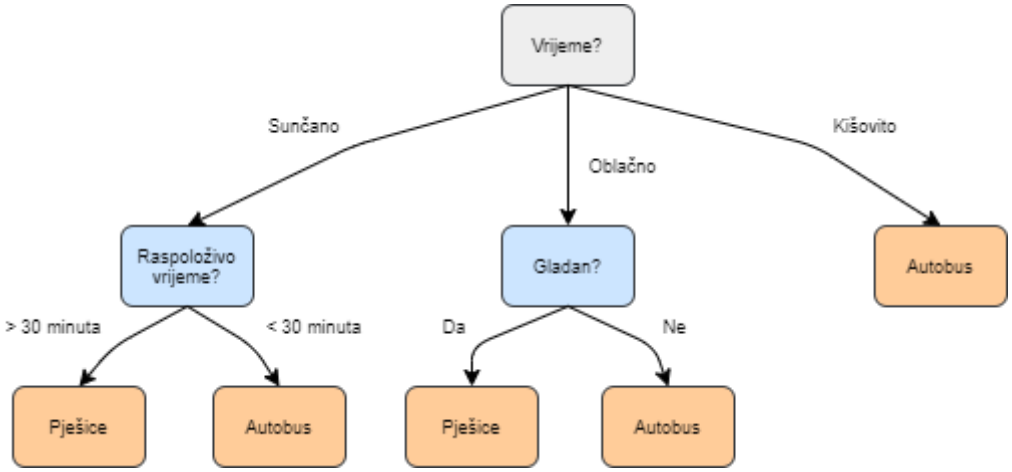
Slika 2: Primjer linearne regresije [5]

3.1.2. Stablo odlučivanja

Stablo odlučivanja je neparametrijski hijerarhijski model za nadzirano učenje koji se sastoji od unutarnjih čvorova odlučivanja i krajnjih listova. To je neparametrijski model zato što struktura stabla odlučivanja nije fiksna, već se broj čvorova i krajnjih listova može povećavati tijekom procesa učenja. Svaki čvor m implementira neku testnu funkciju $f_m(x)$ s diskretnim rezultatima koji označavaju grane u stablu. Ovu funkciju i njezinu rezultate je najlakše opisati kao *ako-onda* pravilo, što čini ovu metodu strojnog učenja lako za interpretirati. U ovom algoritmu se podatak primjenjuje u testnoj funkciji (na čvoru odlučivanja) te se ovisno o rezultatu funkcije odlučuje koja će grana biti uzeta u obzir. Taj proces se rekurzivno ponavlja tako dugo dok se ne dođe do nekog krajnjeg lista koji predstavlja izlazni podatak odnosno rezultat algoritma. Rezultat može biti kontinuirana ili diskretna vrijednost, što znači da ovaj algoritam omogućuje i regresiju i klasifikaciju podataka. Zbog toga, postoje dvije vrste stabla odlučivanja a to su klasifikacijska i regresijska stabla odlučivanja koja će biti objašnjena u narednim potpoglavljima [2].

Na slici ispod se nalazi jednostavan primjer klasifikacijskog stabla odlučivanja. Cilj je odlučiti se na koji će se način doći od neke točke A do točke B, da li pješice ili autobusom. O

toj odluci ovisi kakvo je vani vrijeme, da li je osoba gladna te koliko vremena ima na raspolaganju.



Slika 3: Primjer stabla odlučivanja (Prema: [6])

3.1.2.1. Klasifikacijska stabla odlučivanja

U klasifikacijskim stablima odlučivanja se kvaliteta podjele unutar stabla mjeri s mjerom nečistoće. Podjela je čista ako nakon nje svaka instanca koja odabire neku odgovarajuću granu pripada istoj klasi kao i ostale instance za tu granu. Čvor je čist ukoliko ne postoji klasa čije se instance samo djelomično mogu pojaviti u tom čvoru. Drugim riječima, u nekom se čvoru moraju pojaviti ili sve instance neke klase ili pak nijedna od njih, što znači da vjerojatnost pojavljivanja klase u čvoru smije biti samo 0 ili 1. Ukoliko je podjela čista ne trebamo više dijeliti naše stablo, već možemo dodati krajnji čvor čija će oznaka biti klasa koja je na prethodnom čvoru imala vjerojatnost pojavljivanja 1 [2].

Postoje nekoliko različitih algoritama koji služe za podjelu unutar stabla, a njihov cilj je minimalizirati nečistoću. Neki od njih su Ginijev koeficijent i informacijska dobit. Informacijska dobit određuje koja karakteristika nam daje najviše informacija o klasi. Temelji se na entropiji koja predstavlja stupanj nečistoće. Ginijev koeficijent mjeri vjerojatnost da neka nasumično odabrana varijabla bude pogrešno klasificirana. Ukoliko on iznosi 0, onda to znači da svi elementi pripadaju jednoj klasi, a ako iznosi 1 onda svi elementi imaju različite klase. Možemo ga izračunati uz pomoć sljedeće formule:

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

gdje p_i predstavlja vjerojatnost da objekt bude klasificiran za neku određenu klasu. Kako bismo podijelili čvorove, potrebno je za svaku karakteristiku izračunati Ginijev koeficijent. Ona karakteristika koja ima najmanji koeficijent će se uzeti za idući čvor [7].

3.1.2.2. Regresijska stabla odlučivanja

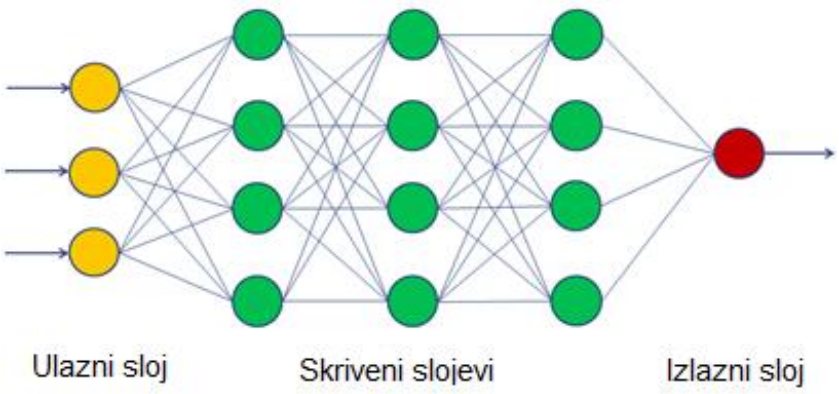
Regresijska stabla odlučivanja su po strukturi gotovo identična klasifikacijskim stablima, a razlikuju se po načinu mjerenja nečistoće stabla. Ovdje se kvaliteta podjele čvora mjeri pomoću srednje kvadratne pogreške koja je već ranije bila objašnjena, dok se predviđena vrijednost za svaki čvor mjeri uz pomoć aritmetičke sredine:

$$\hat{y} = \bar{y} = \frac{1}{N}\sum_{i} y^i$$

gdje je N ukupan broj podataka za treniranje koji se nalaze na određenom čvoru, a y^i stvarna izlazna vrijednost za i-tu instancu podatka. S obzirom na to da ovdje $\hat{y}$ računamo uz pomoć aritmetičke sredine, jednadžba srednje kvadratne pogreške će zapravo ujedno biti ista kao i jednadžba za izračun varijance. Novi se čvor kreira ukoliko je pogreška (varijanca) na promatranom čvoru prihvatljiva, tj. ako je pogreška manja od one vrijednosti koje smo si mi zadali kao prihvatljivu te će imati predviđenu vrijednost $\hat{y}$. Ukoliko pogreška nije prihvatljiva, podaci koji dopiru do čvora se dijele dalje na način da zbroj pogrešaka u granama bude minimalan [2].

3.1.3. Neuronske mreže

Neuronske mreže su skup različitih modela koje omogućuju klasifikaciju i regresiju podataka. Sastoji se od umjetnih neurona koji su podijeljeni na slojeve, konkretnije na ulazni, skriveni i izlazni sloj. Svaki sloj može imati različit broj neurona, a jedna neuronska mreža može sadržavati i više skrivenih slojeva, pa onda u tom slučaju govorimo o dubokim neuronskim mrežama što je područje dubokog učenja. Ulazni sloj služi za primanje ulaznih podataka, dok skriveni sloj služi za učenje i raspoznavanje karakteristika podataka na temelju kojih izlazni sloj predviđa rezultate. Svaki neuron u skrivenim i izlaznim slojevima se sastoji od težina koje se tijekom procesa učenja ažuriraju kako bi se uz pomoć njih mogle raspoznavati karakteristike podataka i raditi predviđanja. Dva su glavna koraka kod neuronskih mreža, a to su širenje unaprijed i širenje unatrag. Širenje unaprijed služi za predviđanje rezultata, dok širenje unatrag služi za učenje neuronske mreže, tj. ažuriranje težina [8].



Slika 4: Primjer neuronske mreže (Prema: [9])

3.1.3.1. Širenje unaprijed

Kako bi se pojednostavile i ubrzale matematičke operacije unutar neuronske mreže, ulazne podatke ćemo spojiti u jedan vektor X, a sve težine na pojedinom sloju u vektor W. To znači da nećemo trenirati samo jedan podatak istovremeno, već više njih. Kod širenja unaprijed izlazne vrijednosti neurona u svakom sloju se računaju pomoću jednadžbe:

$$y^{(i)} = f(W^{(i)^T} X^{(i-1)} + b^i)$$ [8].

Primijetimo da su ulazne vrijednosti X u sloj i zapravo izlazne vrijednosti neurona iz prethodnog sloja $i - 1$. Osim toga, vidimo da je jednadžba slična modelu linearne regresije. Razlika je u tome što ovdje primjenjujemo funkciju f koja će modelu pružati svojstvo nelinearnosti što će omogućiti veću preciznost predviđanja.

Postoji veliki niz različitih funkcija nelinearnosti, a neke od njih su: sigmoidna, ReLU (eng. *rectified linear unit*) i softmax. Izgled sigmoidne funkcije je u obliku slova S, dok njezine izlazne vrijednosti mogu biti između 0 i 1. Može biti korisna kod oboje skrivenih i izlaznih slojeva.

$$f(z) = \frac{1}{1+e^{-z}}\,.$$

ReLU je jedna od najčešće korištenih funkcija nelinearnosti kod skrivenih slojeva. Vrlo je jednostavna te izgledom podsjeća na hokejski štap. Ona sve negativne ulazne vrijednosti pretvara u nulu.

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

Softmax se često koristi kod izlaznog sloja u slučaju klasifikacije podataka. Njezin izlaz je jednak jednom vektoru koji se sastoji od distribucije vjerojatnosti svih klasa. Drugim riječima, ova funkcija određuje koja je vjerojatnost da izlazna vrijednost pripada pojedinoj klasi, što

znači da će element i biti vjerojatnost da izlazna vrijednost pripada i-toj klasi. Logično, ona klasa koja ima najveću vjerojatnost će biti uzeta u obzir kao rezultat predikcije. Vjerojatnost i-tog neurona odnosno klase se izračunava pomoću formule:

$$f(z) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$ [8].

3.1.3.2. Širenje unatrag

Tijekom treniranja neuronske mreže modificiramo težine na način da smanjimo pogrešku rezultata predviđanja u odnosu na stvarne podatke za treniranje. Nakon nekog vremena možemo očekivati da će istrenirana neuronska mreža biti dovoljno dobra za rješavanje naših problema. Kao i kod linearne regresije, cilj je minimalizirati srednju kvadratnu pogrešku uz pomoć modificiranja težina. One se na početku rada algoritma najčešće inicijaliziraju nasumično. Iterativni algoritam modificiranja težina s ciljem minimaliziranja srednje kvadratne pogreške zove se gradijentni spust. On se temelji na izračunu gradijenta, odnosno parcijalne derivacije funkcije pogreške po varijabli (težini) w. Težine se ažuriraju od izlaznog sloja prema ulaznom, tj. u obrnutom redoslijedu u odnosu na širenje unaprijed [8].

Neka je j sloj za kojeg već imamo izračunate derivacije funkcije greške, te i sloj za kojeg želimo izračunati derivacije, što je zapravo sloj $j-1$. Na početku rada ovog algoritma potrebno je izračunati derivaciju funkcije greške za izlazni sloj. To možemo izračunati prema jednadžbi:

$$E = \frac{1}{2}\sum_j (y_j - \hat{y}_j)^2 \Rightarrow \frac{\partial E}{\partial \hat{y}_j} = -(y_j - \hat{y}_j)\ .$$

S obzirom na to da već od prije imamo vrijednosti y_j i $\hat{y}_j$ što su zapravo stvarna izlazna vrijednost i predviđena izlazna vrijednost, sada možemo izračunati parcijalnu derivaciju funkcije greške po težini w na sloju i, odnosno formalnije $\frac{\partial E}{\partial w_{ij}}$. Nju ćemo izračunati prema jednadžbi:

$$\frac{\partial E}{\partial w_{ij}} = \hat{y}_i \hat{y}_j (1 - \hat{y}_j) \frac{\partial E}{\partial \hat{y}_j}\ .$$

Za kraj je potrebno ažurirati težine s novim vrijednostima uz pomoć jednadžbe:

$$\Delta w_{ij} = -\sum_k \alpha \hat{y}_i^{(k)} \hat{y}_j^{(k)} (1 - \hat{y}_j^{(k)}) \frac{\partial E^{(k)}}{\partial \hat{y}_j^{(k)}}$$ [8].

3.1.3.3. Prekomjerno prilagođavanje

Jedan od čestih problema neuronskih mreža je prekomjerno prilagođavanje (eng. *overfitting*) podacima za treniranje. Naime, zbog velike složenosti modela, moguće je da naš model ima visoku preciznost (malu grešku) nad podacima za treniranjem, no znatno manju

preciznost nad podacima za testiranje. U tom slučaju model nije dobar u generalizaciji, odnosno u predviđanju šireg spektra karakteristika kojeg domena našeg problema može poprimiti [8].

Postoje nekoliko tehnika s kojima možemo riješiti ovaj problem. Prva je regularizacija čiji je cilj kažnjavanje, tj. sprječavanje velikih težina. Nju implementiramo tako što modificiramo funkciju greške na sljedeći način:

$$Greška + \lambda f(\theta),$$

gdje je λ hiperparametar, a $f(\theta)$ funkcija regularizacije. Najčešća vrsta regularizacije je L2 regularizacija koja zahtjeva da se funkciji greške za svaku težinu dodaje izraz $\frac{1}{2}\lambda w^2$ [8].

Iduća tehnika je izostavljanje neurona (eng. *dropout*) koja posebice može biti korisna kod modela s velikim brojem neurona. Ova tehnika se implementira tako što se svakom neuronu pridodaje jedan hiperparametar p koji predstavlja vjerojatnost da pojedini neuron bude aktivan prilikom treniranja. To će prisiliti model da bude precizan i bez prisustva nekih informacija [8].

Osim navedenog moguće je i pojednostaviti model tako što smanjimo broj skrivenih slojeva ili neurona. Složeni modeli obično zahtijevaju veliku količinu podataka, pa je zbog toga još jedna tehnika za sprječavanje prekomjernog prilagođavanja i povećanje količine podataka. Povećanje broja podataka je moguće izvesti i bez nabavljanja novih, no to ne vrijedi za bilo koju vrstu podataka, već samo za slikovne podatke. Ova tehnika se može realizirati pomoću zrcaljenja, rotacije, skaliranja, mijenjanja svjetline slike i slično tome [10].

3.2. Nenadzirano učenje

Za razliku od nadziranog učenja gdje je cilj naučiti preslikavanje iz ulaznog u izlazni podatak, u nenadziranom učenju nemamo izlazne podatke, već samo imamo ulazne podatke uz pomoć kojih želimo pronaći pravilnosti. U ovom je području čest pojam procjena gustoće koji predstavlja pronalaženje uzoraka u prostoru ulaznih podataka koji se pojavljuju češće nego neki drugi uzorci. Jedna od najčešćih metoda za procjenu gustoće je klasterifikacija odnosno grupiranje ulaznih podataka [2].

3.2.1. Analiza glavnih komponenti

Analiza glavnih komponenti je popularan algoritam nepotpornog učenja koji uči reprezentaciju podataka. Drugim riječima, ona uči reprezentaciju koja ima manju dimenzionalnost nego što to ima originalan skup podataka. S obzirom na to da se prilikom smanjenja dimenzionalnosti gube informacije o podacima, ovaj algoritam smanjuje dimenziju

podataka tako što pokuša sačuvati što više informacija o podacima na način da se obrišu manje relevantne informacije [4].

Kriterij koji se koristi da se sačuva što više informacija je varijanca. Glavna komponenta w_1 je takva da nakon projekcije podataka na w_1 podaci budu najrasprostranjeniji, tj. da razlika između točaka bude najočitija. Ovaj problem se rješava tako što se pokuša maksimizirati varijanca

$$Var(z_1) = w_1^T \Sigma w_1,$$

gdje je Σ kovarijanca te mora vrijediti uvjet $w_1^T w_1 = 1$ [2].

Kako bismo odredili glavne komponente i uz što veću varijancu reprezentirali podatke u manjoj dimenziji, potrebno je napraviti sljedeće korake. Najprije je potrebno izračunati prosječnu vrijednost za svaku dimenziju u našem skupu podataka. Nakon toga možemo izračunati kovarijancu dviju varijabla X i Y uz pomoć formule:

$$Cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{x})(Y_i - \bar{y})$$ [11].

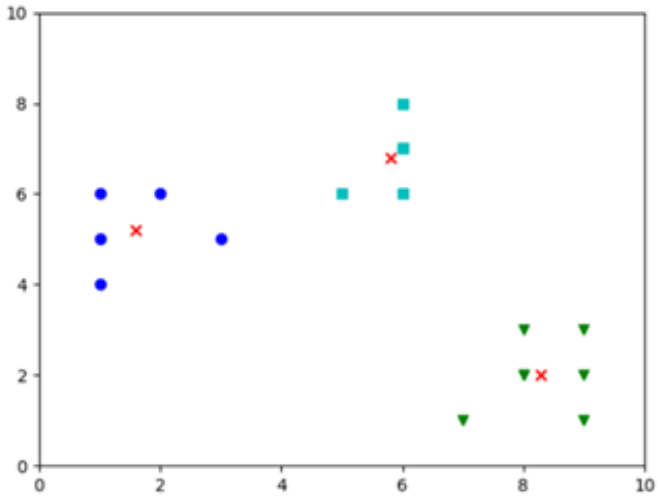
Ovu formulu moramo upotrijebiti za svaku kombinaciju dviju varijabli u našem skupu podataka i onda uz pomoć izračunatih vrijednosti konstruirati matricu kovarijanci A. Zatim je potrebno izračunati svojstvenu vrijednost i svojstveni vektor matrice A. Svojstveni vektori će služiti za formiranje smjera osi novog potprostora, dok će se uz pomoć svojstvenih vrijednosti odrediti koje svojstvene vektore moramo izbaciti kako bismo dobili traženu manju dimenziju. Izbacit ćemo one svojstvene vektore čije su svojstvene vrijednosti najmanje. Za kraj je potrebno matricu svojstvenih vektora pomnožiti s našim podacima kako bismo reprezentirali podatke u novom potprostoru [11].

3.2.2. k-sredina

Algoritam k-sredina razdvaja skup podataka u k različitih grupa (klastera) čiji su elementi , tj. podaci međusobno blizu. Ovaj algoritam svakom podatku pruža jedan vektor. Ako određeni podatak pripada klasteru i, i-ti element će iznositi 1, dok će ostali elementi iznositi 0. Algoritam radi tako što se inicijalizira k različitih centroida $\{\mu^{(1)}, \dots, \mu^{(k)}\}$ s različitim vrijednostima. Nakon toga se izmjenjuju dva različita koraka tako dugo dok ne dođemo do konvergencije [4].

U prvom koraku se svaki podatak dodjeljuje nekom klasteru i, gdje je i indeks najbližeg centroida $\mu^{(i)}$. U drugom koraku se svaki centroid $\mu^{(i)}$ ažurira s prosječnom vrijednošću svih podataka koji su pridruženi klasteru i [4].

Na sljedećoj slici imamo prikaz grupiranja podataka uz pomoć algoritma k-sredina. Podaci su dvodimenzionalni te su grupirani u tri različite grupe koje se razlikuju po boji i obliku reprezentacije podatka. Svaka grupa ima jedan centroid koji je označen s crvenim križićem.



Slika 5: Primjer k-sredina [12]

3.3. Podržano učenje

Podržano učenje se bavi problemom pronalaženja i odabira odgovarajućih akcija koje je potrebno poduzeti kako bi se maksimizirala nagrada. Za razliku od nadziranog i nenadziranog učenja, ovdje nemamo podatke već ih je potrebno saznati iz postupaka pokušaja i pogreške. U podržanom učenju najčešće imamo niz stanja i akcija koje su u interakciji s okolinom. Često trenutna akcija utječe na nagradu u trenutnom vremenu i u nadolazećem vremenu, tj. ona može utjecati na odluku daljnjih akcija. Glavno obilježje podržanog učenja je kompromis između istraživanja i eksploatacije. U istraživanju sustav isprobava nove akcije da utvrdi koliko su one efikasne, dok kod eksploatacije sustav koristi već ranije poznate akcije koje daju visoku nagradu [13].

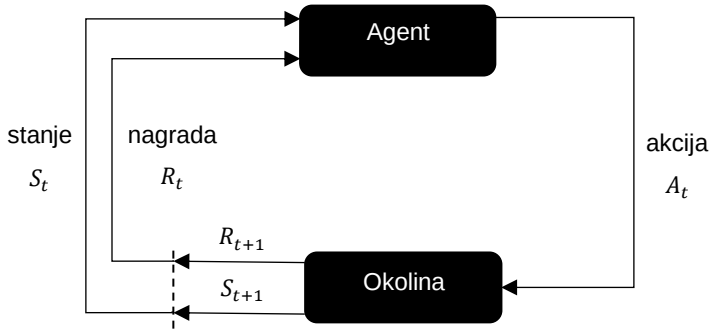
Podržano učenje je okvir opće namjene za kreiranje inteligentnih agenata. Ukoliko agentu pružimo okolinu i nagradu, on će naučiti djelovati s tom okolinom u cilju da maksimizira svoju nagradu. Ovaj princip učenja sliči onom kod živih bića. Koristi se u raznim područjima kao što su: autonomna vozila, robotika, računalne igre, optimizacija pozicioniranja oglasa, trgovanje dionicama [8]…

Na primjer kod autonomnih vozila okolina je promet, a cilj agenta (automobila) je da vozi po prometnoj traci kako bi na taj način maksimizirao svoju nagradu. Osim toga, mora izbjeći sudare jer oni donose negativnu nagradu odnosno kaznu.

3.3.1. Markovljevi procesi odlučivanja

Markovljevi procesi odlučivanja nude matematički okvir za modeliranje donošenja odluka u zadanom okruženju. Sastoji se od sljedećih elemenata:

- stanje – različita stanja koje agent ili okolina mogu poprimiti,
- akcija – akcije koje agent izvodi nad okruženjem odnosno okolinom,
- prijelaz stanja – nakon akcije agenta se mijenja okruženje,
- nagrada – ukoliko agent izvrši ispravnu akciju onda dobiva pozitivnu nagradu, u suprotnom dobije negativnu nagradu (kaznu) [8].



Slika 6: Osnovni model podržanog učenja (Prema: [14])

Prilikom izvršavanja akcije u okviru Markovljevih procesa odlučivanja, formira se jedna epizoda. Ona se sastoji od niza stanja, akcija i nagrada te se one izvršavaju tako dugo dok okruženje ne dostigne željeno završno stanje. Cilj je pronaći optimalnu politiku za agenta, gdje je politika način na koji agent djeluje s obzirom na njegovo trenutno stanje. Formalno se to zapisuje kao funkcija π koja odabire akciju a koju će agent izvršiti u stanju s. Na temelju politike se može konstruirati funkcija cilja Markovljevih procesa odlučivanja koja pronalazi politiku koja će maksimizirati očekivani budući prihod (eng. *future return*). Funkcija cilja se ovdje definira kao:

$$max_\pi \, E[R_0 + R_1 + \cdots + R_t|\pi],$$

gdje R predstavlja budući prihod svake epizode [8].

3.3.1.1. Budući prihod

Kao što je to već ranije navedeno, važno je uzeti u obzir ne samo trenutnu nagradu već i nagradu u budućnosti, odnosno izvršiti onu akciju koja će imati pozitivan efekt u budućnosti. Zbog toga želimo naše agente optimizirati s budućim prihodom, pa ćemo svaku nagradu izračunati tako što zbrojimo trenutnu nagradu i sve buduće očekivane nagrade, odnosno formalnije:

$$R_t = \sum_{k=0}^{T} r_{t+k}.$$

Ovo je naivni postupak izračunavanja budućeg prihoda koji nije najbolji jer može naići na probleme. Primjerice, ukoliko agent može imati beskonačno mnogo koraka onda ovaj algoritam neće nikada izračunati svoje nagrade zbog toga što ne postoji posljednji korak sa kojim bi algoritam prestao s računanjem nagrade. Osim toga, buduće nagrade bi trebale imati nešto manju vrijednost kako bi agent mogao brže učiti [8].

Zbog navedenih problema se uvodi strategija pod nazivom diskontirani budući prihod. Takav prihod se implementira tako što se skalira nagrada trenutnog stanja s diskontnim faktorom γ. Diskontni faktor se potencira s trenutnim brojem koraka t kako bi se ga kaznilo za uzimanje prevelikog broja koraka. Njegove vrijednosti mogu biti između 0 i 1, no najčešće se uzimaju vrijednosti između 0.97 i 0.99. Izračunavamo ga prema formuli:

$$R_t = \sum_{k=0}^{T} \gamma^t r_{t+k+1}$$ [8].

4. Otvorene biblioteke za strojno učenje

Biblioteke su kolekcija resursa koje koriste računalni programi u svrhu razvoja programa. Resursi mogu biti: konfiguracije, dokumentacija, podaci za pomoć, programski kod i slično. Osim toga, biblioteka je kolekcija implementacije ponašanja koja je napisana u nekom programskom jeziku s dobro definiranim sučeljem za pristup odnosno prizivanje ponašanja. Njezin glavni element je unaprijed napisani programski kod koji je organiziran tako da se može koristiti na više različitih programa ili potprograma koji ne ovise jedan o drugom. Još jedno važno svojstvo je da za korištenje neke biblioteke, korisnik ne mora znati interne detalje biblioteke već samo njezino sučelje za pristup. Kada program pozove neku biblioteku on dobiva ponašanje implementacije unutar nje bez potrebe zasebne implementacije za ostvarenje tog ponašanja, pa iz toga onda proizlazi njezina svrha korištenja a to je svojstvo ponovne iskoristivosti [15].

Svrha korištenja biblioteka u strojnom učenju je olakšati implementiranje algoritama na način da u većini slučajeva ne moramo implementirati logiku rada algoritma već je dovoljno pozvati određene funkcije i proslijediti joj ispravne parametre i hiperparametre. Ovaj rad se bavi isključivo s otvorenim bibliotekama, a to su one biblioteke koje su besplatne i imaju javno dostupan izvorni kod.

Primjerice, ukoliko želimo implementirati algoritam k-sredina u Python biblioteci scikit-learn, dovoljno je inicijalizirati objekt klase KMeans sa željenim brojem klastera i nakon toga pozvati njegovu metodu fit(x), gdje je x polje koje sadrži ulazne podatke. Za kraj, ako želimo predvidjeti kojem klasteru neki podatak x pripada, onda je samo potrebno pozvati metodu predict(x). U nastavku će se navesti neke od najpopularnijih otvorenih biblioteka za strojno učenje [16].

4.1. Scikit-learn

Scikit-learn je vrlo popularna biblioteka za strojno učenje koja omogućuje niz raznih algoritama iz područja klasifikacije, regresije, klasterifikacije, smanjenja dimenzionalnosti, konfiguracije modela i preprocesiranja. Jedna od njezinih mana je što nju podržava samo programski jezik Python, za razliku od nekih drugih biblioteka koje podržavaju više jezika. Izgrađena je na temelju poznatih Python biblioteka Numpy, SciPy i matplotlib koji služe za matematičke izračune i grafičke prikaze grafova. Koriste ga brojne tvrtke kao što su: J.P.Morgan, Spotify, Change.org, Booking.com, Inria, betaworks [16]…

Slika 7: Scikit-learn logo [17]

4.2. TensorFlow

TensorFlow je najpopularnija biblioteka za strojno učenje koju je razvio tim iz Google Brain-a. Ona omogućuje kreiranje grafova podatkovnih tokova odnosno strukture koja opisuje kako se podaci kreću kroz graf. Svaki čvor u grafu predstavlja neku matematičku operaciju, dok svaki rub predstavlja višedimenzionalno polje ili tenzor odakle mu i dolazi ime. Matematičke operacije se izvršavaju u C++-u kako bi se mogle postići visoke performanse, međutim Python se najčešće koristi za pristup API-ju zbog toga što on omogućuje brži i lakši razvoj aplikacija. Za razliku od scikit-learn-a kojeg podržava samo Python, TensorFlow je podržan od strane više jezika kao što su Python, C, C++, Go, Java, JavaScript i Swift. Njegove aplikacije se mogu pokrenuti na mnogo različitih platformi, na primjer na računalu, cloudu, iOS-u i Androidu-u. Ono što je također vrlo važno za modele s velikim brojem podataka je i mogućnost treniranja ne samo na centralnom procesoru (CPU), već i na grafičkom procesoru (GPU) koji često omogućuje brže treniranje. Osim toga, kombinirajući TensorFlow s Google-ovim cloud servisom, moguća je upotreba TensorFlow procesnih jedinica (TPU) koje omogućuju još brže treniranje modela. Važno je i reći da TensorFlow koristi Keras-ov API za dodatno olakšavanje rada s neuronskim mrežama [18].

Slika 8: TensorFlow logo [19]

4.3. Keras

Keras je biblioteka za rad s neuronskim mrežama koja je napisana u Pythonu. Kao što je prije napisano, ova se biblioteka može koristiti u TensorFlow-u, ali isto tako i u Microsoft Cognitive Toolkit-u, Theano-u te u programskom jeziku R. Njezina glavna prednost je što je jednostavna za korištenje pa zbog toga ubrzava razvoj modela neuronskih mreža. Ona pokriva sve važne koncepte potrebne pri izradi jednostavnih ili složenijih modela kao što su primjerice slojevi, aktivacijske funkcije, optimizatori te razne normalizacije. Osim standardnih neuronskih mreža, ona omogućuje olakšani razvoj konvolucijskih i rekurentnih neuronskih mreža [20].

Slika 9: Keras logo [21]

4.4. ML.NET

ML.NET je Microsoftova biblioteka za strojno učenje napravljena za programske jezike C# i F#. Ukoliko se koristi zajedno s Python modulom NimbusML, tada je također moguća i upotreba modela izrađenih u Pythonu. Kako je ovo još prilično nova biblioteka, ona podržava manji broj algoritama u odnosnu na scikit-learn ili TensorFlow. Prema tome, trenutno se može koristiti za rješavanje problema kao što su: analiza sentimenata, predviđanje cijena, otkrivanje kvarova, prognoziranje prodaje, preporuka proizvoda, segmentacija klijenata i detekcija anomalije, dok se raspoznavanje objekata na slici i klasifikacija slika može koristiti uz pomoć prethodno istreniranih modela. Prednost ML.NET-a je što nudi tzv. Model Builder što je jednostavan alat uz pomoć kojeg je lako moguće izraditi prilagođene modele strojnog učenja. On koristi metodu pod nazivom automatizirano strojno učenje koje automatski odabire model s najboljim performansama za zadani problem iz domene strojnog učenja [22].

Slika 10: ML.NET logo [23]

4.5. PyTorch

PyTorch je biblioteka za strojno učenje koja se temelji na biblioteci Torch. Razvili su je Facebookovi znanstvenici za umjetnu inteligenciju. Preferirani jezik za upotrebu PyTorcha je Python, no postoji sučelje i za C++ i Javu (za Android upotrebu). PyTorch nudi dvije važne značajke koje omogućavaju lakšu implementaciju algoritama strojnog učenja s visokim performansama. Prva značajka je tenzorsko računanje uz mogućnost upotrebe grafičkog procesora za bolje performanse, a druga značajka je da su duboke neuronske mreže izgrađene na metodi automatske diferencijacije. Automatska diferencijacija služi za snimanje izvedenih operacija kako bi se kasnije prilikom izračuna gradijenata kod širenja unatrag one mogle ponoviti te tako ubrzati vrijeme treniranja neuronske mreže. U ožujku 2018. godine PyTorch se spojio s popularnom bibliotekom Caffe2 te tako proširio svoje mogućnosti. Slično kao i TensorFlow, PyTorch nudi klasu pod imenom Tensor koja služi za obradu višedimenzionalnih polja. Osim nje, PyTorch nudi module za lakšu izgradnju neuronskih mreža, zatim za automatsku diferencijaciju te za optimizacijske algoritme [24].

Slika 11: PyTorch logo [25]

4.6. Apache MXNet

Apache MXNet je biblioteka za duboko učenje razvijena od tvrtke Apache. Kao i većina ostalih biblioteka, najbolje ju podržava Python, a također postoji podrška i za programske

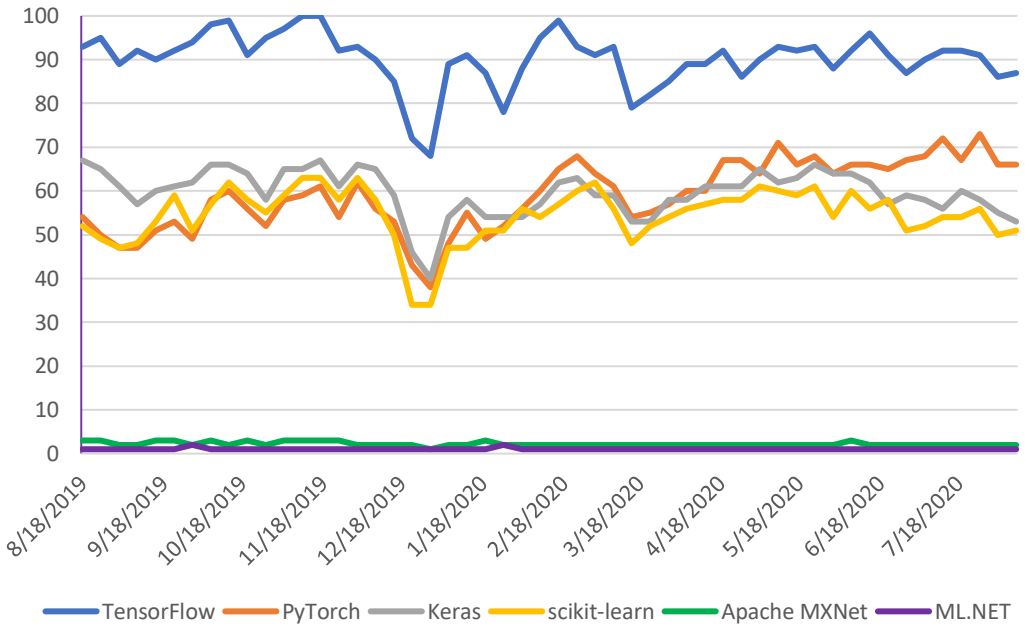
jezike Java, Scala, Julia, Clojure, C++, R i Perl. Ova biblioteka je fleksibilna i podržava simboličko programiranje jednostavnim pozivanjem funkcije hybridize. Simboličko izvršavanje omogućuje brži i optimiziraniji rad te mogućnost spremanja i upotrebe modela u nekom drugom jeziku kojeg podržava MXNet. Osim toga, MXNet pruža efikasno distribuirano treniranje modela s više grafičkih procesora s gotovo linearnim skaliranjem [26].

Slika 12: Apache MXNet logo [27]

4.7. Usporedba otvorenih biblioteka za strojno učenje

Biblioteke za strojno učenje s kojima smo se ranije upoznali se razlikuju po nekoliko faktora kao što su funkcionalnosti, popularnost, performanse i podržani programski jezici. Biblioteka scikit-learn ima veliku širinu pokrivenih algoritama i vrlo je jednostavna za korištenje, no najčešće se ne koristi za implementaciju složenijih algoritama kao što su duboke neuronske mreže. Osim toga, za razliku od recimo TensorFlowa i PyTorcha, scikit-learn ne podržava upotrebu GPU-a što zapravo i nije od velike važnosti zbog toga što se koristi samo za jednostavnije algoritme koji ne zahtijevaju jake računalne performanse. Keras i MXNet se isključivo koriste za rad s neuronskim mrežama, s time da se Keras razlikuje po tome što je on uključen u jezik R i TensorFlow dok se MXNet koristi zasebno. ML.NET je još prilično nova biblioteka pa zbog toga podržava nešto manje algoritama u odnosu na neke druge biblioteke, ali zato omogućuje implementaciju algoritama uz pomoć Model Buildera.

Popularnost pojedine biblioteke je moguće odrediti na više načina. Vjerojatno jedan od najvjerodostojnijih i najpouzdanijih načina za provjeriti ovu statistiku je uz pomoć Google trendova koji daju rezultate o kvantitativnoj razini web upita na Google pretraživaču koji su skalirani od 0 do 100. Rezultate za prethodnu godinu dana možemo vidjeti na slici 13. Vidimo da TensorFlow čitavo vrijeme dominira s popularnošću, a slijede ga PyTorch, Keras i scikit-learn. Apache MXNet i ML.NET su znatno manje popularni i teško je za očekivati da će se situacija u bližoj budućnosti promijeniti s obzirom na to da su ostale navedene biblioteke postale standard u strojnom učenju.



Slika 13: Popularnost biblioteka za strojno učenje (Prema: [28])

Svaka biblioteka za strojno učenje ima za cilj podržati određenu skupinu programskih jezika. Na primjer, scikit-learn podržava samo Python koji je najčešće korišten jezik za strojno učenje kojeg podržava većina biblioteka. Za razliku od scikit-learna-a, TensorFlow i Apache MXNet podržavaju veliku skupinu programskih jezika što omogućuje korištenje strojnog učenja na više različitih platforma kao što su web i pametni uređaji. ML.NET se izdvaja od ostalih biblioteka zbog toga što je njihova ciljna skupina programskih jezika iz Microsoftove .NET domene (C# i F#), pa je zbog toga teško za očekivati da će u budućnosti podržavati jezike koji nisu u toj domeni. Na sljedećoj tablici vidimo koje programske jezike podržava pojedina biblioteka za strojno učenje.

Tablica 1: Podržani programski jezici za pojedinu biblioteku strojnog učenja

Biblioteka za strojno učenje	**Podržani programski jezici**	**Ukupno**
scikit-learn	Python	1
TensorFlow	Python, C, C++, Go, Java, JavaScript, Swift	7
Keras	Python, R	2
ML.NET	C#, F#	2
PyTorch	Python, C++, Java	3
Apache MXNet	Python, Java, Scala, Julia, Clojure, C++, R, Perl	8

5. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže su vrste neuronskih mreža koje se često koriste za procesiranje slikovnih podataka i podataka vremenskih serija, pa će se zbog toga uz njihovu pomoć izraditi model za kolorizaciju crno-bijelih slika. Njihov naziv proizlazi iz matematičke operacije konvolucije koja se koristi pri radu mreže. Ovaj tip neuronske mreže se ne razlikuje mnogo od klasičnih neuronskih mreža koje su objašnjene u poglavlju 3.1.3. Glavna razlika je u tome što konvolucijske neuronske mreže umjesto matričnih množenja izvršavaju konvolucije. Operator konvolucije se označava s $*$, a računa se uz pomoć neodređenog integrala:

$$s(t) = \int x(a)w(t-a)da,$$

gdje je $x(a)$ funkcija ulaza, dok $w(t-a)$ označava filter čije se vrijednosti tijekom vremena ažuriraju, odnosno uče. Rezultat konvolucije je mapa značajki [4].

Filter je matrica manjih dimenzija (npr. 3x3) uz pomoć koje se mogu raspoznati značajke u podacima kod konvolucijskih neuronskih mreža. U praksi se u jednom sloju konvolucijske neuronske mreže koristi više filtera čime se omogućuje raspoznavanje veće količine značajki. Broj filtera utječe na dimenziju mape značajki, pa će se primjenom veće količine filtera u nekom sloju bitno utjecati na brzinu treniranja modela i njegovu kompleksnost. Kako bi se smanjila dimenzija mape značajki, moguće je povećati skok (eng. *stride*) koji predstavlja broj piksela koji se pomiču prilikom računanja konvolucije. Čest je slučaj da filter ne stane na sliku, pa je zbog toga potrebno podesiti punjenje slike (eng. *padding*). Postoje dvije opcije punjenja. Prva opcija je da se nedostajući pikseli na slici popune s nulama, a druga opcija je izostavljanje onog dijela slike koji ne stane na filter [29].

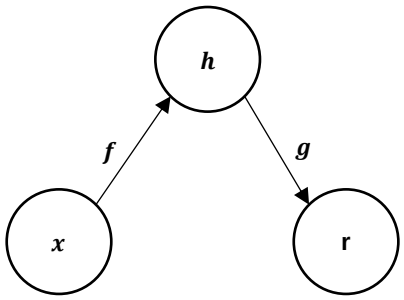
Izlazna dimenzija mape značajki ovisi o ulaznoj dimenziji d_{ulaz}, broju (količini) filtera k i njihovim dimenzijama e, broju skokova s te broju piksela korištenih pri punjenju podatka p. Dubina mape značajki je jednaka broju filtera k, dok se preostale dvije dimenzije (širina i visina) računaju po formuli:

$$d_{izlaz} = \frac{d_{ulaz} - e + 2p}{s} + 1 \text{ [8].}$$

Konvolucije se računaju u konvolucijskom sloju, međutim osim tog sloja postoji i sloj sažimanja i potpuno povezani sloj koji dodatno proširuju mogućnosti konvolucijske neuronske mreže. Sloj sažimanja služi za smanjenje broja parametara tako da se smanji dimenzija mape značajki. Koristan je zbog toga što on prilikom smanjivanja broja parametara može zadržati važne značajke o podacima, pa se zbog toga ovaj sloj često koristi kad postoji problem s velikim brojem parametara. Potpuno povezni sloj služi za pretvorbu podataka u jedan vektor koji se sastoji od svih prethodno naučenih značajki [29].

5.1. Autoenkoder

Autoenkoder je algoritam koji uči reprezentaciju podataka uz pomoć funkcije kodera $h = f(x)$ i funkcije dekodera $r = g(x)$. Koder pretvara ulazni podatak u neku reprezentaciju, dok dekoder pretvara novu reprezentaciju natrag u originalan oblik. Uspješni autoenkoder nije onaj koji uspije savršeno kopirati ulazni podatak u izlazni, odnosno kopirati sve značajke ulaznog podataka, već onaj koji uspije naučiti samo važne i korisne značajke o ulaznom podatku kako bi se uz pomoć njih kreirao izlazni podatak. Prema tome, oni mogu biti vrlo korisni kod rješavanja problema smanjenja dimenzionalnosti podataka. Na slici ispod se nalazi osnovna arhitektura autoenkodera [4].



Slika 14: Arhitektura autoenkodera (Prema: [4])

Ovisno o problemu, autoenkodere je moguće implementirati na različite načine. S obzirom na to da je cilj aplikacije kolorizirati crno-bijele slike, implementirani autoenkoder će sadržavati samo konvolucijske slojeve. Ranije je spomenuto da se autoenkoderi često koriste za smanjenje dimenzionalnosti podataka, međutim, u problemu kolorizacije je potrebno uraditi suprotno, tj. povećati dimenziju slike kako bi ona mogla sadržavati veći spektar boja. To će se realizirati tako što će crno-bijela slika biti ulazni podatak čije će se značajke pokušati naučiti u koderu, a u dekoderu će se one iskoristiti za rekonstruiranje slike u koloriziranoj verziji. Boje značajki će se naučiti uz pomoć izlaznih podataka, konkretnije slike u AB formatu koji će se detaljnije objasniti u poglavlju 6.1.

6. Aplikacija za kolorizaciju crno-bijelih slika

Cilj ove aplikacije je omogućiti njezinom korisniku odabir željene crno-bijele slike te ju na jednostavan pritisak gumb kolorizirati. Nakon što aplikacija uz pomoć strojnog učenja, konkretnije autoenkodera kolorizira odabranu sliku, korisniku se prikazuje predviđena kolorizirana verzija slike koju je moguće spremiti.

Slika 15: Aplikacija za kolorizaciju crno-bijelih slika

6.1. Skup podataka i korištene tehnologije

Skup podataka koji se je koristio pri treniranju i testiranju modela za kolorizaciju crno-bijelih slika je COCO (eng. *Common Objects in Context*) koji se sastoji od 123 000 slika raznovrsnih kategorija. Slike su različitih rezolucija (dimenzija), pa ih je bilo potrebno prilikom njihovog učitavanja obraditi kako bi sve slike imale rezoluciju 256x256 koja je kompatibilna s modelom [30].

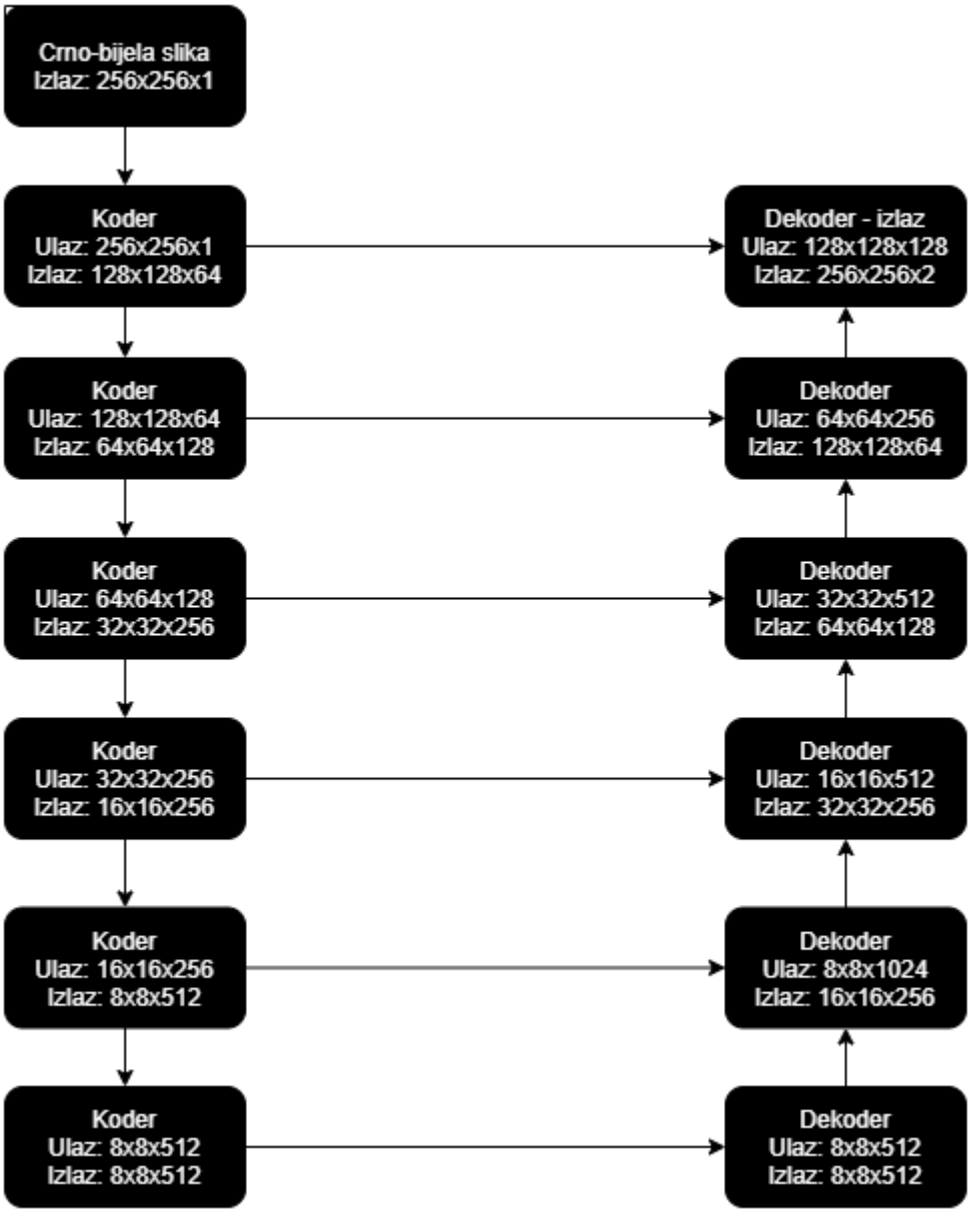
Osim postavljanja istih rezolucija, slike je potrebno pretvoriti u LAB prostor boja. Razlog tome je što prva dimenzija (L) ovog prostora predstavlja svjetlinu čije su vrijednosti izražene od 0 do 100, gdje 0 označava potpuno crnu boju, a 100 bijelu boju. Prema tome, crno-bijela slika se može prikazati uz pomoć prve dimenzije, a preostale dvije dimenzije su izlazni podatak koji se predviđa. Vrijednosti druge i treće dimenzije se najčešće kreću od -128 do 127. Druga dimenzija (A) predstavlja zelenu-crvenu komponentu, gdje negativne vrijednosti označavaju zelenu boju, a pozitivne crvenu boju. Slično, treća dimenzija (B) predstavlja plavo-žutu

komponentu, gdje negativne vrijednosti označavaju plavu boju, a pozitivne žutu boju. Prednost korištenja ovog prostora boja je da se ne moraju predviđati tri dimenzije kao što bi to slučaj bio s RGB slikama, već je dovoljno da se predvide dvije dimenzije (A i B). Rezultat spajanja prve dimenzije s preostalim predviđenim dimenzijama je kolorizirana slika u LAB formatu koju je lako moguće pretvoriti u RGB format [31].

Čitava aplikacija je razvijena u programskome jeziku Python zbog toga što je taj jezik jednostavan i najbolje podržava biblioteke za strojno učenje. Otvorene biblioteke za strojno učenje koje su se koristile pri razvoju aplikacije su Keras [32] i Tensorflow [33]. S obzirom na to da model ima mnogo parametara te skup podataka sadrži mnogo slika, za treniranje modela su se koristili GPU resursi koji su značajno ubrzali proces treniranja. Prema tome, koristio se Google Colaboratory [34] koji nudi besplatni pristup GPU i TPU resursima na određeno vrijeme.

6.2. Model autoenkodera

Ulazni podaci modela su crno-bijele slike dimenzija 256x256x1, dok su izlazni podaci slike pretvorene u AB format dimenzija 256x256x2. Kako bismo ubrzali treniranje, na početku skoro svakog sloja se smanjuje širina i visina mape značajki koja je dobivena na prethodnom sloju. U petom poglavlju je objašnjeno da je smanjenje moguće izvesti uz pomoć povećanja broja skokova ili korištenjem sloja sažimanja. Kod problema kolorizacije se bolji rezultati postižu upotrebom skokova, pa se zbog toga na većini konvolucijskih slojeva upotrebljava skok od dva piksela čime se duplo smanjuje širina i visina mape značajki. Svaka funkcija kodera koristi konvolucijske slojeve, dok funkcije dekodera koriste slojeve transponirane konvolucije koje služe za vraćanje slike u izvorni oblik, odnosno za rekonstruiranje slike. Zadnji transponirani konvolucijski sloj mora obavezno imati dva filtera kako bi se na taj čin dobio traženi format slike. Model je simetričan što znači da ima jednak broj slojeva za kodiranje i dekodiranje, konkretnije 6 slojeva za kodiranje i 6 slojeva za dekodiranje. Svi slojevi izuzev zadnjeg koriste ReLU funkciju nelinearnosti, a zadnji sloj koristi funkciju tangens hiperbolni kako bi raspon izlaznih vrijednosti bio između -1 i 1. Također, svi slojevi osim zadnjeg spajaju svoj ulaz s pripadajućom izlaznom vrijednošću nasuprotnog sloja. Time model lakše zadržava komponente originalne slike [35].



Slika 16: Model autoenkodera za kolorizaciju crno-bijelih slika

6.3. Opis implementacije aplikacije

Implementaciju aplikacije je moguće podijeliti na tri dijela. Prvi dio predstavlja pripremu i procesiranje podataka za treniranje modela u što je uključeno kreiranje generatora koji će vraćati podatke u grupama odnosno serijama (eng. *batches*) prilikom treniranja modela te podjelu podataka na podatke za treniranje i testiranje. Drugi dio predstavlja izradu modela i njegovo treniranje, a treći dio je zaslužan za grafičko sučelje i predviđanje kolorizacije odabrane crno-bijele slike te njezin prikaz na ekran. Ova tri dijela su organizirana u četiri zasebne Python datoteke: data_process.py, train_model.py, predict.py i GUI.py. U nastavku nije objašnjen čitav programski kod, već samo dijelovi koji se vežu uz korištenje biblioteka za strojno učenje. Čitav programski kod je javno dostupan na: https://github.com/pcrncec/Zavrsni_rad

6.3.1. Priprema podataka

Najprije je potrebno uvesti biblioteke koje će omogućiti lakše implementiranje funkcionalnosti za pripremu i procesiranje podataka. Od biblioteka za strojno učenje, uvozi se Keras-ova klasa ImageDataGenerator koja će poslužiti za učitavanje slika i njihovu normalizaciju. Osim toga, ova klasa omogućuje i umjetno povećanje podataka koje je već objašnjeno u poglavlju 3.1.3.3, no kako korišteni skup podataka sadrži mnogo slika ta funkcionalnost klase neće biti korištena. Slike se normaliziraju na način da se njihove vrijednosti podjele s 255 zbog toga što vrijednost svakog piksela u RGB formatu je od 0 do 255 za svaki sloj boje. Nakon kreiranja objekta klase ImageDataGenerator potrebno je pozvati metodu flow_from_directory() uz pomoć koje se na vrlo jednostavan način mogu generirati serije podataka iz željenog direktorija. Ovdje je bitno kao argument class_mode postaviti na None zbog toga što se korišteni skup podataka ne dijeli na klase. Za kraj kako bi se kreirao generator za treniranje autoenkodera potrebno je slike pretvoriti u LAB format te onda na temelju njega stvoriti ulazne i izlazne podatke.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from skimage.color import rgb2lab

def generator(batch_size, images_path):
    train_datagen = ImageDataGenerator(rescale=1./255)
    train_generator = train_datagen.flow_from_directory(
        images_path,
        class_mode=None,
        batch_size=batch_size
    )
```

```
    for batch in train_generator:
        lab_batch = rgb2lab(batch)
        x_batch = lab_batch[:, :, :, 0]
        y_batch = lab_batch[:, :, :, 1:] / 128
        yield x_batch, y_batch
```

6.3.2. Kreiranje i treniranje modela

Programski kod je ovdje podijeljen na dvije funkcije: create_model() i train(), gdje prva služi za kreiranje modela, a druga za treniranje tog modela. Kod izrade modela se koriste Keras-ovi slojevi Input, Conv2D, Conv2DTranspose i concatenate. S Inputom se definiraju dimenzije ulaznih slika. Uz pomoć Conv2D i Conv2DTranspose se definiraju konvolucijski i transponirani konvolucijski slojevi. Kod ovih slojeva je vrlo važno znati definirati parametre s obzirom na to da uz pomoć njih definiramo arhitekturu modela. Prvi parametar služi za definiranje broja filtera, dok drugi za dimenzije filtera koje u svakom sloju iznose (3, 3). Nakon toga se definira parametar activation koji označava funkciju nelinearnosti (prijenosa). Zatim se definira parametar padding koji je u svakom sloju 'same', što predstavlja punjenje slike nulama na način koji je objašnjen u petom poglavlju. Zadnji parametar koji je potrebno definirati je strides odnosno skok piksela prilikom računanja konvolucije. S obzirom na to da sve veze u modelu nisu sekvencijalne, već postoje veze između nasuprotnih slojeva, na kraju definiranja sloja je potrebno pozvati prethodni sloj na koji se trenutni nadovezuje. Sloj concatenate nam služi za spajanje dva sloja u jedan po zadnjoj dimenziji, a uvjet za tu realizaciju je da prethodne dvije dimenzije budu međusobno jednake. Nakon instanciranja objekta klase Model, poziva se metoda summary() koja nije obavezna, ali je korisna zbog toga što opisuje strukturu modela. U nastavku se nalazi implementirani programski kod za kreiranje modela.

```
from tensorflow.keras import Model
from tensorflow.keras.layers import Input, Conv2D, Conv2DTranspose,
concatenate


def create_model():
    input = Input(shape=(256, 256, 1,))
    encoder1 = Conv2D(64, (3, 3), activation='relu', padding='same',
strides=2)(input)
    encoder2 = Conv2D(128, (3, 3), activation='relu', padding='same',
strides=2)(encoder1)
    encoder3 = Conv2D(256, (3, 3), activation='relu', padding='same',
strides=2)(encoder2)
    encoder4 = Conv2D(256, (3, 3), activation='relu', padding='same',
strides=2)(encoder3)
```

```
    encoder5 = Conv2D(512, (3, 3), activation='relu', padding='same',
strides=2)(encoder4)


    encoder_output = Conv2D(512, (3, 3), activation='relu',
padding='same')(encoder5)


    decoder1 = Conv2DTranspose(512, (3, 3), activation='relu',
padding='same')(encoder_output)
    merge_decoder1 = concatenate([encoder5, decoder1])
    decoder2 = Conv2DTranspose(256, (3, 3), activation='relu',
padding='same', strides=2)(merge_decoder1)
    merge_decoder2 = concatenate([encoder4, decoder2])
    decoder3 = Conv2DTranspose(256, (3, 3), activation='relu',
padding='same', strides=2)(merge_decoder2)
    merge_decoder3 = concatenate([encoder3, decoder3])
    decoder4 = Conv2DTranspose(128, (3, 3), activation='relu',
padding='same', strides=2)(merge_decoder3)
    merge_decoder4 = concatenate([encoder2, decoder4])
    decoder5 = Conv2DTranspose(64, (3, 3), activation='relu',
padding='same', strides=2)(merge_decoder4)
    merge_decoder5 = concatenate([encoder1, decoder5])
    output = Conv2DTranspose(2, (3, 3), activation='tanh', padding='same',
strides=2)(merge_decoder5)
    model = Model(inputs=input, outputs=output)
    model.summary()
    return model
```

Funkcija train prima parametre o broju epoha za treniranje (nb_epochs), veličini serije (batch_size), stopi učenja modela (learning_rate), putanji na kojoj će se spremiti istrenirani model (save_path) te bool varijablu koja odlučuje da li je potrebno podijeliti podatke ili ne. Na početku je potrebno učitati kreirani model, podesiti Adam optimizator te uz pomoć metode compile() podesiti funkciju greške i metriku po kojoj će se pratiti uspješnost treniranja. Za funkciju greške je odabrana srednja kvadratna pogreška, dok se za metriku koristi preciznost modela.

```
from tensorflow.keras.optimizers import Adam
def train(nb_epochs, batch_size, learning_rate, save_path=os.getcwd(),
split_data=True):
    model = create_model()
    adam = Adam(lr=learning_rate)
    model.compile(optimizer=adam, loss='mse', metrics=['acc'])
```

Nakon podjele podataka na treniranje i testiranje kreiraju se njihovi generatori te je nakon toga model spreman za treniranje. Treniranje modela je moguće započeti pozivanjem metode fit()

kojoj je potrebno proslijediti parametre o generatorima, veličini serije podataka, broju koraka po epohi za treniranje i testiranje te broju epoha. Nakon što model završi s treniranjem, potrebno ga je spremiti za kasniju upotrebu uz pomoć metode save().

```
model.fit(train_generator,
batch_size=batch_size,steps_per_epoch=train_steps, epochs=nb_epochs,
validation_data=validation_generator, validation_steps=val_steps)
save_path = os.path.join(save_path, 'model.h5')
model.save(save_path)
```

6.3.3. Predviđanje rezultata

Kako bi bilo moguće predvidjeti rezultate kolorizacije crno-bijelih slika, potrebno je prvo učitati model uz pomoć metode load_model(). Nakon toga je potrebno učitati sliku, pretvoriti ju u polje, normalizirati ju, pretvoriti u LAB format te uzeti njezin prvi sloj pošto taj sloj predstavlja crno-bijelu sliku. Zatim je potrebno proširiti dimenziju slike kako bi bila kompatibilna s modelom te konačno pozvati modelovu metodu predict() i proslijediti joj kao parametar novonastalu crno-bijelu sliku. Rezultat metode je slika u AB formatu koju je potrebno spojiti s crno-bijelom slikom kako bi se dobio puni LAB format koji će predstavljati koloriziranu sliku. Za kraj je potrebno pretvoriti sliku iz LAB formata u RGB format kako bi se slika mogla na lakši način prikazati na sučelju.

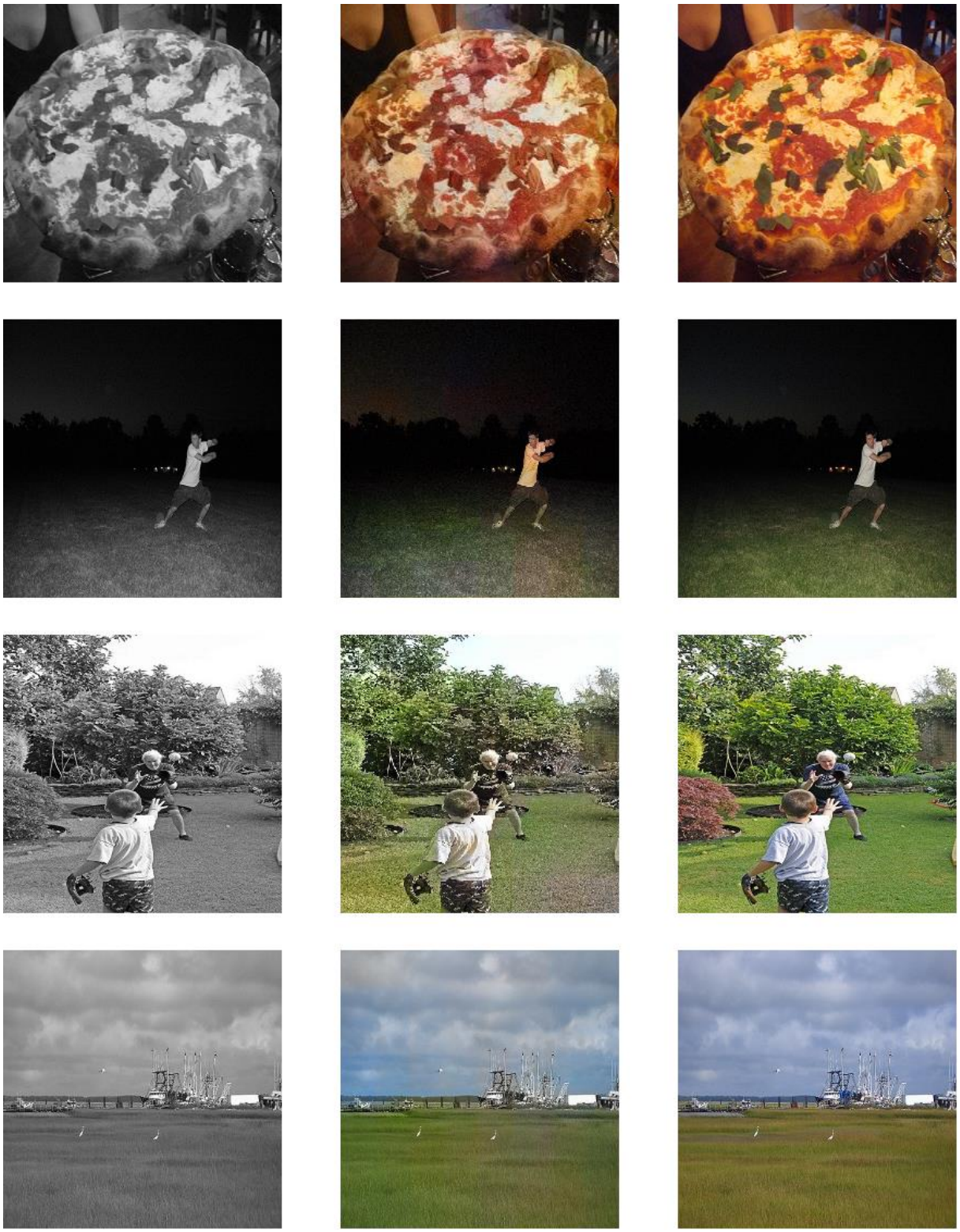
```
def predict_rgb_image(image_path):
    model = load_model('model.h5')
    loaded_img = load_img(image_path, target_size=(256, 256))
    img_array = img_to_array(loaded_img)
    img_array = 1.0/255 * img_array
    lab_img = rgb2lab(img_array)
    lab_img_bw = lab_img[:, :, 0]
    lab_img_bw = np.expand_dims(lab_img_bw, axis=0)
    colorized_img = model.predict(lab_img_bw)
    colorized_img = (128 * colorized_img)[0]
    merged_lab_img = np.zeros((256, 256, 3))
    merged_lab_img[:, :, 0] = lab_img_bw
    merged_lab_img[:, :, 1:] = colorized_img
    rgb_img = lab2rgb(merged_lab_img)
    return array_to_img(rgb_img)
```

6.4. Rezultati i evaluacija modela

Na slici 18 se nalaze rezultati modela prikazani kroz četiri primjerka slika na kojima model nije bio treniran. Očito je da je model prepoznao što je na slici te je na temelju toga kolorizirao sliku. Gotovo je nemoguće za očekivati da bi bilo koji model mogao uspješno kolorizirati svaki piksel slike, već samo one dijelove koji se lakše prepoznaju. Važno je za napomenuti da su na toj slici prikazani rezultati koje je model poprilično uspješno kolorizirao, međutim na mnogo slika on ne uspijeva prepoznati njihov kontekst te zbog toga ih ne zna kolorizirati. Takvi neuspješni rezultati su prikazani na slici 17. Kroz proučavanje rezultata kolorizacije moglo se zaključiti da je model uspješniji u predviđanju boja elemenata kao što su nebo, more, zelene površine i ljudi koji se često pojavljuju na slikama te ih je lako za prepoznati. Lošiji je u predviđanju elemenata čija je distribucija manja te onih elemenata koji se na slici pojavljuju na neprepoznatljiv ili neuobičajen način. Rezultati bi vrlo vjerojatno bili bolji da se model trenirao na punom većem skupu podataka, npr. na nekoliko milijuna slika, te da je on bio složeniji što bi mu omogućilo prepoznavanje većeg broja objekata. Međutim, treba uzeti u obzir da bi takvo treniranje trajalo tjednima ili čak mjesecima.

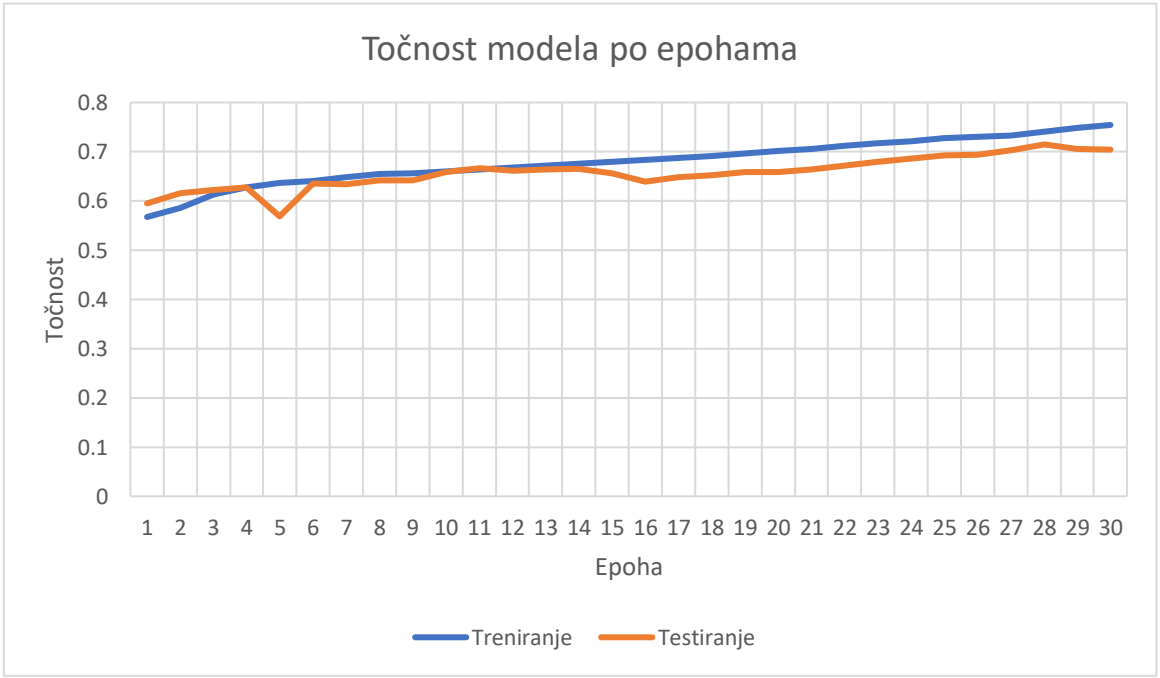
Slika 17: Neuspješni rezultati kolorizacije

Crno-bijela slika | Predviđena kolorizirana slika | Stvarna kolorizirana slika

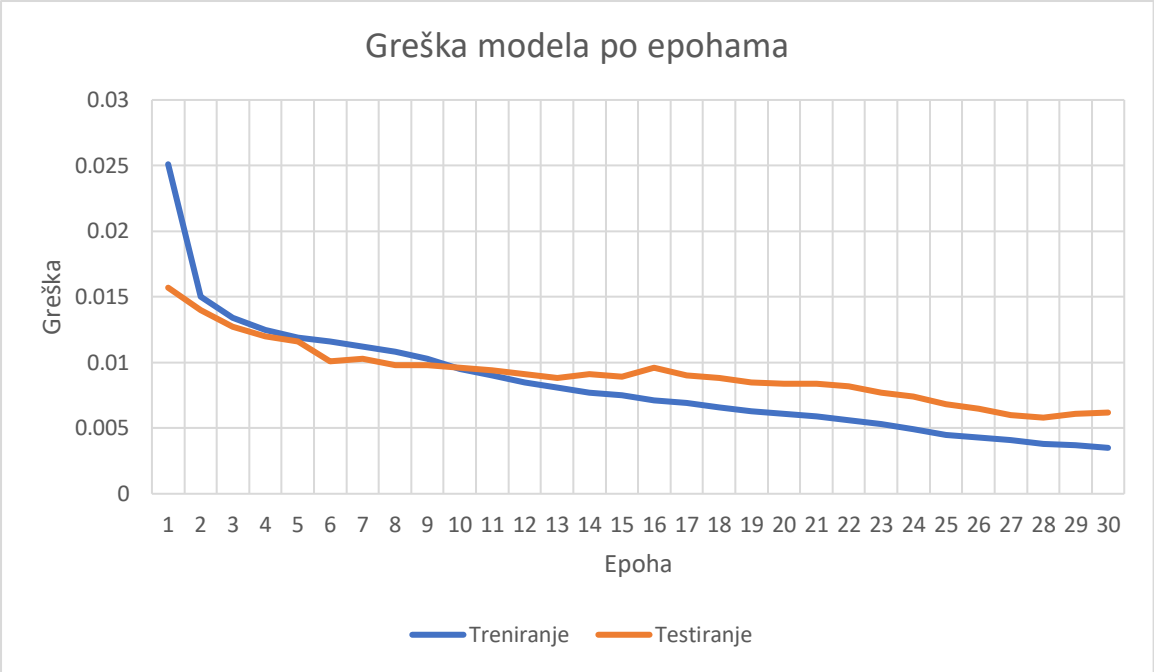


Slika 18: Uspješni rezultati kolorizacije

Model je bio treniran na 95% ukupnog broja podataka, a ostalih 5% su bili iskorišteni za testiranje. Točnost modela je kroz epohe konstantno rasla te je na 30. epohi model dostigao točnost od 75.43%. Točnost modela nad podacima za testiranje na kraju treniranja je iznosila 70.37%, što nije loš omjer, odnosno ne postoji problem prekomjernog prilagođavanja. Greška modela je konstantno padala te je na kraju treniranja iznosila 0.35%, dok je greška nad podacima za testiranja iznosila 0.62%. Postoje dva razloga zašto je prekinuto treniranje nakon 30 epoha. Prvi razlog je što je samo treniranje ovog modela vremenski zahtjevno, a Google Colaboratory ne dopušta dugotrajno treniranje modela (više od 12 sati), pa je sam proces treniranja bio otežan ovim ograničenjem. Drugi razlog koji je presudio u odluci je taj što je model konvergirao te se je greška modela počela povećavati. U slučaju da se model nastavio trenirati, razlika između preciznosti treniranja i testiranja bi bila sve veća te bi zbog toga nastao problem prekomjernog prilagođavanja.



Slika 19: Točnost modela po epohama



Slika 20: Greška modela po epohama

7. Zaključak

Otvorene biblioteke za strojno učenje služe kako bi se olakšala implementacija njegovih algoritama. Od korisnika ne zahtijevaju dubinsko znanje rada algoritma, pa se on može više posvetiti problemima kao što je primjerice postavljanje optimalnih postavki modela za određeni problem. Biblioteke se razlikuju po parametrima kao što su podrška programskih jezika, performanse, pokrivenost algoritama i popularnost. Danas se najčešće koriste biblioteke TensorFlow, PyTorch i Keras, međutim kombinacija TensorFlowa i Kerasa se posebno izdvaja zbog toga što se one mogu zajedno koristiti te zajedno čine jedan vrlo moćan alat. Razlog tome je što Keras omogućava jednostavan API u Pythonu za korištenje složenijih algoritama strojnog učenja kao što su konvolucijske neuronske mreže, dok TensorFlow koji je pisan u C++-u može na brz način izvoditi potrebne matematičke operacije. Prema tome, ove dvije biblioteke iz područja strojnog učenja su korištene pri razvoju aplikacije.

Primjena strojnog učenja na problemu kolorizacije crno-bijelih slika može znatno ubrzati i olakšati proces kolorizacije. Međutim, to je vrlo izazovan problem zbog toga što kolorizirane slike sadrže eksponencijalno više kombinacija boja u odnosu na crno-bijele slike, pa je prema tome teško odrediti boju objekta koji se može pojaviti u više različitih boja. Jedan od najboljih pristupa za kolorizaciju crno-bijelih slika upotrebom strojnog učenja su autoenkoderi koji upotrebljavaju konvolucijske neuronske mreže. Uz pomoć njih se mogu rekonstruirati slike na način da se u koderu nauče značajke slika, a u dekoderu se na temelju značajki te predviđenih boja značajki kolorizira slika. Jedna od mana ovog pristupa je što rezolucija izlazne kolorizirane slike nije dinamična, što znači da će izlazna rezolucija uvijek biti jednaka onoj koja je zadana na specifikaciji modela. Također još jedna mana korištenja strojnog učenja, konkretnije konvolucijskih neuronskih mreža je što postoji problem kolorizacije onih slika koje sadrže manje poznate objekte. Djelomično rješenje može biti ili upotreba već ranije uspješno istreniranog modela na velikom skupu podataka ili pak povećanje skupa podataka te ponovno treniranje na njemu, no i takve alternative neće riješiti problem u potpunosti. Prema tome, najbolje rješenje problema kolorizacije bi bio hibridni pristup. Najprije bi se uz pomoć aplikacije kolorizirala crno-bijela slika, a nakon toga bi čovjek uz pomoć alata za obradu slika ispravio nepravilnosti i očite greške koje je prouzročila aplikacija.

Popis literature

[1] Expert System Team, "What is Machine Learning? A definition", (06.05.2020.). [Na internetu]. Dostupno: https://expertsystem.com/machine-learning-definition/ [pristupano 02.07.2020.].

[2] E. Alpaydin, *Introduction to Machine Learning,* 2. izd., MA, USA: The MIT Press. 2010

[3] V. Nasteski, "An overview of the supervised machine learning methods", pro. 2017. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/directory/publications. [pristupano 03.08.2020]

[4] I. Goodfellow, Y. Bengio, i A. Courville, *Deep Learning*, MA, USA: The MIT Press, 2016.

[5] "Linear regression", [Slika] (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Linear_regression [pristupano 03.08.2020.]

[6] J. Hoare, "What is a Decision Tree?", [Slika] (bez dat.). [Na internetu]. Dostupno: https://www.displayr.com/what-is-a-decision-tree/ [pristupano 06.08.2020.].

[7] S. Tahsildar, "Gini Index For Decision Trees", (18.04.2019.). [Na internetu]. Dostupno: https://blog.quantinsti.com/gini-index/ [pristupano 06.08.2020.].

[8] N. Buduma i N. Locascio, *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*, CA, USA: O'Reilly Media, 2017.

[9] A. Navlani, "Neural Network Models in R", [Slika] (9.12.2019.). [Na internetu]. Dostupno: https://www.datacamp.com/community/tutorials/neural-network-models-r [pristupano 08.08.2020.].

[10] A. Sagar, "5 Techniques to Prevent Overfitting in Neural Networks", pro. 2019. [Na internetu]. Dostupno: https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html [pristupano 08.08.2020.].

[11] A. Dubey, "The Mathematics Behind Principal Component Analysis", (20.12.2018). [Na internetu]. Dostupno: https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-fff2d7f4b643 [pristupano 09.08.2020.].

[12] "kmeans clustering centroid", [Slika] (bez dat.). [Na internetu]. Dostupno: https://pythonprogramminglanguage.com/kmeans-clustering-centroid/ [pristupano 09.08.2020.].

[13] C. M. Bishop, *Pattern Recognition and Machine Learning*, NY, USA: Springer, 2006.

[14] S. Bhatt, "5 Things You Need to Know about Reinforcement Learning", [Slika] ozu. 2018. [Na internetu]. Dostupno: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html [pristupano 14.08.2020.].

[15] "Library (computing)," (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Library_(computing) [pristupano 12.08.2020.]

[16] "scikit-learn: Machine Learning in Python". [Na internetu]. Dostupno: https://scikit-learn.org/stable/ [pristupano 12.08.2020.].

[17] *Scikit-learn logo*, [Slika] (bez dat.). [Na internetu]. Dostupno: https://www.hiclipart.com/free-transparent-background-png-clipart-lyxix [pristupano 13.08.2020.].

[18] S. Yegulalp, "What is TensorFlow? The machine learning library explained", (18.06.2019.). [Na internetu]. Dostupno: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html [pristupano 13.08.2020.].

[19] *TensorFlow logo*, [Slika] (bez dat.). [Na internetu]. Dostupno: https://commons.wikimedia.org/wiki/File:Tensorflow_logo.svg [pristupano 13.08.2020.].

[20] "Keras," (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Keras [pristupano 13.08.2020.]

[21] K. Atey, "Creating a simple Neural Network using Keras for a binary classification task", [Slika] (27.01.2020). [Na internetu]. Dostupno: https://medium.com/analytics-vidhya/creating-a-simple-neural-network-using-keras-for-a-binary-classification-task-545757bb07a5 [pristupano 13.08.2020.].

[22] Microsoft (bez dat.) *ML.NET*. [Na internetu]. Dostupno: https://www.wikiwand.com/en/ML.NET [pristupano 13.08.2020.].

[23] *ML.NET logo*, [Slika] (bez dat.). [Na internetu]. Dostupno: https://www.logo.wine/logo/ML.NET [pristupano 13.08.2020.].

[24] "PyTorch," (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/PyTorch [pristupano 14.08.2020.]

[25] *PyTorch logo*, [Slika] (bez dat.). [Na internetu]. Dostupno: https://commons.wikimedia.org/wiki/File:PyTorch_logo_black.svg [pristupano 14.08.2020.].

[26] Apache (bez dat.) *Apache MXNet*. [Na internetu]. Dostupno: https://mxnet.apache.org/versions/1.6/features [pristupano 14.08.2020.].

[27] *Apache MXNet logo*, [Slika] (bez dat.). [Na internetu]. Dostupno: https://www.redbubble.com/es/people/james9834/works/30758620-mxnet-logo [pristupano 14.08.2020.].

[28] *Google trendovi*, [Slika] (bez dat.). [Na internetu]. Dostupno: https://trends.google.hr/trends [pristupano 15.08.2020.].

[29] R. Prabhu, "Understanding of Convolutional Neural Network (CNN) — Deep Learning", (04.03.2018.). [Na internetu]. Dostupno: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 [pristupano 17.08.2020.].

[30] COCO – Common Objects in Context (bez dat.). [Na internetu]. Dostupno: https://cocodataset.org/#home [pristupano 16.08.2020]

[31] "CIELAB color space," (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/CIELAB_color_space [pristupano 17.08.2020.]

[32] Keras (bez dat.) Keras API [Na internetu]. Dostupno: https://keras.io/api/ [pristupano 16.08.2020]

[33] Tensorflow (bez dat.) Tensorflow API [Na internetu]. Dostupno: https://www.tensorflow.org/api_docs [pristupano 16.08.2020]

[34] Google (bez dat.) Google Colab [Na internetu]. Dostupno: colab.research.google.com [pristupano 16.08.2020]

[35] Q. Fu, W.T. Hsu, i M.H. Yang, "Colorization Using ConvNet and GAN", 2017. [Na internetu]. Dostupno: http://cs231n.stanford.edu/reports/2017/pdfs/302.pdf. [pristupano 18.08.2020]

Popis slika

Popis tablica