Utjecaj hiperparametara na rad modela dubokog učenja

Sever, Mislav

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mislav Sever

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

izv. prof. dr. sc. Tomislav Stipančić, dipl. ing. Mislav Sever

Zagreb, 2023.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru izv. prof. dr. sc. Tomislavu Stipančiću na pristupačnosti, vremenu, korisnim savjetima i pruženoj pomoći tijekom izrade ovog rada.

Također se zahvaljujem svojim roditeljima i prijateljima na podršci tijekom preddiplomskog dijela studija.

Mislav Sever

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 23 – 6 / 1**	
Ur.broj: **15 - 1703 - 23 -**	

ZAVRŠNI ZADATAK

Student: **Mislav Sever** JMBAG: **0035221290**

Naslov rada na hrvatskom jeziku: **Utjecaj hiperparametara na rad modela dubokog učenja**

Naslov rada na engleskom jeziku: **The influence of hyperparameters on the performance of deep learning models**

Opis zadatka:

Tehnike dubokog učenja se sve više primjenjuju u inženjerstvu te postižu zapažene rezultate u različitim primjenama. Modeli temeljeni na tehnikama dubokog učenja koriste velike količine podataka kako bi uspješno predvidali, segmentirali, tražili uzorke, klasificirali i sl. Uspješnost izvedbe rada modela uvelike ovisi o kvaliteti podataka, te o parametrima učenja (tzv. hiperparametrima).

U radu je potrebno:

- izgraditi i uvježbati vlastiti model konvolucijske neuronske mreže proizvoljne strukture na proizvoljnom skupu podataka,
- odrediti i objasniti hiperparametre za poboljšanja rada mreže,
- načiniti analizu te odabrati prikladne hiperparametre te ih dovesti u vezu s izvedbom modela,
- dati kritički osvrt na rezultate te predložiti moguća poboljšanja.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:
30. 11. 2022.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
1. rok: 20. 2. 2023.
2. rok (izvanredni): 10. 7. 2023.
3. rok: 18. 9. 2023.

Predviđeni datumi obrane:
1. rok: 27. 2. – 3. 3. 2023.
2. rok (izvanredni): 14. 7. 2023.
3. rok: 25. 9. – 29. 9. 2023.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

Oznaka	Jedinica	Opis
A_{CC}	/	točnost
a_i	/	ulazni signal
b_j	/	izlazni signal
F	/	F1 - mjera
f	/	aktivacijska funkcija
FN	/	broj lažno negativnih slučajeva
FP	/	broj lažno pozitivnih slučajeva
P	/	preciznost
R	/	odziv
T_j	/	prag osjetljivosti
TN	/	broj istinito negativnih slučajeva
TP	/	broj istinito pozitivnih slučajeva
w_i	/	težinski faktor

SAŽETAK

Tema ovog rada je utjecaj hiperparametara na rad modela dubokog učenja. Hiperparametri predstavljaju parametre koji kontroliraju proces učenja. Kao model dubokog učenja odabrana je duboka konvolucijska neuronska mreža koja predstavlja temelj algoritama računalnog vida. Rad se sastoji od teorijskog i praktičnog dijela. U sklopu teorijskog dijela iznesena je sva teorija potrebna za razumijevanje utjecaja pojedinih hiperparametara na rad modela dubokog učenja. Praktični dio rada sastoji se od izrade modela duboke konvolucijske neuronske mreže i podešavanja njegovih hiperparametara u programskom jeziku Python.

Ključne riječi: hiperparametri, duboko učenje, konvolucijska neuronska mreža, računalni vid, Python

SUMMARY

The theme of this work is the influence of hyperparameters on the performance of the deep learning models. Hyperparameters represent parameters that control the learning process. A deep convolution neural network is selected as a deep learning model that forms the basis of computer vision algorithms. The work consists of a theoretical and practical part. The theoretical part sets out all the theories necessary to understand the influence of individual hyperparameters on the performance of the deep learning models. The practical part of the work consists of developing a model of a deep convolution neural network and setting up its hyperparameters in the Python programming language.

Key words: hyperparameters, deep learning, convolutional neural network, computer vision, Python

1. UVOD

Duboko učenje kao grana strojnog učenja danas pronalazi primjenu u različitim područjima od kojih se najviše ističu računalni vid, prepoznavanje govora i automatsko prevođenje jezika. Ljudi počinju shvaćati kako umjetna inteligencija nije daleka budućnost nego je ona sadašnjost i počinju se sve više oslanjati na njezine mogućnosti. Od modela temeljenih na tehnikama dubokog učenja očekuje se velika brzina procesiranja zadanog skupa podataka i visoka točnost krajnjeg rezultat. Brzina i točnost uvelike ovise o kvaliteti zadanog skupa podataka kao i o parametrima učenja tzv. hiperparametrima koji su tema ovoga rada. U sklopu ovoga rada proučiti će se utjecaj hiperparamatera na rad konvolucijske neuronske mreže i pokušati će se odrediti optimalni hiperparametri uz koje će se dobivati točni rezultati u što kraćem vremenskom periodu. Unutar rada će se prvo iznijeti teorijska podloga bitna za razumijevanje cjelokupnog rada, zatim će se prikazati izrada modela konvolucijske neuronske mreže i prilagođavanje hiperparametara, a na samom kraju će se dati zaključci vezani uz rezultate i predložiti moguća poboljšanja.

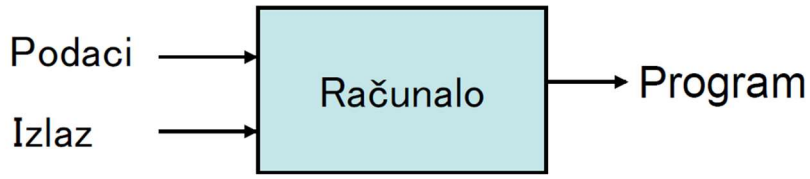
2. TEORIJSKA PODLOGA RADA

2.1. Umjetna inteligencija

Umjetna inteligencija omogućuje računalnim sustavima izvršavanje zadataka koji zahtijevaju ljudsku inteligenciju. Razlikujemo jaku i slabu umjetnu inteligenciju. Primjer slabe ili uske umjetne inteligencije su strojevi koji reagiraju na specifične situacije, no oni ne mogu misliti, za razliku od strojeva s jakom umjetnom inteligencijom koji mogu razmišljati i djelovati kao ljudi učeći kroz iskustva. [1]

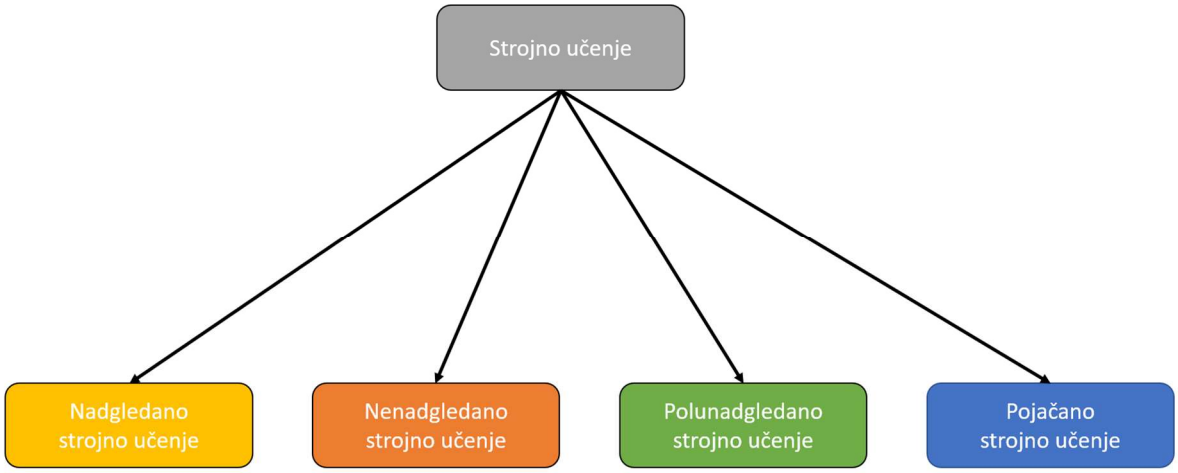
2.2. Strojno učenje

Strojno učenje je jedna od grana umjetne inteligencije koja omogućava računalima da programiraju sama sebe. Način na koji funkcionira strojno učenje je prikazan na [Slika 1.]. Kao ulaz u algoritam strojnog učenja se koristi zadani skup podataka i željeni izlaz tj. rješenje nekog zadanog problema, a izlaz iz algoritma je program tj. skup pravila koji za zadani skup podataka daje željeni izlaz. [1]



Slika 1. Model strojnog učenja

Osnovna podjela strojnog učenja prema [1] je prikazana na [Slika 2.].



Slika 2. Podjela strojnog učenja

Nadgledano strojno učenje

Kod nadgledanog strojnog učenja skup podataka za trening je označen tj. on sadrži željene izlazne podatke. Skup podataka za trening služi za treniranje algoritama koji se kasnije primjenjuju za klasifikaciju i predviđanje događaja. Neke vrste algoritama nadgledanog strojnog učenja su neuronske mreže, linearna regresija, logistička regresija i ostali. [2]

Nenadgledano strojno učenje

Skup podataka za trening kod nenadgledanog strojnog učenja je neoznačen tj. on ne sadrži željene izlazne podatke. Algoritmi nenadgledanog strojnog učenja analiziraju i grupiraju neoznačene skupove podataka bez prisustva ljudi. [2]

Polunadgledano strojno učenje

Polunadgledano strojno učenje predstavlja spoj nadgledanog i nenadgledanog strojnog učenja. Podaci za trening sadrže manji skup izlaznih (označenih) podataka koji se prilikom učenja koriste za usmjeravanje klasifikacije i grupiranje većeg neoznačenog skupa podataka. [1][2]

Pojačano strojno učenje

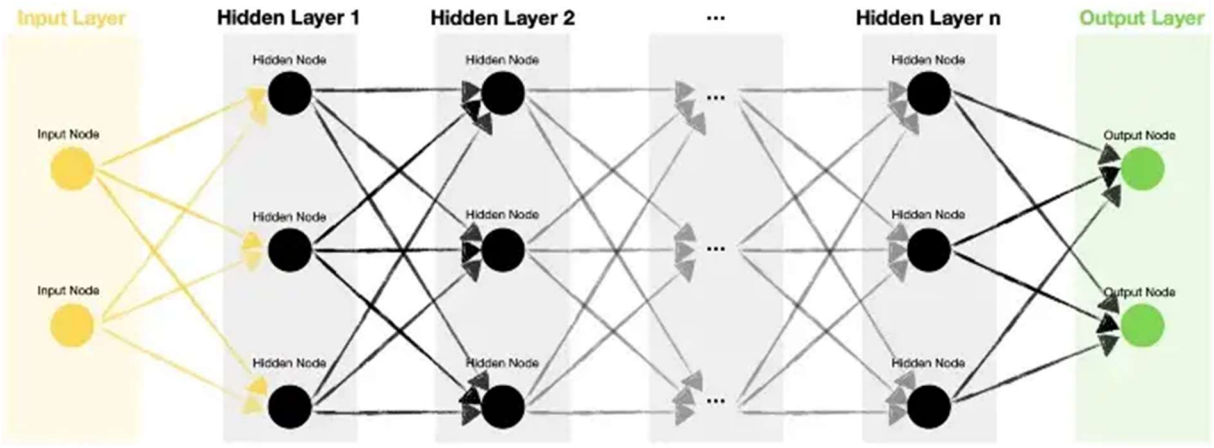
Pojačano strojno učenje podrazumijeva pronalaženje optimalnog ponašanja u nekoj određenoj situaciji metodom pokušaja i pogrešaka koristeći povratne informacije. Ishodi sa pozitivnom povratnom informacijom će biti uzeti u obzir, dok će ishodi sa negativnom povratnom informacijom biti odbačeni. Za razliku od nadgledanog strojnog učenja pojačano ne sadrži označeni skup podatak za trening. Pojačano strojno učenje je vrsta učenje u kojem model uči iz vlastitog iskustva. [2]

2.3. Umjetne neuronske mreže

Predstavljaju podskup strojnog učenja i temelj su algoritma dubokog učenja. Umjetne neuronske mreže su stvorene po uzoru na ljudski mozak koji se sastoji od 10^{11} bioloških neurona koji rade paralelno. Prema [3] umjetne neuronske mreže su definirane kao skup međusobno povezanih čvorova čija se funkcionalnost temelji na biološkom neuronu i koji služe paralelnoj obradi podataka. Umjetne neuronske mreže su se pokazale dobre u rješavanju problema kod kojih postoji nelinearna veza ulaza i izlaza tj. u rješavanju problema klasifikacije i predviđanja. [3]

2.3.1. Struktura umjetne neuronske mreže

Struktura umjetne neuronske mreže prema [4] je prikazana na [Slika 3.]

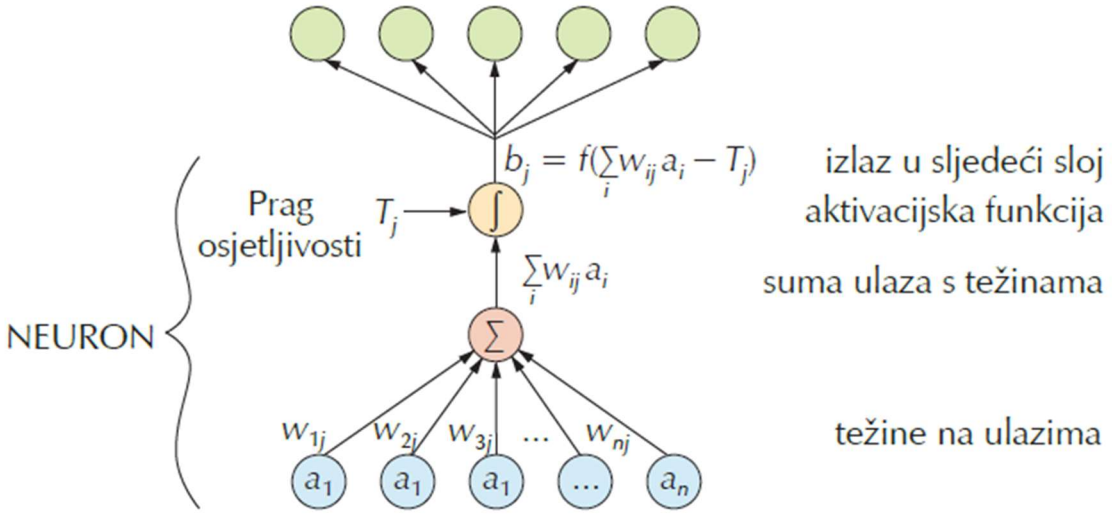


Slika 3. Struktura umjetne neuronske mreže

Struktura neuronske mreže se sastoji od slojeva unutar kojih se nalaze čvorovi. Pojedini čvor tj. umjetni neuron iz jednog sloja se povezuje sa jednim ili više čvorova iz drugih slojeva, a te veze su opisane težinskim faktorima, pragovima i prijenosnom tj. aktivacijskom funkcijom. Unutar neuronske mreže postoji više slojeva odnosno ulazni sloj, jedan ili više sakrivenih slojeva i izlazni sloj.

2.3.2. Način rada umjetne neuronske mreže

Način rada umjetne neuronske mreže prikazati će se na modelu perceptrona prikazanog na [Slika 4.]. [5] Perceptron je najstarija i najjednostavnija vrsta neuronske mreže koja sadrži samo jedan umjetni neuron.



Slika 4. Model perceptrona

Kao što je već prije spomenuto svaka neuronska mreža se sastoji od ulaznog sloja, sakrivenog sloja i izlaznog sloja. Ulazni sloj predstavljaju ulazni signali, a_i. Ulazni signali se množe odgovarajućim težinskim faktorima w_i koji definiraju važnost određenog ulaznog signala. Množenje ulaznih signala s težinskim faktorima predstavlja sinaptičku operaciju. Unutar sakrivenog sloja otežani ulazni signali se sumiraju pomoću neke od funkcija sumiranja, dobivena suma se zatim uspoređuje s pragom osjetljivosti neurona, T_j. Ukoliko je dobivena suma veća od praga osjetljivosti neurona aktivacijska funkcija f generira izlazni signal b_j koji predstavlja izlazni sloj neuronske mreže. [5]

Postoje različite vrste aktivacijskih funkcija od kojih su najpoznatije sigmoidna, step, linearna, ReLU, Softmax i tangens – hiperbolna. [6]

2.3.3. Vrste umjetnih neuronskih mreža

Peceptron

Kao što smo već spomenuli perceptron je najstarija vrsta umjetnih neuronskih mreža. Izumio ju je Frank Rosenblatt 1958. godine. To je najjednostavnija vrsta iz razloga što sadrži samo jedan neuron. Kao aktivacijska funkcija se koristi step funkcija čiji izlaz može poprimiti vrijednost 0 ili 1. [3]

Statičke unaprijedne neuronske mreže (engl. Feedforward Neural Networks)

Statičke unaprijedne neuronske mreže sadrže više slojeva (ulazni, nekoliko sakrivenih i izlazni sloj). Kako je većina stvarnih problema ne linearna kao aktivacijska funkcija se koristi sigmodina funkcija čiji se izlaz može nalaziti u rasponu između 0 i 1. Koriste se za prepoznavanje slika, računalni vid i ostalo. [3]

Konvolucijske neuronske mreže (engl. Convolutional Neural Networks – CNNs)

Kovolucijske neuronske mreže su slične statički unaprijednim mrežama no one pored već spomenutog ulaznog, sakrivenog i izlaznog sloja sadrže i kovolucijski sloj. Koriste se prvenstveno za računalni vid, prepoznavanje slika i prepoznavanje uzoraka. Način rada konvolucijskih neuronskih mreža će biti objašnjen kasnije unutar rada. [3]

Povratne neuronske mreže (engl. Recurrent Neural Networks – RNNs)

Povratne neuronske mreže su specifične po svojim povratnim vezama. Koriste se za predviđanje budućih ishoda. [3]

2.4. Duboko učenje

Duboko učenje je jedna od grana strojnog učenja, a odnosi se na dubinu neuronske mreže tj. na broj slojeva koje ona sadrži. Algoritmom dubokog učenja se može nazvati neuronska mreža koja se sastoji od najmanje tri sloja uključujući ulazni i izlazni sloj. Veći broj sakrivenih slojeva omogućava neuronskoj mreži učenje iz većeg skupa podataka, a uz to povećava i njezinu točnost. [7]

Model dubokog učenja sadrži algoritme za izračunavanje pogreške u predviđanju događaja koji ujedno prilagođavaju ranije spomenute težinske faktore i pragove, a sve to kako bi model postao što precizniji. Kod strojnog učenja parametri takvog tipa se ručno prilagođavaju od strane čovjeka što je glavna razlika u odnosu na modele dubokog učenja. [7]

2.5. Učenje umjetne neuronske mreže

Učenje ili treniranje umjetne neuronske mreže odnosi se na postupak iterativnog prilagođavanja težinskih faktora kako bi smanjila razlika između vrijednosti proračunate modelom tj. predviđene vrijednosti i stvarne vrijednosti. Jedan od načina prilagođavanja težinskih faktora je algoritam unatražne propagacije izlazne pogreške (engl. back propagation). [5]

2.5.1. Postupak učenja

Postupak učenja tj. treniranja započinje podjelom podataka na tri dijela: skup za učenje, skup za provjeru i skup za testiranje. Učitavanjem skupa za učenje na neuronsku mrežu kreće postupak prilagođavanja težinskih faktora pomoću nekog od algoritama tako dugo dok se pogreška ne minimizira tj. dok pogreška ne bude manja od zadanog odstupanja. Nakon svake iteracije iznos pogreške se provjerava nad skupom za provjeru. Završna provjera modela odvija se nad skupom za testiranje, ukoliko model ne pruža zadovoljavajuće rezultate tada se mora mijenjati struktura modela. [5]

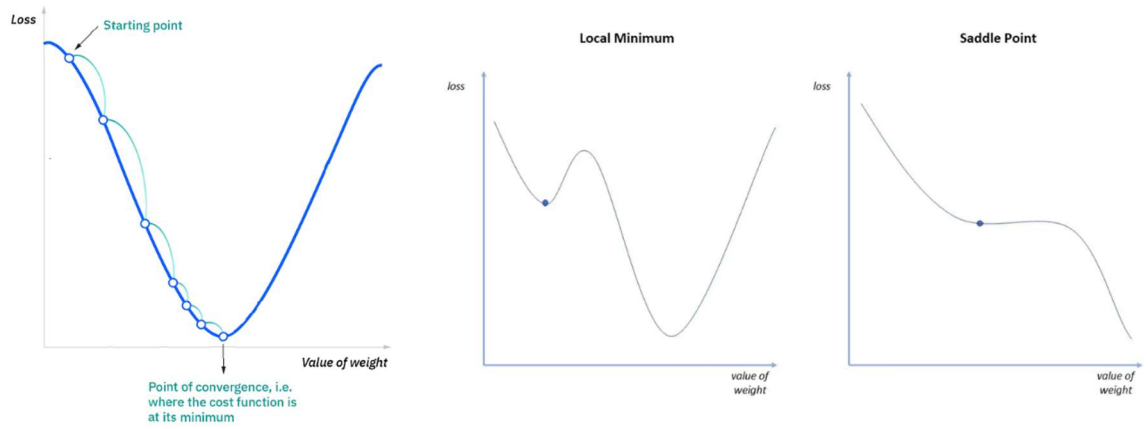
2.5.2. Funkcija pogreške

Funkcija pogreške (engl. cost function) predstavlja razliku između modelom predviđene vrijednosti i stvarne vrijednosti. Izuzetno je bitna prilikom određivanja točnosti modela. Cilj je tijekom postupka učenja postići minimum funkcije pogreške tj. odrediti težinske faktore za koje će pogreška biti minimalna. [3]

U literaturi se često spominje i naziv funkcija gubitka (engl. loss function) koja se odnosi na pogrešku jednoga primjera za učenje dok se funkcija pogreške odnosi na pogrešku cjelokupnog skupa za učenje. [8]

U većini slučajeva funkcija pogreške nije konveksna već ona sadrži veliku količinu lokalnih minimuma gdje je naš cilj pronaći globalni minimum. Također problematične znaju biti točke sedla u kojima je gradijent jednak nuli. [3]

Primjer jednostavne konveksne funkcije pogreške je prikazan lijevo, ne konveksna funkcija pogreške je prikazana u sredini i desno se nalazi funkcija pogreške sa točkom sedla na [Slika 5.] [3]



Slika 5. Funkcije pogreške

2.5.3. Algoritam unatražne propagacije izlazne pogreške

Kao što je to već ranije spomenuto algoritam unatražne propagacije izlazne pogreške služi za prilagođavanje težinskih faktora u procesu učenja tj. treniranja. [9]

Prilagođavanje težinskih faktora se temelji na izračuni gradijenta pada (engl. gradient descent) tj. parcijalnih derivacija ranije spomenute funkcije pogreške. Vrijednost gradijenta tj. nagib govori da li trebamo povećati ili smanjiti vrijednost težinskog faktora što nam uvelike pomaže kod pronalaska minimuma funkcije pogreške. Ako je nagib negativan potrebno je povećati vrijednost težinskog faktora, no dok je pozitivan tada treba smanjiti vrijednost težinskih faktora kako bi se postigao minimum funkcije. [9]

Izračun gradijenta i prilagođavanje težinskih faktora je iterativan postupak kod kojeg se algoritam svaki put nakon izračuna vraća na početak tako dugo dok je vrijednost gradijenta različita od 0 tj. tako dugo dok se ne postigne minimum funkcije pogreške. Svaki ciklus vraćanja unatrag i ponovnog prilagođavanja težinskih faktora se naziva epoha i ona predstavlja vrlo bitan hiperparametar čiji će utjecaj na točnost modela biti objašnjen kasnije. [9]

Vrijednost za koju se povećavaju ili smanjuju težinski faktori unutar jedne epohe naziva se stopa učenja. Stopa učenja je najbitniji hiperparametar kod algoritma unatražne propagacije.

Ukoliko je stopa učenja prevelika minimum funkcije pogreške se može preskočiti, no ako je premala algoritam bi se mogao beskonačno odvijati i nikad ne bi postigao minimum funkcije. [8]

Unutar ovoga rada će se pokušati odrediti optimalna vrijednost spomenutog hiperparametra.

Postoje različite vrste gradijenta pada koje su razvijene kako bi se riješio problem ranije spomenutih lokalnih minimuma i sedlastih točaka kod funkcije pogreške. Vrsta gradijenta pada se također može nazvati hiperparametrom budući da pravilan odabir znatno utječe na točnost modela. U nastavku su nabrojane i opisane vrste gradijenta pada.

Grupni gradijent pada (engl. Batch Gradient Descent)

Grupni gradijent pada prilagođava težinske faktore modela nakon što obradi čitav skup podataka za učenje. Računalima je ovakav model lakši za računati, no vrijeme potrebno za procesiranje većih skupova podataka može biti dugo. Vrlo lako se može desiti da se odredi lokalni minimum umjesto globalnog minimuma. [8]

Pojedinačni (stohastički) gradijent pada (engl. Stochastic Gradient Descent)

Kod pojedinačnog gradijenta pada prilagođavanje težinskih faktora se odvija nakon što se obradi svaki pojedinačni primjer za učenje. Ovakav model je puno zahtjevniji za procesiranje od strane računala no može pružiti veću brzinu i količinu detalja što omogućuje izbjegavanje lokalnog minimuma tj. određivanje globalnog minimuma. [8]

Mini – grupni gradijent pada (engl. Mini - Batch Gradient Descent)

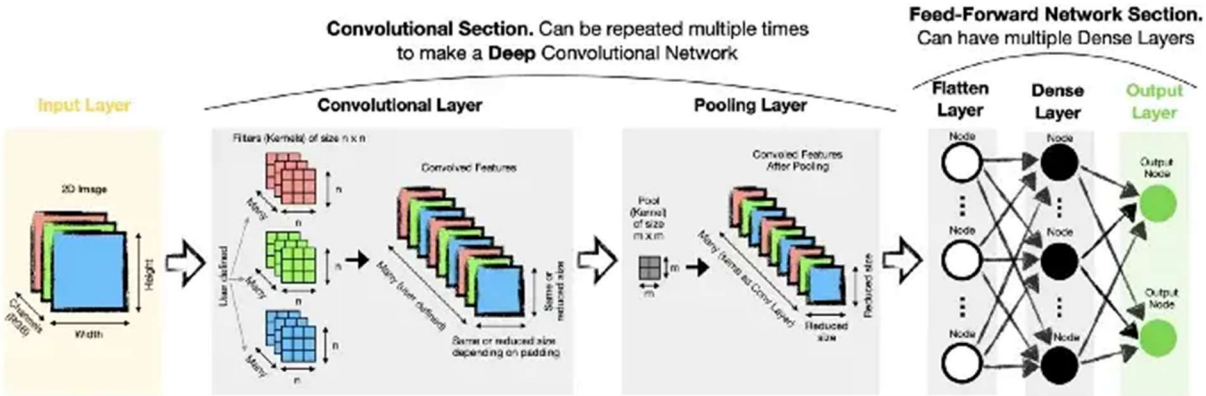
Mini – grupni gradijent pada je spoj prethodne dvije vrste. Zadani skup podataka za učenje grupira u manje grupe nakon čega prilagođavanje težinskih faktora izvodi nakon što se obradi svaka grupa pojedinačno. Ovakav pristup donekle omogućava lakše procesiranje podataka računalima i postizanje optimalne brzine i razine detalja. [8]

2.6. Duboke konvolucijske neuronske mreže

Duboke konvolucijske neuronske mreže (engl. deep convolutional neural networks) predstavljaju temelj algoritama računalnog vida (engl. computer vision). Računalni vid omogućuje računalima donošenje određenih zaključaka na temelju slike ili videozapisa. Primjena računalnog vida se može pronaći u različitim granama industrije. Neke primjene računalnog vida u industriji su inspekcija velike količine proizvoda koji se transportiraju pomoću pokretne trake, kontrola alata za vrijeme strojne obrade, detekcija stranih objekata ili čak očitavanje vrijednosti sa analognih i digitalnih mjernih instrumenata što se može vidjeti na [Slika 6.]. [10]

Slika 6. Primjena računalnog vida u industriji

Struktura konvolucijskih neuronskih mreža je vrlo slična strukturi statički unaprijednih mreža tj. ona pored ulaznog sloja, jednog ili više sakrivenih slojeva i izlaznog sloja sadrži konvolucijski sloj uz koji se može pronaći, sloj udruživanja (engl. pooling layer), sloj poravnanja (engl. flatten layer) i sloj gustoće (engl. dense layer) što se može vidjeti na [Slika 7.]. [4]



Slika 7. Struktura konvolucijske mreže

2.6.1. Konvolucijski sloj

Konvolucijski sloj je zadužen za prepoznavanje uzoraka, objekata i oblika na slikama ili videozapisima. Ovisno o broju dimenzija razlikujemo 1D, 2D i 3D konvolucijske slojeve. U sklopu ovog rada koristiti će se 2D konvolucjski slojevi čija je namjena obrada slike, dok je kod 1D to obrada teksta odnosno kod 3D obrada videozapisa ili 3D slika. Također će se u sklopu ovoga rada koristiti konvolucijski sloj s tri kanala kako bi se omogućila obrada slike u boji, tu postoji i opcija sa jednim kanalom za obradu crno – bijelih slika. Još jedan od vrlo bitnih parametara je filter. Filter je manjih dimenzija od slike koja se obrađuje što znači da se obrada slike odvija u više koraka. Dimenzija i sadržaj samog filtera se prilagođava tijekom procesa učenja dok broj filtera prilagođava korisnik tj. on se može definirati kao hiperaparametar. Proces konvolucije se izvodi na način da se filter primjenjuje na određene dijelove slike. Što se više piksela podudara između slike i filtera to je veća vrijednost konvolvuirane značajke (engl. convolved feature) koja predstavlja izlaznu veličinu kovolucijskog sloja. Kod konvolucijskog sloja postoje i dvije dodatne opcije tj. ispuna (engl. padding) i korak (engl. stride) čije vrijednosti predstavljaju hiperparametre. Opcija ispune se koristi kada želimo da nam izlaz iz konvolucijskog sloja bude jednakih dimenzija kao i ulaz, dok se opcija korak koristi kada želimo smanjiti dimenzije izlaza u odnosu na ulaz u konvolucijsku mrežu. Dubina kovolucijske neuronske mreže je povezana sa brojem konvolucijskih slojeva koji ujedno predstavlja jedan od hiperaparametra, veći broj konvolucijskih slojeva omogućava obradu složenijih slika i pruža točnije rezultate. [4]

2.6.2. Sloj udruživanja

Sloj udruživanja se dodaje nakon konvolucijskog sloj u svrhu udruživanja tj. objedinjavanja konvolvuiranih značajku kako bi se smanjila njihova veličina. Manja veličina konvolvuiranih značajki skraćuje vrijeme procesiranja. Pored toga omogućuje i uklanjanje tzv. šuma iz podataka tj. uklanjanje onih značajki koje se značajno razlikuju od drugih iz zajedničkog skupa. [4]

Prema [4] postoje dvije vrste udruživanja:

- maksimalno udruživanje (engl. max pooling) – uzima maksimalnu vrijednost konvolvuiranih značajki iz skupa čije su dimenzije definirane dimenzijama sloja udruživanja,
- prosječno udruživanje (engl. average pooling) - uzima srednju vrijednost konvolvuiranih značajki iz skupa čije su dimenzije definirane dimenzijama sloja udruživanja.

2.6.3. Sloj poravnanja i sloj gustoće

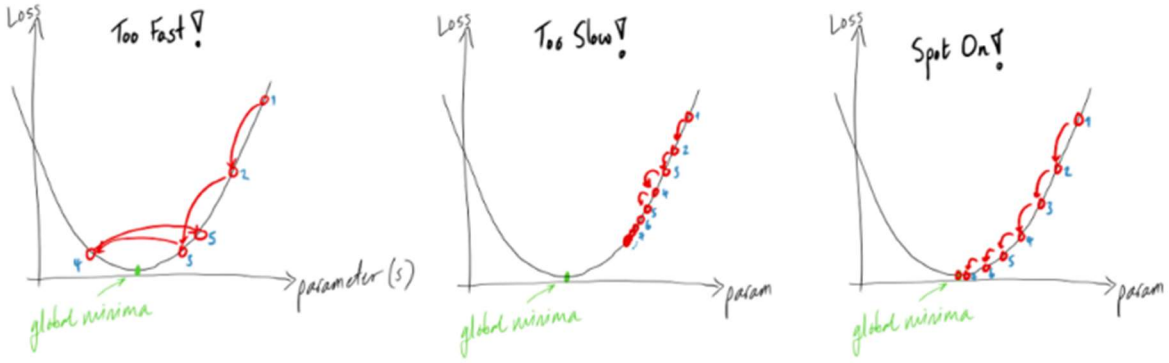
Kod korištenja višedimenzionalnih konvolucijskih mreža izlazne podatke treba prilagoditi kako bi se oni mogli dalje koristiti kod statički unaprijednih neuronskih mreža koje kao ulazne podatke koriste jednodimenzijske vektore. Iz toga razloga se koriste sloj poravnanja i sloj gustoće. [4]

2.7. Hiperparametri

Hiperparametri su parametri koji kontroliraju proces učenja i utječu na preciznost i brzinu samog modela. Hiperparametri ne proizlaze iz procesa učenja tj. čovjek njih mora definirati prije samog procesa učenja te se često kaže da se oni nalaze izvan modela. Tijekom prethodnih poglavlja navedeni su neki od hiperparametara, a unutar ovog poglavlja će se detaljno objasniti njihov utjecaj na rad modela dubokog učenja.

2.7.1. Stopa učenja

Kao što je to već ranije spomenuto stopa učenja predstavlja vrijednost za koju povećavamo ili smanjujemo vrijednost težinskih faktora unutar jedne epohe. Stopa učenja predstavlja najbitniji hiperparametar. Na [Slika 8.] je prikazano kako vrijednost stope učenja utječe na postizanje globalnog minimuma.



Slika 8. Utjecaj stope učenja na pronalazak globalnog minimuma

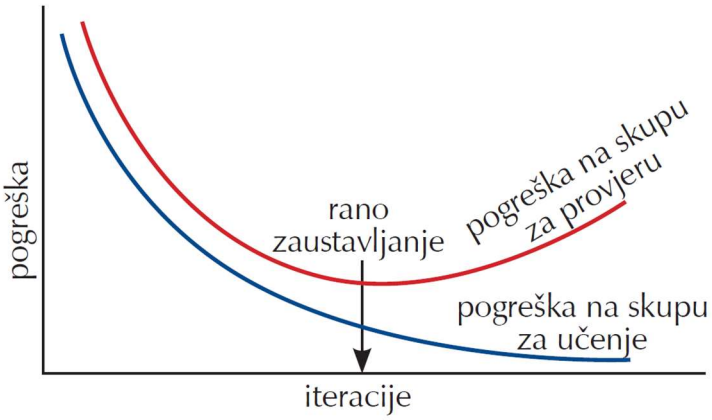
Na lijevoj slici je prikazan slučaj kada je vrijednost stope učenja prevelika, u tom slučaju dolazi do preskakanja globalnog minimuma funkcije pogreške. Slika u sredini prikazuje slučaj kada je stopa učenja premala, a u tom slučaju globalni minimum funkcije pogreške se neće nikad postići. Idealna vrijednost stope učenja je ona prikazana na desnoj slici tj. ona kod koje je moguće postići globalni minimum funkcije pogreške. [11]

Na prethodnoj slici je pokazan primjer konveksne funkcije pogreške, no u stvarnosti ona posjeduje veći broj lokalnih minimuma među kojima mi moramo pronaći globalni minimum.

Korištenjem konstantne vrijednosti stope učenja nećemo postići globalni minimum, već je potrebno koristiti adaptivnu vrijednost stope učenja tj. mijenjati vrijednost stope učenja nakon određenog broja epoha. [11]

2.7.2. Broj epoha

Broj epoha je vrlo bitan hiperparametar koji nam govori u kojem trenutku je potrebno zaustaviti proces učenja kako ne bi došlo do efekta koji se naziva „pretreniranje“ modela. Pretreniranje modela je efekt kod kojeg model nad skupom podataka za učenje daje dobre rezultate, dok nad skupom za podataka za provjeru daje loše rezultate tj. ima veću pogrešku. Razlog tome je prikazan na [Slika 9.]. [5]



Slika 9. Efekt „pretreniranja“ modela

Kao što se može vidjeti sa slike, nakon određenog broja epoha tj. iteracija pogreška nad skupom za učenje i dalje nastavlja padati, dok nad skupom za provjeru počinje rasti. Upravo iz tog razloga rezultati nad skupom za provjeru su loši. Postupak zaustavljanja procesa učenja nakon određenog broja epoha kada pogreška nad skupom za provjeru počne rasti naziva se tehnikom ranog zaustavljanja (engl. early stopping).

2.7.3. Veličina grupe kod mini – grupnog gradijenta pada

U poglavlju *2.5.3.* smo naveli kako postoje različite metode za izračun gradijenta pada kod algoritma unatražne propagacije izlazne greške. Pored grupnog i pojedinačnog najčešće se koristi mini – grupni gradijent pada. Veličina grupe kod mini – grupnog gradijenta pada je značajan hiperparametar koji istovremeno utječe na točnost i brzinu samog modela. Što je veća grupa to će biti potrebno više memorije i procesorske moći kako bi se obradio zadani skup podataka ali će rezultati biti precizniji. [11]

2.7.4. Broj sakrivenih slojeva

Broj sakrivenih slojeva kao i konvolucijskih slojeva, slojeva pridruživanja, poravnanja i gustoće kod konvolucijskih neuronskih mreža bitno utječe na rad modela dubokog učenja. Velikim brojem navedenih slojeva dobivamo jako složeni model kod kojeg može doći do efekta „pretreniranja“ modela. Točan broj pojedinih slojeva je vrlo teško odrediti. [11]

2.7.5. Hiperparametri kod konvolucijskih neuronskih mreža

Kod konvolucijskih neuronskih mreža se najviše ističu sljedeći hiperparametri:

- broj filtera kod procesa konvolucije,
- vrsta aktivacijske funkcije,
- iznos koraka,
- dimenzije ispune,
- dimenzije sloja udruživanja i
- vrsta udruživanja.

2.7.6. Podešavanje hiperparametara

Podešavanje hiperparametara (engl. hyperparameter tuning) se odvija metodom pokušaja i pogrešaka. Hiperparametri se podešavaju ručno, no ipak postoje određeni algoritmi čiji su ulazni podaci različite vrijednosti više različitih hiperparamtera od kojih algoritam odabere najbolju kombinaciju tj. onu koja rezultira najmanjom greškom modela. U sklopu ovoga rada će se pokušati implementirati takvi algoritmi. Bitno je napomenuti kako ne ovisi sve o algoritmu, najveći faktor je iskustvo i znanje osobe koja se bavi izradom modela dubokog učenja.

3. IZRADA ZADATKA

Cilj je izraditi model duboke konvolucijske neuronske mreže, definirati skupove podataka za treniranje i testiranje, odabrati odgovarajuće hiperparamtere, te na temelju njih i definiranog skupa podataka uvježbati model konvolucijske mreže.

Za izradu modela konvolucijske neuronske mreže odabran je programski jezik Python. Python je najčešće korišteni programski jezik u području umjetne inteligencije i strojnog učenja. Razlog tome je to što Python sadrži veliki broj biblioteka usmjerenih prema rješavanju različitih problema glede umjetne inteligencije i strojnog učenja. Biblioteke sadrže već unaprijed napisane kodove u obliku funkcija što uvelike pomaže prilikom pisanja programa. Pored toga sama sintaksa unutar Pythona je vrlo jednostavna i postoji puno dokumentacije na internetu što omogućava brzo učenje samog jezika. [12]

3.1. Biblioteke

Unutar ovoga rada su korištene sljedeće biblioteke:

TensorFlow

TensorFlow je biblioteka otvorenog koda razvijena od strane Google Brain tima koja obuhvaća niz modela i algoritama strojnog učenja. Izuzetno je korisna prilikom izrade dubokih neuronskih mreža namijenjenih za klasifikaciju, obradu slika i prepoznavanje govora. [13]

Keras

Keras je također biblioteka otvorenog koda koja predstavlja sučelje za TensorFlow biblioteku. Koristi se za izgradnju neuronskih mreža. Glavna prednost ove biblioteke je u podržavanja gotovo svih modela neuronskih mreža. Rad s Keras bibliotekom je izuzetno jednostavan pa je vrlo često korištena od strane početnika. [14]

NumPy

NumPy je jedna od temeljnih biblioteka koja se koristi za izvođenje različitih matematičkih i transformacijskih operacija nad višedimenzionalnim poljima podataka. Primjena NumPy – a se može pronaći i unutar ostalih biblioteka što ga čini neizbježnim alatom u Python programskom jeziku. [15]

Matplotlib

Matplotlib je biblioteka namijenjena za grafičko prikazivanje podataka tj. izradu grafova i ostalih vizualizacija unutar Python – a i NumPy biblioteke. [16]

Pandas

Pandas je biblioteka otvorenog koda koja se koristi za analizu i obradu podataka kao i za zadatke vezane uz strojno učenje. Ova biblioteka se temelji na NumPy biblioteci. Neke od operacija nad podacima koje omogućava Pandas biblioteka su učitavanje i spremanje podataka, statistička analiza podataka, čišćenje podataka i ostale. [17]

Scikit Learn (SKLearn)

Scikit Learn je najpoznatija i najrobusnija biblioteka strojnog učenja koja sadrži različite algoritme i alate koji omogućuju vizualizaciju, pretprocesiranje, treniranje i evaluaciju. Navedena biblioteka se temelji na NumPy, SciPy i Matplotlib bibliotekama. Glavna značajka Sckit Learn biblioteke je izrada, treniranje i evaluacija modela dubokog učenja u samo nekoliko linija koda. [18]

OpenCV

OpenCV je također biblioteka otvorenog koda namijenjena za strojno učenje i računalni vid. Ona omogućuje obradu slika i videozapisa tj. njihovo učitavanje, promjenu veličine i ostale zadaće, a u uz to i mnoge kompleksnije zadaće poput prepoznavanje lica, objekata i kontura na slikama i videozapisima. Rad s ovom bibliotekom je također vrlo jednostavan. [19]

Verzija Python programskog jezika korištena u ovome radu je 3.10.

Verzije prethodno navedenih biblioteka korištenih u sklopu ovoga rada su:

- TensorFlow/Keras 2.10.0,
- NumPy 1.23.4,
- Matplotlib 3.6.1,
- Pandas 1.5.1,
- Sckit Learn 1.1.2,
- OpenCV 4.6.0.

Na [Slika 10.] je prikazan način učitavanja prethodno navedenih biblioteka u Python programskom jeziku.

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras import Input
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
from keras.callbacks import EarlyStopping
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import load_model
from keras.optimizers import Adam, SGD, RMSprop, Adadelta, Adagrad, Adamax, Nadam, Ftrl

import pandas as pd

import numpy as np

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

import cv2

import matplotlib
import matplotlib.pyplot as plt

import sys
import os
```

Slika 10. Učitavanje biblioteka

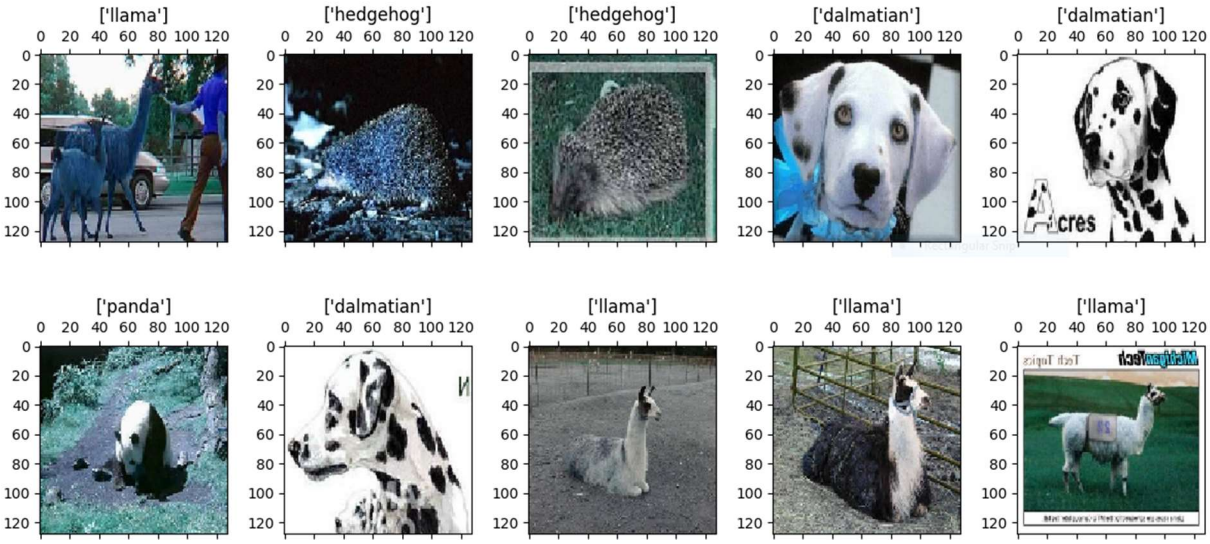
3.2. Podaci za treniranje

Podaci za treniranje tj. učenje su preuzeti sa stranice „CaltechDATA“ [20] i sadrže veliku količinu različitih slika od kojih su odabrane četiri kategorije životinja:

- dalmatiner (engl. dalmatian),
- jež (engl. hedgehog),
- ljama (engl. llama) i
- panda (engl. panda).

Ukupan broj slika iz navedenih kategorija iznosi 237 što je jako malo i tu činjenicu treba uzeti u obzir prilikom vrednovanja modela. Navedeni ukupan broj slika će se još podijeliti na skup za treniranje i skup za testiranje o čemu će biti više rečeno kasnije.

Na [Slika 11.] je prikazano nekoliko primjera podataka za treniranje.



Slika 11. Podaci za treniranje

Kako bi mogli koristiti preuzete podatke prvo unutar Python programskog jezika moramo definirati lokaciju na kojoj se nalaze podaci za treniranje. Nakon definiranja lokacije potrebno je specificirati koje podatke želimo koristiti kako bi ih mogli učitati. Učitani podaci tj. slike moraju biti jednakih dimenzija pa im je potrebno promijeniti veličinu. Odabrane dimenzije slika su 128 x 128 piksela. Učitavanje slika i njihova prilagodba je izvedena pomoću funkcija iz ranije spomenute OpenCV biblioteke. Učitane i prilagođene slike potrebno je pretvoriti u polja podataka standardiziranih vrijednosti kako bi ostali dijelovi programa mogli raditi s njima za što su nam poslužile funkcije iz NumPy biblioteke.

Na [Slika 12.] su prikazani prethodno opisani koraci za učitavanje i prilagodbu podataka unutar Python- a.

```
ImgLocation = main_dir + '\\pythonProject\\caltech-101\\101_ObjectCategories\\'

LABELS = set(["dalmatian", "hedgehog", "llama", "panda"])

ImagePaths = []
ListLabels = []
for label in LABELS:
    for image in list(os.listdir(ImgLocation + label)):
        ImagePaths = ImagePaths + [ImgLocation + label + "/" + image]
        ListLabels = ListLabels + [label]

data = []
for img in ImagePaths:
    image = cv2.imread(img)
    image = cv2.resize(image, (128, 128))
    data.append(image)

data = np.array(data, dtype="float") / 255.0
```

Slika 12. Učitavanje i prilagođavanje podataka za treniranje

Kao što je to već ranije spomenuto učitane podatke je potrebno podijeliti na skup za treniranje (engl. training data) i skup za testiranje (engl. test data), to izvodimo pomoću naredbe prikazane na [Slika 13.] gdje argument „test_size“ = 0.2 označava da podaci za testiranje čine 20% od ukupnog broja slika. Na kraju imamo 189 slika unutar skupa za trening i 48 slika unutar skupa za testiranje što je jako malo.

```
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.2, random_state=0)
```

Slika 13. Podjela podataka na skupove za treniranje i testiranje

Konačan oblik podataka je prikazan na [Slika 14.] gdje vrijednosti u zagradama označavaju redom broj primjera, broj redaka, broj stupaca i broj kanala. Redci i stupci se tu pojavljuju kao posljedica zapisivanja podataka u obliku polja i oni predstavljaju dimenzije slika, dok broj kanala predstavlja boje koje se koriste tj. RGB sustav u ovome slučaju.

```
Shape of whole data:  (237, 128, 128, 3)
Shape of X_train:  (189, 128, 128, 3)
Shape of y_train:  (189, 1)
Shape of X_test:  (48, 128, 128, 3)
Shape of y_test:  (48, 1)
```

Slika 14. Oblik podataka

3.3. Izrada modela duboke konvolucijske neuronske mreže

Kod izrade modela prvo krećemo od definiranja arhitekture modela. U poglavlju 2.6. su detaljno opisani svi slojevi koji se koriste pored standardnih ulaznih i izlaznih slojeva kod konvolucijskih neuronskih mreža. Pored ranije spomenutih slojeva potrebno je još naglasiti sloj ispadanja (engl. dropout layer) koji sprječava pojavu efekta „pretreniranja“ modela, a koji će se također koristiti. Arhitektura modela duboke konvolucijske neuronske mreže koja je korištena u sklopu ovoga zadatka je sljedeća:

- ulazni sloj,
- konvolucijski sloj, sloj maksimalnog udruživanja i sloj ispadanja,
- konvolucijski sloj, sloj maksimalnog udruživanja i sloj ispadanja,
- konvolucijski sloj, sloj maksimalnog udruživanja i sloj ispadanja,
- sloj poravnanja,
- sloj gustoće kao sakriveni sloj i
- sloj gustoće kao izlazni sloj.

Ranije je spomenuto kako arhitekturu mreže tj. broj i vrstu pojedinih slojeva možemo proglasiti hiperapramterom. Ova arhitektura je odabrana iz razloga što ne sadrži pretjerano veliki broj različitih slojeva, a daje dobre rezultate uz ubrzani i olakšani postupak procesiranja. Uz ovakvu arhitekturu smanjena je i mogućnost pojave efekta „pretreniranja“ modela.

Prilikom definiranja arhitekture na raspolaganju je niz hiperparametara koje možemo podešavati kako bi dobili što kvalitetniji model. Kod konvolucijskog sloja (engl. convolutional layer) najvažniji hiperparametri su broj filtera (argument „filters“) koji se primjenjuje na sliku, vrsta aktivacijske funkcije (argument „activation“) i veličina kernela (argument „kernel_size“). Pored navedenih značajni hiperparametri su korak i ispuna (argumenti „strides“ i „padding“) koji su detaljno objašnjeni u poglavlju 2.6.1. Ukoliko je vrijednost argumenta „strides“ jednaka jedinici i ako je argument „padding“ definirana kao „same“ dimenzije izlaza i ulaza u konvolucijski sloj su jednake. Unutar sloja maksimalnog udruživanja (engl. Max Pooling layer) kao hiperparametar možemo definirati dimenzije sloja udruživanja tj. argument „pool_size“, a uz njega se ponovno pojavljuju već spomenuti argumenti „strides“ i „padding“. Kod sloja ispadanja je potrebno definirati postotak ulaznih jedinica koje će poprimiti vrijednost nulu i neće utjecati na određivanje težinskih faktora modela, navedeni postotak se također može prozvati hiperparametrom. U sklopu ovog rada taj postotak će iznositi 20 %. Sloj gustoće (engl. dense layer) definiraju dva hiperparametra, a to su veličina sloja i aktivacijska funkcija

(argument „activation“). Bitno je napomenuti kako je veličina izlaznog sloja gustoće definirana ukupnim brojem kategorija učitanog skupa podataka tj. četiri u sklopu ovog modela. Također je bitno napomenuti da je model definiran kao „Sequential“, ovakva vrsta modela se koristi kod jednostavnih modela koji sadrže slojeve s jednim ulazom i izlazom.

Na [Slika 13.] je prikazan način na koji se definira arhitektura i prethodno navedeni hiperparametri duboke konvolucijske neuronske mreže.

```
model = Sequential(name="DCN-Model")

model.add(Input(shape=(X_train.shape[1],X_train.shape[2],X_train.shape[3]), name='Input-Layer'))

model.add(Conv2D(filters=filter, kernel_size=(3,3), strides=(1,1), padding='valid', activation='relu', name='2D-Convolutional-Layer-1'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2), padding='valid', data_format=None, name='2D-MaxPool-Layer-1'))
model.add(Dropout(0.2, name='Dropout-Layer-1'))

model.add(Conv2D(filters=filter*2, kernel_size=(3,3), strides=(1,1), padding='valid', activation='relu', name='2D-Convolutional-Layer-2'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2), padding='valid', name='2D-MaxPool-Layer-2'))
model.add(Dropout(0.2, name='Dropout-Layer-2'))

model.add(Conv2D(filters=filter*4, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu', name='2D-Convolutional-Layer-3'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2), padding='same', name='2D-MaxPool-Layer-3'))
model.add(Dropout(0.2, name='Dropout-Layer-3'))

model.add(Flatten(name='Flatten-Layer'))
model.add(Dense(16, activation=activation, name='Hidden-Layer-1', kernel_initializer='HeNormal'))
model.add(Dense(4, activation='softmax', name='Output-Layer'))
```

Slika 13. Arhitektura modela

Nakon što je arhitektura mreže definirana potrebno je kompajlirati model. Prilikom kompajliranja modela definira se algoritam unatražne propagacije izlazne pogreške, jedni od najpoznatijih algoritama su Adam, Adagrad, Adamax, Adadelta, RMSprop i Nadam. Navedeni algoritmi se razlikuju u odnosu na pravilo (engl. update rule) prema kojem prilagođavaju težinske faktore nakon pojedine epohe. Velika prednost upotrebe ovih algoritama je u tome što oni koriste adaptivnu tj. promjenjivu vrijednost stope učenja tokom treninga, a to smo naveli ranije kao vrlo koristan alata pri pronalasku globalnog minimuma funkcije pogreške. Unutar navedenih algoritama definira se najvažniji hiperparametar tj. stopa učenja (argument „learning_rate“) koja predstavlja početnu vrijednost pošto navedeni algoritmi primjenjuju adaptivnu stopu učenja. Kod kompajliranja je potrebno definirati i vrstu funkcije pogreške (argument „loss“) koju želimo koristiti tj. način izračuna pogreške modela. Još jedan parametar koji se definira kod kompajliranja modela je metrika tj. mjerna jedinica (argument „metrics“) koja će se koristiti za vrednovanje modela, no njezina vrijednost ne utječe na postupak

treniranja modela za razliku od vrijednosti pogreške modela. Najčešće se kao mjerna jedinica koristi točnost (engl. accuracy).

U nastavku na [Slika 14.] je prikazan kod pomoću kojeg se kompajlira model i definiraju spomenuti parametri.

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
              loss='SparseCategoricalCrossentropy',
              metrics=['accuracy'],
              loss_weights=None,
              weighted_metrics=None,
              run_eagerly=None,
              steps_per_execution=None
              )
```

Slika 14. Kompajliranje modela

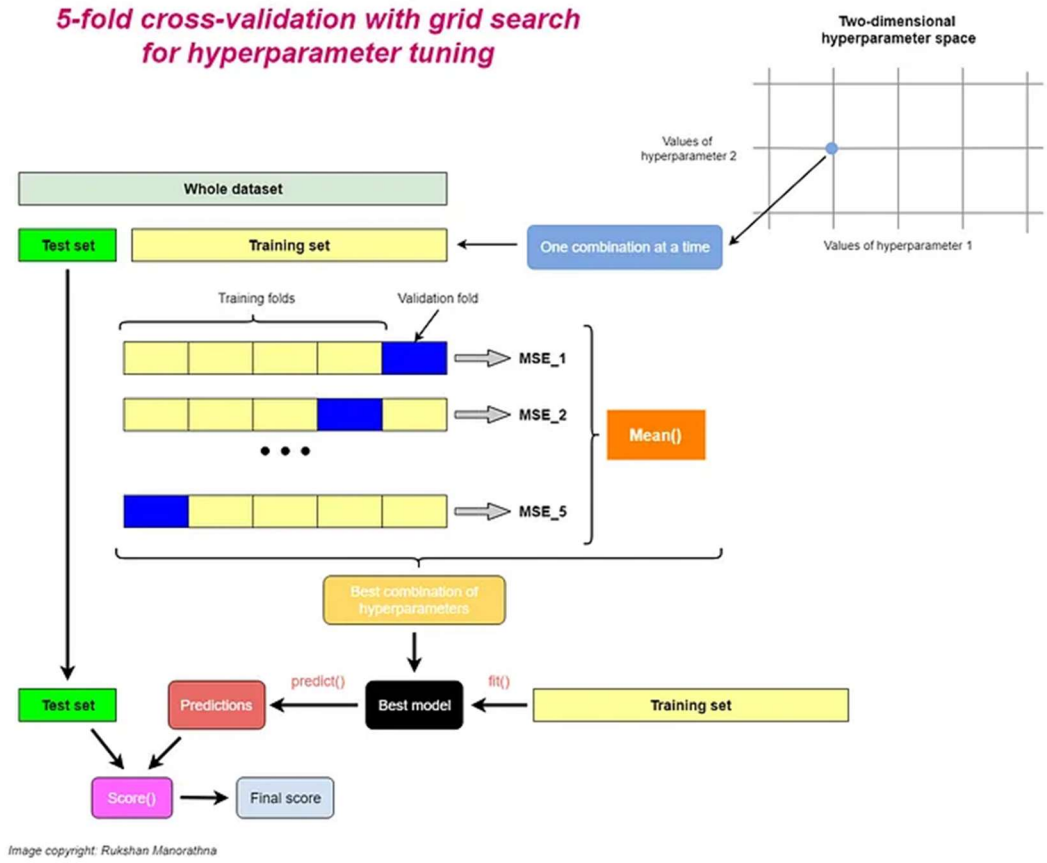
3.4. Algoritmi za podešavanje hiperparametara

Za određivanje optimalnih hiperparametara je potrebno znanje i iskustvo, algoritmi su samo alat kako bi se brže došlo do rješenja problema. Algoritmi za podešavanje hiperparametara isprobavaju određene kombinacije hiperparametara i odabiru onu za koju je pogreška modela najmanja. Kako bi algoritam bio efikasan, njegovi ulazni podaci tj. različite vrijednosti hiperparametara moraju biti kvalitetno zadane, upravo tu do izražaja dolazi znanje i iskustvo osobe koja se bavi tim poslom. U sklopu ovoga rada će se implementirati dvije vrste algoritama za podešavanje hiperparametara tj. grid search i random search algoritmi.

3.4.1. Grid Search

Naziv grid search proizlazi iz načina definiranja pojedinih kombinacija hiperparametara. Ako želimo na primjer naći idealnu kombinaciju različitih vrijednosti dva različita hiperparametra to možemo grafički prikazati mrežom (engl. grid) vodoravnih i okomitih linija gdje vodoravne linije predstavljaju različite vrijednosti jednog hiperparametra, a okomite linije različite vrijednosti drugog hiperparametra. Sjecišta vodoravnih i okomitih linija predstavljaju moguće kombinacije dva hiperparametra. Iz čega se može zaključiti kako grid search algoritam isprobava sve moguće kombinacije zadanih vrijednosti hiperparametara. U sklopu grid search algoritma se može pronaći još jedan iznimno koristan algoritam za pronalaženje hiperparametara, a to je k – fold algoritam. K – fold algoritam je algoritam unakrsne provjere (engl. cross – validation) koji zadani skup podataka za trening dijeli na k skupina od kojih k – 1 skupina predstavlja skup podatak za trening, a ostala jedna skupina predstavlja skup podataka za provjeru na temelju koje se procjenjuje model. Taj postupak se ponavlja k puta. Konačna procjena modela dobije se kao srednja vrijednost procjene svake skupine. [21]

Prethodni postupak je grafički prikazan na [Slika 15.]. [21]



Slika 15. Grid search algoritam

Na temelju opisanog postupka i prethodne slike može se zaključiti da će ukupan broj iteracija biti jednak umnošku broja mogućih kombinacija zadanih hiperparametara i broja skupina na koje smo podijelili zadani skup podataka za trening prilikom primjene k – fold algoritma. Taj broj će biti dosta velik pa će proces trajati duže.

Implementacija grid search algoritma u programski kod je dosta jednostavna, sastoji se od definiranja mreže željenih hiperparametara čije kombinacije želimo ispitati (argument „hyper_param“) i definiranja parametara samog algoritma. Parametri algoritma su model nad kojim želimo ispitati pojedine kombinacije hiperparametara (argument „estimator“), mreža prethodno definiranih parametara (argument „param_grid“), mjerna jedinica na temelju koje se procjenjuje model prilikom unakrsne provjere (argument „scoring“), broj skupina na koji se dijeli skup podataka za trening kod k – fold algoritma (argument „cv“), broj zadaća koje se mogu istovremeno izvršavati tj. argument „n_jobs“ i na kraju argument „refit“ koji je definiran kao „True“ što znači da se u prethodno definirani model prilagođava uzimajući u obzir najoptimalnije hiperparametre. [22]

Na [Slika 16.] je prikazana implementacija grid search algoritma u programski kod.

```
batches = [16, 32]
epochs = [40, 70]
learning_r = np.arange(0.001, 0.005, 0.001)
filters = [16, 32]
activation_fun = ['sigmoid', 'relu']

hyper_param = dict(epochs=epochs, batch_size=batches, learning_rate=learning_r, filter=filters, activation=activation_fun)

grid = GridSearchCV(estimator=model_grid_search, param_grid=hyper_param, scoring='accuracy', cv=3, refit=True, n_jobs=-1)
```

Slika 16. Implementacija grid search algoritma

3.4.2. Random Search

Random search algoritam kao i grid search isprobava različite kombinacije hiperparametara čije vrijednosti mi zadajemo. Glavna razlika u odnosu na grid search algoritam je u tome što random search algoritam ne isprobava sve moguće kombinacije vrijednosti zadanih hiperparametara nego odabire nasumične kombinacije hiperparametara u onoliko iteracija koliko mi zadamo. Random search algoritam također sadrži k – fold algoritam koji smo objasnili kod grid search algoritma. Možemo zaključiti kako je ukupan broj iteracija jednak umnošku broja iteracija koji smo definirali i broja skupina na koje smo podijelili zadani skup podataka za trening prilikom primjene k -fold algoritma.

Implementacija random search algoritma se također sastoji od definiranja vrijednosti različitih hiperparametara modela (argument „hyper_param“) i definiranja parametara algoritma. Parametri algoritma su isti kao kod grid search algoritma, razlika je u tome što kod random search algoritma moramo definirati ranije spomenuti broj iteracija (argument „n_iter“) i argument koji definira zadane hiperparametre se naziva „param_distributions“. [23]

Na [Slika 17.] je prikazana implementacija random search algoritma u programski kod.

```
batches = [16, 32]
epochs = [40, 70]
learning_r = np.arange(0.001, 0.005, 0.001)
filters = [16, 32]
activation_fun = ['sigmoid', 'exponential']

hyper_param = dict(epochs=epochs, batch_size=batches, learning_rate=learning_r, filter=filters, activation=activation_fun)

random_rf = RandomizedSearchCV(estimator=model_random_search, param_distributions=hyper_param, n_iter=10, scoring='accuracy', cv=5, refit=True, n_jobs=-1)
```

Slika 17. Implementacija random search algoritma

3.5. Treniranje modela

Treniranje tj. učenje modela je najvažniji dio prilikom izrade modela. Kako je to ranije spomenuto, treniranje je iterativni postupak prilagođavanja težinskih faktora modela. Prije početka treniranja potrebno je definirati optimalne hiperparametre budući da rezultat treniranja modela bitno ovisi o njima. Uz prethodno navedene hiperparametre kod postupka treniranja potrebno je definirati još dva vrlo važna hiperparametra koji su izravno povezani s procesom treniranja modela, a to su broj epoha (engl. epoch) i veličina grupe kod mini – grupnog gradijenta pada (engl. mini - batch size). Važnost tih hiperparametara je opisana u poglavljima 2.7.2. i 2.7.3.

Definiranje postupka treniranja modela unutar programskog koda je također vrlo jednostavno. Sastoji se od definiranja skupa podataka za trening i callback – ova. Callback je dio programskog koda koji se može pozvati u bilo kojoj fazi izvođenja nekog drugog programskog koda tj. na početku, u tijeku samog izvođenja ili na kraju programa. Callback – ovi korišteni u ovome radu će biti kasnije opisani.

Način na koji se definira postupak treniranja modela unutar programskog koda je prikazan na [Slika 18.].

```
grid_result = grid.fit(X_train, y_train, callbacks=[early_stop_cb, early_stop_mc])
```

Slika 18. Treniranje modela

3.5.1. EarlyStopping Callback

EarlyStopping callback koristi se u svrhu zaustavljanja postupka treniranja modela u slučaju kada se mjerena veličina ne poboljšava nakon definiranog broja epoha.

Najbitniji parametri koji se moraju definirati prilikom implementacije spomenutog callback – a u programski kod su veličina koja se mjeri (argument „monitor“) koja će unutar ovoga zadatka biti gubitak (engl. loss), broj epoha nakon kojih se zaustavlja postupak treniranja ukoliko ne dolazi do poboljšanja mjerene veličine (argument „patience“) i težinski faktori koji se dodjeljuju modelu nakon prekida postupka definiranja tj. argument „restore_best_weights“ koji u slučaju kada je definiran kao „True“ dodjeljuje modelu težinske faktore iz one epohe kod koje je model pokazao najbolje vrijednosti mjerene veličine, a kada je definiran kao „False“ dodjeljuje modelu težinske faktore iz zadnje epohe prije prekida izvođenja. [48]

Early stopping callback implementiran u programski kod je prikazan na [Slika 19.].

```
early_stop_cb = tf.keras.callbacks.EarlyStopping(
    monitor="loss",
    min_delta=0,
    patience=10,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=True
)
```

Slika 19. EarlyStopping callback

3.5.2. ModelCheckpoint Callback

ModelCheckpoint callback se koristi u svrhu spremanja kompletnog model zajedno s težinskim faktorima ili spremanje samo težinskih faktora koji su pokazali najbolju vrijednost mjerene veličine. Model ili samo njegovi pripadajući težinski faktori se spremaju u datoteku formata „h5“ ili „json“.

Definiranje callback – a unutar programskog koda je također vrlo jednostavno. Parametri koji ga definiraju su putanja do datoteke u koju se spremaju podaci, mjerena veličina (argument „monitor“), argument „save_best_only“ koji za vrijednost „True“ sprema samo najbolji model, argument „save_weights_only“ kod kojeg definiramo da li želimo spremati samo težine ili cijeli model i parametar kod kojeg definiramo na kojoj razini želimo spremati podatke (argument „save_freq“). [49]

ModelCheckpoint callback implementiran u programski kod je prikazan na [Slika 20.].

```
early_stop_mc = keras.callbacks.ModelCheckpoint(
    "model.h5",
    monitor="loss",
    save_best_only=True,
    save_weights_only=False,
    save_freq="epoch"
)
```

Slika 20. ModelCheckpoint callback

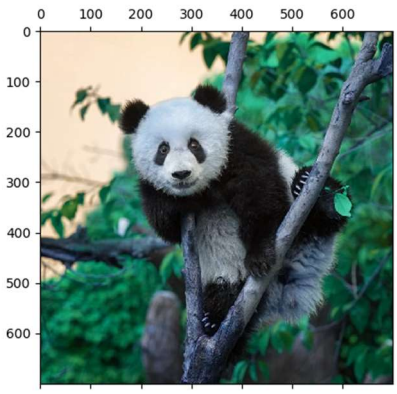
3.6. Predviđanje modela

Trenirani model možemo koristiti u svrhu predviđanja. Prije samog predviđanja potrebno je učitati prethodno spremljeni model s optimalnim težinskim faktorima. Postupak učitavanja modela iz datoteke formata „h5“ je prikazan na [Slika 21.].

```
ucitani_model = load_model('model.h5')
```

Slika 21. Učitavanje modela

Pored učitavanja modela potrebno je učitati i sliku koja će poslužiti za predviđanje. Učitanu sliku je potrebno prilagoditi na isti način na koji su prilagođene slike tj. podaci za treniranje, a koji je opisan u poglavlju 3.2. Na [Slika 22.] je prikazana slika koja će koristiti za postupak predviđanja, a na [Slika 23.] je prikazan postupak učitavanja i prilagodbe spomenute slike.



Slika 22. Slika korištena za postupak predviđanja

```
prediction_img = cv2.imread(main_dir+'\pythonProject\data\panda.JPG')

prediction_img = cv2.resize(mydog, (128, 128))

prediction_img = prediction_img / 255.0

prediction_img = prediction_img[np.newaxis, ...]
```

Slika 23. Učitavanje i prilagodba slike za postupak predviđanja

Oblik slike koja će se koristiti za predviđanje je prikazan na [Slika 24.] iz njega se može saznati redom kako se radi o jednom primjeru, dimenzija 128 x 128 piksela u boji (RGB) što odgovara obliku podataka za treniranje.

```
Shape of the input:  (1, 128, 128, 3)
```

Slika 24. Oblik slike za postupak predviđanja

Na kraju je potrebno izvršiti predviđanje učitane i prilagođene slike te prilagoditi izlazni oblik podataka kako bi se ispisivalo ime životinje koja se nalazi na slici umjesto vjerovatnosti koja je standardni izlazni podatak iz modela predviđanja. Izvedba prethodno navedenih postupaka u programskom kodu je prikazana na [Slika 25.].

```
prediction = enc.inverse_transform(np.array(tf.math.argmax(ucitani_model.predict(prediction_img),axis=1)).reshape(-1, 1))
```

Slika 25. Predviđanje modela i prilagodba izlaznih podataka

U svrhu vrednovanja modela postupak predviđanja će se provesti i nad skupovima podataka za treniranje i testiranje modela. Izlazni oblik podataka je također potrebno prilagoditi kao i u prethodnom slučaju. Na [Slika 26.] je prikazan programski kod pomoću kojeg se izvršava spomenuto predviđanje i prilagođava izlazni oblik podataka.

```
pred_labels_tr = np.array(tf.math.argmax(ucitani_model.predict(X_train), axis=1))
pred_labels_te = np.array(tf.math.argmax(ucitani_model.predict(X_test), axis=1))
```

Slika 26. Predviđanje modela nad skupovima podataka za treniranje i testiranje

4. VREDNOVANJE MODELA

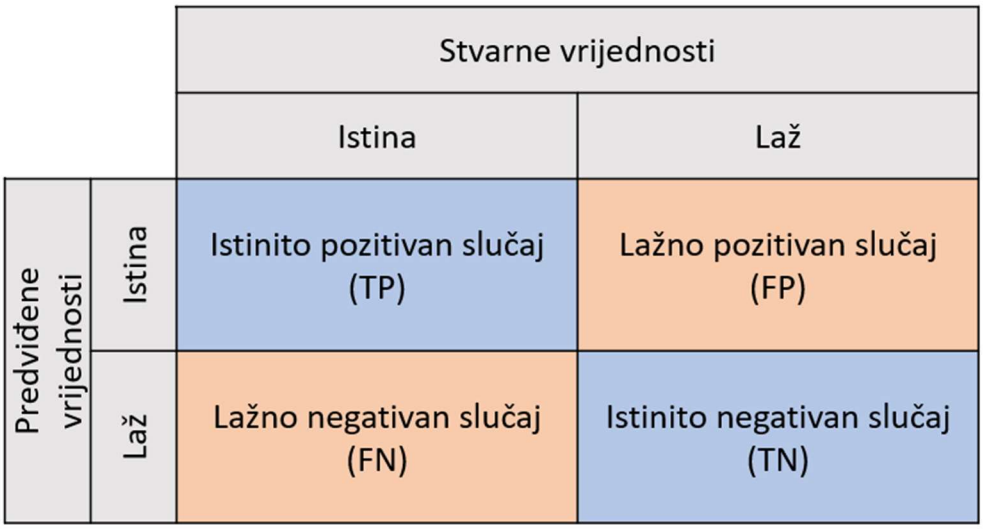
4.1. Parametri kod vrednovanja modela

Kod vrednovanja modela koristi se više različitih parametara, a svi se definiraju na temelju matrice zabune (engl. confusion matrix). Matrica zabune povezuje stvarne vrijednosti i modelom previđene vrijednosti. Najjednostavnija matrica zabune obuhvaća dva primjera tj. istina i laž. Takva matrica je dimenzija 2 x 2 i sastoji se od četiri slučaja:

- TP – istinito pozitivan slučaj (engl. true positive),
- FP – lažno pozitivan slučaj (engl. false positive),
- FN – lažno negativan slučaj (engl. false negative) i
- TN – istinito negativan slučaj (engl. true negative).

Istinito pozitivan slučaj (TP) se dešava kada su stvarna i predviđena vrijednost istina, u slučaju kada je stvarna vrijednost istina, a predviđena laž radi se o lažno negativnom slučaju (FN). Istinito negativan slučaj (TN) se dešava kada su stvarna i predviđena vrijednost laž, u slučaju kada je stvarna vrijednost laž, a predviđena istina radi se o lažno pozitivnom slučaju (FP).

Primjer prethodno opisane matrice zabune je prikazan na [Slika 27.]. [26]



Slika 27. Matrica zabune

Na temelju prethodne slike može se zaključiti kako se na dijagonali matrice nalaze ispravno predviđeni primjeri, dok sve ostale vrijednosti predstavljaju krivo predviđene primjere.

4.1.1. Točnost

Točnost (engl. accuracy) predstavlja omjer između broja ispravno predviđenih primjera i ukupnog broja primjera. Izraz za računanje točnosti prema [26] glasi:

$$A_{CC} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{4.1}$$

Potrebno je napomenuti kako točnost nije najbolji parametar za validaciju modela koji sadrži skup podataka koji nije uravnotežen.

4.1.2. Preciznost

Izraz za računanje preciznosti (engl. precision) prema [26] glasi:

$$P = \frac{TP}{TP + FP}. \tag{4.2}$$

Idealna vrijednost preciznosti je u slučaju kada je broj lažno negativnih slučajeva *FP* jednak nuli. Porastom vrijednosti lažno negativnih slučajeva *FP* smanjuje se vrijednost preciznosti *P*.

4.1.3. Odziv

Odziv (engl. recall) je predstavljen omjerom broja istinito pozitivnih slučajeva i ukupnog broja pozitivnih primjera. Izraz za računanje odziva prema [26] glasi:

$$R = \frac{TP}{TP + FN}. \tag{4.3}$$

Iz prethodnog izraza se vidi kako će se vrijednost odziva *R* smanjivati povećanjem lažno negativnih slučajeva *FN*.

4.1.4. F1 – mjera

F1 – mjera (engl. f1 – score) predstavlja harmonijsku sredinu između vrijednosti preciznosti i odziva. Izraz za računanje f1 – mjere prema [26] glasi:

$$F = 2 \cdot \frac{P \cdot R}{P + R}. \tag{4.4}$$

F1 – mjera je predstavlja bolji parametar za vrednovanje modela u odnosu na točnost iz razloga što objedinjava preciznost i odziv tj. da bi F1 – mjera pružala dobre rezultate preciznost i odziv moraju pružati dobre rezultate.

Bitno je za napomenuti kako se prethodni parametri računaju posebno za svaku kategoriju podataka unutar zadanog skupa podataka. Vrijednost navedenih parametara za cjelokupni skup podataka dobiva se kao srednja vrijednost parametara po pojedinim kategorijama. Razlikuju se „macro – avg“, „micro – avg“ i „weighted – avg“ načini izračuna srednje vrijednosti.

4.2. Odabir optimalnih hiperparametara

Kao što je to već ranije spomenuto određivanje optimalnih hiperparametara je iterativni postupak i tu nam uvelike pomažu grid search i random search algoritmi koji su detaljno objašnjeni u poglavljima 3.4.1. i 3.4.2. U sklopu ovoga poglavlja pokazati će se nekoliko primjera u kojima su navedeni algoritmi korišteni nad ranije opisanom dubokom konvolucijskom neuronskom mrežom.. Vrednovanje modela se temelji na parametrima definiranim u prethodnom poglavlju. Slika koja će se koristiti za predviđanje je prikazana u poglavlju 3.7. [Slika 22.] na kojoj se nalazi panda.

Odabrani hiperparametri koji će se optimirati su:

- broj epoha (argument „epochs“),
- veličina grupe kod mini – grupnog gradijenta pada (argument „batch_size“),
- stopa učenja (argument „learning_rate“) i
- broj filtera kod konvolucijskog sloja (argument „filters“).

4.2.1. Primjer 1.

U prvom primjeru će se koristiti grid search algoritam. Zadani skup različitih vrijednosti hiperparametara i parametri grid search algoritma su prikazani na [Slika 28.].

```
batches = [16, 32]
epochs = [40, 70]
learning_r = np.arange(0.001, 0.005, 0.001)
filters = [16, 32]

hyper_param = dict(epochs=epochs, batch_size=batches, learning_rate=learning_r, filter=filters)

grid = GridSearchCV(estimator=model_grid_search, param_grid=hyper_param, cv=3, scoring='f1_macro', refit=True)
```

Slika 28. Primjer 1. – Zadani skup hiperparametara

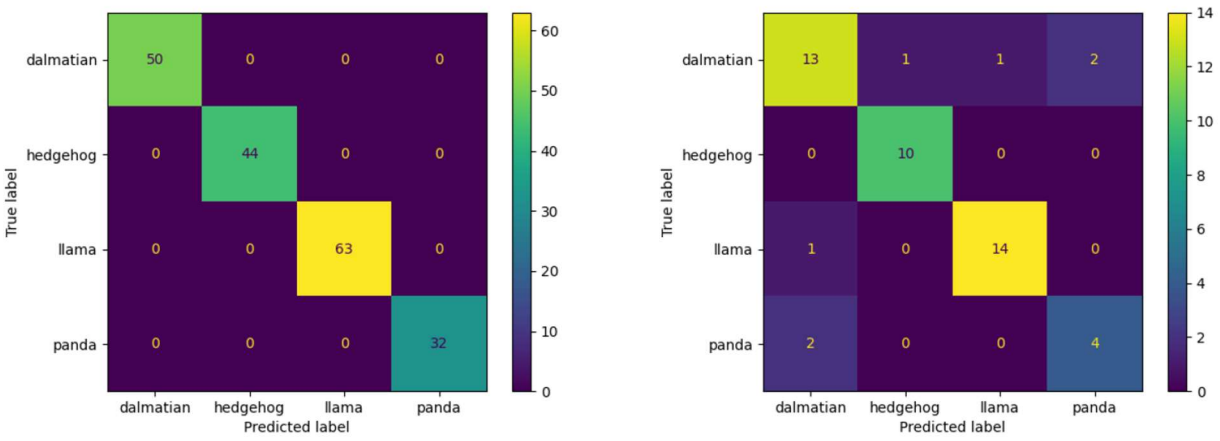
Na temelju zadanog skupa hiperparametara i zadanog broja unakrsnih provjera će se provesti ukupno 120 iteracija.

Rezultat tj. optimalni hiperparametri su prikazani na [Slika 29.].

```
Best hyperparameters are: {'batch_size': 16, 'epochs': 40, 'filter': 16, 'learning_rate': 0.001}
```

Slika 29. Primjer 1. – Optimalni hiperparametri

Na [Slika 30.] lijevo je prikazana matrica zabune nad skupom podataka za treniranje, a desno nad skupom podataka za testiranje.



Slika 30. Primjer 1. – Matrice zabune

Parametri za vrednovanje modela koji proizlaze i matrica zabune su prikazani na [Slika 31.].

```
------------------ Evaluation on Training Data ------------------

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        50
         1.0       1.00      1.00      1.00        44
         2.0       1.00      1.00      1.00        63
         3.0       1.00      1.00      1.00        32

    accuracy                           1.00       189
   macro avg       1.00      1.00      1.00       189
weighted avg       1.00      1.00      1.00       189


------------------- Evaluation on Test Data --------------------
              precision    recall  f1-score   support

         0.0       0.81      0.76      0.79        17
         1.0       0.91      1.00      0.95        10
         2.0       0.93      0.93      0.93        15
         3.0       0.67      0.67      0.67         6

    accuracy                           0.85        48
   macro avg       0.83      0.84      0.84        48
weighted avg       0.85      0.85      0.85        48
```

Slika 31. Primjer 1. – Vrednovanje modela

Prema vrijednostima parametara prikazanih na prethodnoj slici može se zaključiti da model nad skupom podataka za treniranje daje odlične rezultate, dok nad skupom podatak za testiranje daje malo lošije rezultate, a to se definira kao efekt „pretreniranja“ modela koji je ranije opisan

u poglavlju 2.7.2. U ovome slučaju razlog zbog kojeg dolazi do tog efekta pored prevelikog broja epoha može biti i manja količina podataka koja se koristi prilikom treniranja ove mreže.

Na [Slika 32.] je prikazan rezultat predviđanja nad slikom koju smo učitali.

```
1/1 [==============================] - 0s 28ms/step
DCN model prediction:  [['panda']]
1/1 [==============================] - 0s 32ms/step

Probabilities for each category:
dalmatian  :  0.24200304
hedgehog  :  0.033601075
llama  :  0.28688258
panda  :  0.4375133
```

Slika 32. Primjer 1. – Rezultat predviđanja modela

Prema prethodnoj slici model je uspio točno predvidjeti, no ne sa baš pretjerano velikom sigurnošću što je također posljedica male količine podataka koja se koristi za treniranje modela.

4.2.2. Primjer 2.

Kod ovog primjera se koristio random search algoritam. Na [Slika 33.] je prikazan skup zadanih hiperparametara kao i parametri random search algoritma.

```
batches = [16, 32, 64]
epochs = [10, 20, 40, 70]
learning_r = [0.001, 0.01, 0.1]
filters = [16, 32, 64]

hyper_param = dict(epochs=epochs, batch_size=batches, learning_rate=learning_r, filter=filters)

random_rf = RandomizedSearchCV(estimator=model_random_search, param_distributions=hyper_param, n_iter=20, scoring='f1_macro', cv=5, refit=True)
```

Slika 33. Primjer 2. – Zadani skup hiperparametara

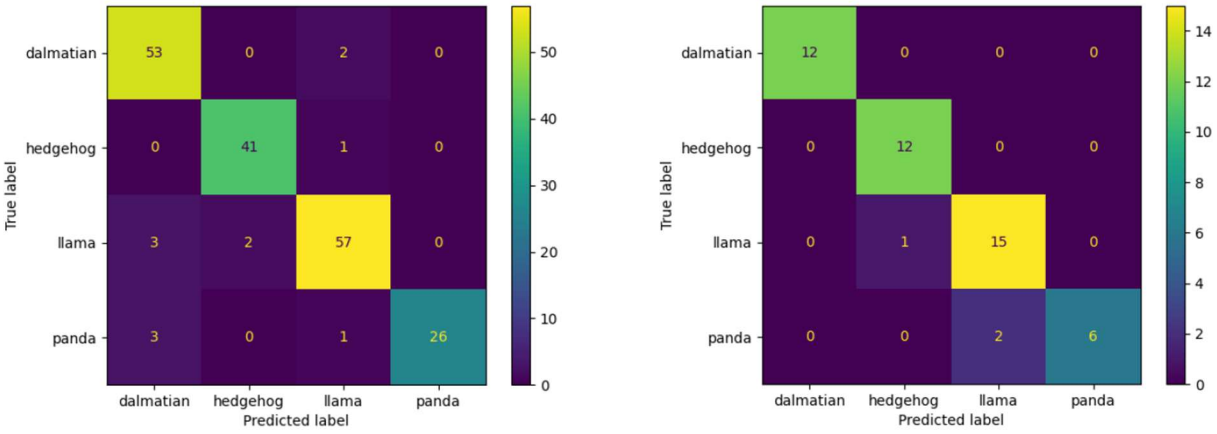
Ukupan broj iteracija kod ovog primjera obzirom na broj zadanih iteracija i zadani broj unakrsnih provjera će iznositi 100 iteracija. Primjenom ovog algoritma možemo zadati veći broj vrijednosti različitih parametara budući da broj iteracija ne ovisi o njima, za razliku od grid search algoritma koji bi u ovome slučaju imao 540 iteracija što bi zahtijevalo dosta vremena za izvršavanje.

Odabrani hiperparametri od strane algoritma su prikazani na [Slika 34.].

```
Best hyperparameters are: {'learning_rate': 0.001, 'filter': 32, 'epochs': 70, 'batch_size': 16}
```

Slika 34. Primjer 2. – Optimalni hiperparametri

Na [Slika 35.] su prikazane matrice zabune, lijeva matrice je matrica zabune nad skupom podataka za treniranje, dok desna matrica predstavlja matricu zabune nad skupom podataka za testiranje.



Slika 35. Primjer 2. – Matrice zabune

Vrednovanje modela je prikazano na [Slika 36.].

```
------------------ Evaluation on Training Data ------------------

              precision    recall  f1-score   support

         0.0       0.90      0.96      0.93        55
         1.0       0.95      0.98      0.96        42
         2.0       0.93      0.92      0.93        62
         3.0       1.00      0.87      0.93        30

    accuracy                           0.94       189
   macro avg       0.95      0.93      0.94       189
weighted avg       0.94      0.94      0.94       189


-------------------- Evaluation on Test Data --------------------
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        12
         1.0       0.92      1.00      0.96        12
         2.0       0.88      0.94      0.91        16
         3.0       1.00      0.75      0.86         8

    accuracy                           0.94       48
   macro avg       0.95      0.92      0.93       48
weighted avg       0.94      0.94      0.94       48
```

Slika 36. Primjer 2. – Vrednovanje modela

Na temelju matrica zabune i parametara se prethodne slike može se zaključiti kako se ovdje radi o kvalitetnom modelu. U ovome slučaju ne dolazi do pojave efekta „pretreniranja“ modela kao u prethodnom primjeru.

Rezultat predviđanja modela nad učitanom slikom pande prikazan je na [Slika 37.].

```
1/1 [==============================] - 0s 32ms/step
DCN model prediction:  [['dalmatian']]
1/1 [==============================] - 0s 38ms/step

Probabilities for each category:
dalmatian  :  0.45296475
hedgehog  :  0.01749828
llama  :  0.3509102
panda  :  0.17862676
```

Slika 37. Primjer 2. – Rezultat predviđanja modela

Unatoč vrlo dobrim rezultatima koje je model pokazao tijekom predviđanja nad podacima iz skupa za treniranje i testiranje učitanu sliku nije uspio točno predvidjeti. Razlog tome je već prethodno navedena mala količina podataka koja se koristi prilikom treniranja modela.

4.2.3. Primjer 3.

U sklop ovoga primjera pokazati će rezultati modela za vrlo loše zadane vrijednosti hiperparametara i loše parametre random search algoritma koji su prikazani na [Slika 38.].

```
batches = [64, 128]
epochs = [60, 80]
learning_r = np.arange(0.05, 0.1, 0.01)
filters = [32, 64]

hyper_param = dict(epochs=epochs, batch_size=batches, learning_rate=learning_r, filter=filters)

random_rf = RandomizedSearchCV(estimator=model_random_search, param_distributions=hyper_param, n_iter=5, scoring='f1_macro', cv=3, refit=True)
```

Slika 38. Primjer 3. – Zadani skup hiperparametara

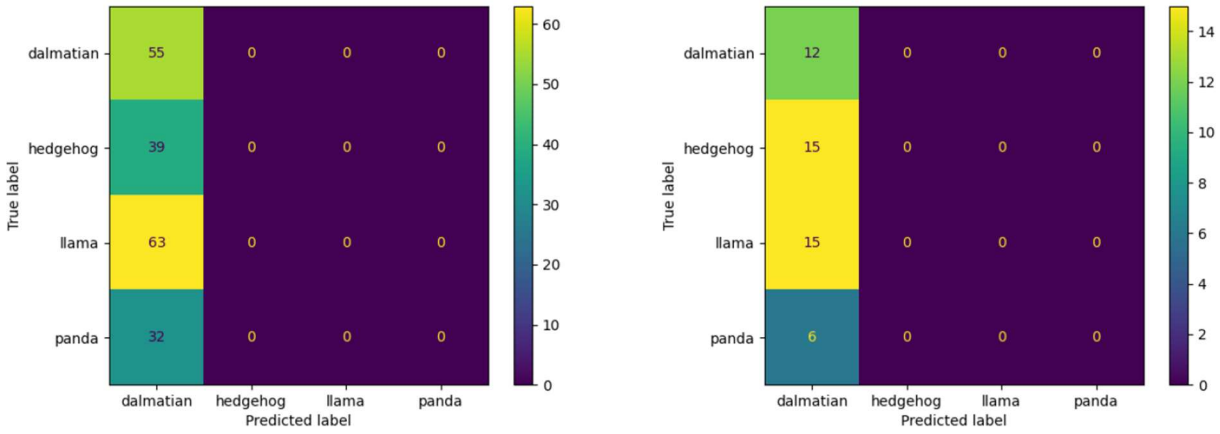
Iz zadanih parametara se može vidjeti kako se ovdje radi o velikoj stopi učenja u odnosu na prethodne primjere, a uz to je povećan broj filtera koji se primjenjuju unutar konvolucijskog sloja i povećane su veličine grupa kod mini – grupnog gradijenta pada. Ukupan broj iteracija unutar random search algoritma je smanjen na 15 iteracija.

Optimalni hiperparametri su prikazani na [Slika 39.].

```
Best hyperparameters are: {'learning_rate': 0.06, 'filter': 64, 'epochs': 80, 'batch_size': 128}
```

Slika 39. Primjer 3. – Optimalni hiperparametri

Na [Slika 40.] su prikazane matrice zabune, lijeva matrica je rezultat predviđanja nad skupom podataka za treniranje, dok je desna matrica rezultat predviđanja nad skupom podataka za testiranje.



Slika 40. Primjer 3. – Matrice zabune

Na temelju samih matrica se vidi kako ne možemo očekivati dobre rezultate jer model u svim slučajevima predviđa kako se na slikama nalazi dalmatiner.

Vrednovanje modela na temelju prethodnih matrica je prikazano na [Slika 41.].

	precision	recall	f1-score	support
0.0	0.29	1.00	0.45	55
1.0	0.00	0.00	0.00	39
2.0	0.00	0.00	0.00	63
3.0	0.00	0.00	0.00	32
accuracy			0.29	189
macro avg	0.07	0.25	0.11	189
weighted avg	0.08	0.29	0.13	189

---------- Evaluation on Test Data ----------

	precision	recall	f1-score	support
0.0	0.25	1.00	0.40	12
1.0	0.00	0.00	0.00	15
2.0	0.00	0.00	0.00	15
3.0	0.00	0.00	0.00	6
accuracy			0.25	48
macro avg	0.06	0.25	0.10	48
weighted avg	0.06	0.25	0.10	48

Slika 41. Primjer 3. – Vrednovanje modela

Dobiveni model daje izuzetno loše rezultate, što je bilo vidljivo već iz samih matrica zabune. Uzrok tako loših rezultata su loše vrijednosti hiperparametara. Očito je u odnosu na prethodne primjere kako je ovdje vrijednost stope učenja prevelika, pa se ne može postići minimum

funkcije pogreške što je detaljno opisano u poglavlju 2.7.1. Osim stope učenja veliki utjecaj na dobivene rezultate ima i dosta veliki broj epoha. Uz sve navedeno lošim rezultatima je pridonio i mali broj iteracija kod random search algoritma.

Rezultat predviđanja modela nad učitanom slikom je prikazan na [Slika 42.].

```
1/1 [==============================] - 0s 35ms/step
DCN model prediction:  [['dalmatian']]
1/1 [==============================] - 0s 33ms/step

Probabilities for each category:
dalmatian  :  0.31717986
hedgehog  :  0.19300477
llama  :  0.30561563
panda  :  0.1841998
```

Slika 42. Primjer 3. – Rezultat predviđanja modela

Bilo je očekivano kako će model i na temelju učitane slike predvidjeti dalmatinera. Ovim primjerom se želi pokazati kako algoritam bez kvalitetnih ulaznih podataka tj. odgovarajućih vrijednosti hiperparametara neće pružiti optimalne rezultate.

5. ZAKLJUČAK

Cilj ovog zadatka je bio definiranje optimalnih hiperparametara duboke konvolucijske neuronske mreže. Kako bi se razumjeli pojedini hiperparametri i njihov učinak na rad modela bilo je potrebno iznijeti teorijsku podlogu na početku rada. Izrada modela konvolucijske neuronske mreže, podešavanje njegovih hiperparametara i testiranje je izvedeno u programskom jeziku Python. Podešavanje hiperparametara modela dubokog učenja je neizmjerno važno ako se želi osigurati kvalitetan rad modela. U svrhu podešavanja hiperparametara su razvijeni različiti algoritmi. Ovim radom su prikazana dva takva algoritma tj. grid search i random search algoritmi. Kako bi se hiperparametri uspješno optimirali pored navedenih algoritama potrebno je znanje i iskustvo osobe koja se bavi izradom modela dubokog učenja iz razloga jer ona definira vrijednosti različitih hiperparametara koje algoritmi kasnije koriste. Navedeni algoritmi se mogu definirati samo ako alat u pronalasku optimalnih hiperparametara. U sklopu ovog zadatka pomoću grid search i random search algoritama utvrdili su se optimalni hiperparametri za model konvolucijske neuronske mreže uz koje se postigla točnost modela 94 % koja je utvrđena nad podacima za treniranje i testiranje. Bez obzira na veliku postignutu točnost model bi trebalo poboljšati na način da se poveća baza podataka koja se koristi za treniranje i testiranje modela.

LITERATURA

[1] Stipančić, T.: Podloge za predavanje iz kolegija „Umjetna inteligencija“, Fakultet strojarstva i brodogradnje, Zagreb, 2022.

[2] IBM, https://www.ibm.com/topics/machine-learning, pristupljeno: 27.12.2022.

[3] IBM, https://www.ibm.com/topics/neural-networks, pristupljeno: 27.12.2022.

[4] Towards Data Science, https://towardsdatascience.com/convolutional-neural-networks-explained-how-to-successfully-classify-images-in-python-df829d4ba761, pristupljeno: 22.1.2023.

[5] Ujević Andrijić, Ž.: OSVJEŽIMO ZNANJE: Umjetne neuronske mreže, Zagreb, 2003.

[6] Towards Data Science, https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, pristupljeno: 23.1.2023.

[7] IBM, https://www.ibm.com/topics/deep-learning, pristupljeno: 28.12.2022.

[8] IBM, https://www.ibm.com/topics/gradient-descent, pristupljeno: 2.1.2023.

[9] Microsoft, https://learn.microsoft.com/en-us/archive/msdn-magazine/2019/april/artificially-intelligent-how-do-neural-networks-learn, pristupljeno: 2.1.2023.

[10] visio.ai, https://viso.ai/applications/computer-vision-in-energy-and-utilities-industry/, pristupljeno: 22.1.2023.

[11] Mayuri – GitHub, https://mayurji.github.io/blog/2020/08/06/Hyperparameter?fbclid=IwAR3ViEtWPQt3P3jUxA-KkfWPGlBCDRvUNYe4G41xgEbX9lHLv04ahqHpwV8, pristupljeno: 23.1.2023.

[12] Django Stars, https://djangostars.com/blog/why-python-is-good-for-artificial-intelligence-and-machine-learning/, pristupljeno: 1.2.2023.

[13] InfoWorld, https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html, pristupljeno: 1.2.2023.

[14] Simplilearn, https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-keras, pristupljeno: 1.2.2023.

[15] W3Schools, https://www.w3schools.com/python/numpy/numpy_intro.asp, pristupljeno: 1.2.2023.

[16] ActiveState, https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/, pristupljeno: 1.2.2023.

[17] ActiveState, https://www.activestate.com/resources/quick-reads/what-is-pandas-in-python-everything-you-need-to-know/, pristupljeno: 1.2.2023.

[18] NVIDIA, https://www.nvidia.com/en-us/glossary/data-science/scikit-learn/, pristupljeno: 1.2.2023.

[19] Great Learning, https://www.mygreatlearning.com/blog/opencv-tutorial-in-python/, pristupljeno: 1.2.2023.

[20] CaltechDATA, https://data.caltech.edu/records/mzrjq-6wc02, pristupljeno: 2.2.2023.

[21] Towards Data Science, https://towardsdatascience.com/k-fold-cross-validation-explained-in-plain-english-659e33c0bc0, pristupljeno: 2.2.2023.

[22] scikit – learn, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html, pristupljeno: 2.2.2023.

[23] scikit – learn, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html, pristupljeno: 2.2.2023.

[24] Keras, https://keras.io/api/callbacks/early_stopping/, pristupljeno: 3.2.2023.

[25] Keras, https://keras.io/api/callbacks/model_checkpoint/, pristupljeno: 3.2.2023.

[26] Medium, https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd, pristupljeno: 5.2.2023.

PRILOZI

I. Programski kod – Grid Search

II. Programski kod – Random Search

I. Programski kod – Grid Search

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras import Input
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
from keras.callbacks import EarlyStopping
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import load_model
from keras.optimizers import Adam, SGD, RMSprop, Adadelta, Adagrad, Adamax,
Nadam, Ftrl

import pandas as pd

import numpy as np

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import cv2

import matplotlib
import matplotlib.pyplot as plt

import sys
import os

print('Tensorflow/Keras: %s' % keras.__version__)
print('numpy: %s' % np.__version__)
print('matplotlib: %s' % matplotlib.__version__)
print('pandas: %s' % pd.__version__)
print('sklearn: %s' % sklearn.__version__)
print('OpenCV: %s' % cv2.__version__)

main_dir=os.path.dirname(sys.path[0])
print('main_dir: %s' % main_dir)

ImgLocation = main_dir + '\\pythonProject\\caltech-
101\\101_ObjectCategories\\'

LABELS = set(["dalmatian", "hedgehog", "llama", "panda"])

ImagePaths = []
ListLabels = []
for label in LABELS:
    for image in list(os.listdir(ImgLocation + label)):
        ImagePaths = ImagePaths + [ImgLocation + label + "/" + image]
        ListLabels = ListLabels + [label]

data = []
for img in ImagePaths:
    image = cv2.imread(img)
    image = cv2.resize(image, (128, 128))
    data.append(image)
```

```
data = np.array(data, dtype="float") / 255.0

print("Shape of whole data: ", data.shape)

LabelsArray = np.array(ListLabels)

enc = OrdinalEncoder()
y = enc.fit_transform(LabelsArray.reshape(-1, 1))

X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.2,
random_state=0)

y_train = y_train.reshape(-1, 1)
y_test = y_test.reshape(-1, 1)

print("Shape of X_train: ", X_train.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_test: ", y_test.shape)

def model_1(learning_rate=0.001, filter=16):

    model = Sequential(name="DCN-Model")


model.add(Input(shape=(X_train.shape[1],X_train.shape[2],X_train.shape[3]),
name='Input-Layer'))

    model.add(Conv2D(filters=filter, kernel_size=(3,3), strides=(1,1),
padding='valid', activation='relu', name='2D-Convolutional-Layer-1'))
    model.add(MaxPool2D(pool_size=(2,2), strides=(2,2), padding='valid',
data_format=None, name='2D-MaxPool-Layer-1'))
    model.add(Dropout(0.2, name='Dropout-Layer-1'))

    model.add(Conv2D(filters=64, kernel_size=(3,3), strides=(1,1),
padding='valid', activation='relu', name='2D-Convolutional-Layer-2'))
    model.add(MaxPool2D(pool_size=(2,2), strides=(2,2), padding='valid',
name='2D-MaxPool-Layer-2'))
    model.add(Dropout(0.2, name='Dropout-Layer-2'))

    model.add(Conv2D(filters=128, kernel_size=(3,3), strides=(1,1),
padding='same', activation='relu', name='2D-Convolutional-Layer-3'))
    model.add(MaxPool2D(pool_size=(2,2), strides=(2,2), padding='same',
name='2D-MaxPool-Layer-3'))
    model.add(Dropout(0.2, name='Dropout-Layer-3'))

    model.add(Flatten(name='Flatten-Layer'))
    model.add(Dense(16, activation='sigmoid', name='Hidden-Layer-1',
kernel_initializer='HeNormal'))
    model.add(Dense(4, activation='softmax', name='Output-Layer'))


model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rat
e),
              loss='SparseCategoricalCrossentropy',
              metrics=['accuracy'],
              loss_weights=None,
              weighted_metrics=None,
              run_eagerly=None,
              steps_per_execution=None
              )
```

```
    return model

seed = 7
np.random.seed(seed)

model_grid_search = KerasClassifier(build_fn=model_1, verbose=1)

batches = [16, 32]
epochs = [40, 70]
learning_r = np.arange(0.001, 0.005, 0.001)
filters = [16, 32]

hyper_param = dict(epochs=epochs, batch_size=batches,
learning_rate=learning_r, filter=filters)

grid = GridSearchCV(estimator=model_grid_search, param_grid=hyper_param,
cv=3, scoring='f1_macro', refit=True)

early_stop_cb = tf.keras.callbacks.EarlyStopping(
    monitor="loss",
    min_delta=0,
    patience=10,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=True
)

early_stop_mc = keras.callbacks.ModelCheckpoint(
    "model_grid.h5",
    monitor="loss",
    save_best_only=True,
    save_weights_only=False,
    save_freq="epoch"
)

grid_result = grid.fit(X_train, y_train, callbacks=[early_stop_cb,
early_stop_mc])

print('Best hyperparameters are: '+str(grid_result.best_params_))

ucitani_model = load_model('model_grid.h5')

pred_labels_tr = np.array(tf.math.argmax(ucitani_model.predict(X_train),
axis=1))
pred_labels_te = np.array(tf.math.argmax(ucitani_model.predict(X_test),
axis=1))

cm_train = confusion_matrix(y_train, pred_labels_tr)
cm_disp_train = ConfusionMatrixDisplay(confusion_matrix=cm_train,
display_labels=['dalmatian', 'hedgehog', 'llama', 'panda'])
cm_disp_train.plot()
plt.show()

cm_test = confusion_matrix(y_test, pred_labels_te)
cm_disp_test = ConfusionMatrixDisplay(confusion_matrix=cm_test,
display_labels=['dalmatian', 'hedgehog', 'llama', 'panda'])
cm_disp_test.plot()
plt.show()

print("")
```

```
print('------------------------ Model Summary ------------------------')
ucitani_model.summary()
print("")

print("")
print('------------------------- Encoded Names -------------------------')
for i in range(0, len(enc.categories_[0])):
    print(i, ": ", enc.categories_[0][i])
print("")

print('------------------ Evaluation on Training Data ------------------')
print("")

print(classification_report(y_train, pred_labels_tr))
print("")

print('-------------------- Evaluation on Test Data --------------------')
print(classification_report(y_test, pred_labels_te))
print("")

prediction_img = cv2.imread(main_dir+'\pythonProject\data\panda.JPG')

prediction_img = cv2.resize(prediction_img, (128, 128))

prediction_img = prediction_img / 255.0

prediction_img = prediction_img[np.newaxis, ...]

print("Shape of the input: ", prediction_img.shape)
print("")

prediction =
enc.inverse_transform(np.array(tf.math.argmax(ucitani_model.predict(predict
ion_img),axis=1)).reshape(-1, 1))
print("DCN model prediction: ", prediction)

pred_probs=ucitani_model.predict(prediction_img)

print("")
print("Probabilities for each category:")
for i in range(0,len(enc.categories_[0])):
    print(enc.categories_[0][i], " : ", pred_probs[0][i])
```

II. Programski kod – Random Search

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras import Input
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
from keras.callbacks import EarlyStopping
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import load_model
from keras.optimizers import Adam, SGD, RMSprop, Adadelta, Adagrad, Adamax,
Nadam, Ftrl

import pandas as pd

import numpy as np

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import cv2

import matplotlib
import matplotlib.pyplot as plt

import sys
import os

print('Tensorflow/Keras: %s' % keras.__version__)
print('numpy: %s' % np.__version__)
print('matplotlib: %s' % matplotlib.__version__)
print('pandas: %s' % pd.__version__)
print('sklearn: %s' % sklearn.__version__)
print('OpenCV: %s' % cv2.__version__)

main_dir=os.path.dirname(sys.path[0])
print('main_dir: %s' % main_dir)

ImgLocation = main_dir + '\\pythonProject\\caltech-
101\\101_ObjectCategories\\'

LABELS = set(["dalmatian", "hedgehog", "llama", "panda"])

ImagePaths = []
ListLabels = []
for label in LABELS:
    for image in list(os.listdir(ImgLocation + label)):
        ImagePaths = ImagePaths + [ImgLocation + label + "/" + image]
        ListLabels = ListLabels + [label]

data = []
for img in ImagePaths:
    image = cv2.imread(img)
    image = cv2.resize(image, (128, 128))
    data.append(image)

data = np.array(data, dtype="float") / 255.0
```

```
print("Shape of whole data: ", data.shape)

LabelsArray = np.array(ListLabels)

enc = OrdinalEncoder()
y = enc.fit_transform(LabelsArray.reshape(-1, 1))

X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.2,
random_state=0)

y_train = y_train.reshape(-1, 1)
y_test = y_test.reshape(-1, 1)

print("Shape of X_train: ", X_train.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_test: ", y_test.shape)

def model_1(learning_rate=0.01, filter=16):

    model = Sequential(name="DCN-Model")


model.add(Input(shape=(X_train.shape[1],X_train.shape[2],X_train.shape[3]),
name='Input-Layer'))

    model.add(Conv2D(filters=filter, kernel_size=(3,3), strides=(1,1),
padding='valid', activation='relu', name='2D-Convolutional-Layer-1'))
    model.add(MaxPool2D(pool_size=(2,2), strides=(2,2), padding='valid',
data_format=None, name='2D-MaxPool-Layer-1'))
    model.add(Dropout(0.2, name='Dropout-Layer-1'))

    model.add(Conv2D(filters=filter*2, kernel_size=(3,3), strides=(1,1),
padding='valid', activation='relu', name='2D-Convolutional-Layer-2'))
    model.add(MaxPool2D(pool_size=(2,2), strides=(2,2), padding='valid',
name='2D-MaxPool-Layer-2'))
    model.add(Dropout(0.2, name='Dropout-Layer-2'))

    model.add(Conv2D(filters=filter*4, kernel_size=(3,3), strides=(1,1),
padding='same', activation='relu', name='2D-Convolutional-Layer-3'))
    model.add(MaxPool2D(pool_size=(2,2), strides=(2,2), padding='same',
name='2D-MaxPool-Layer-3'))
    model.add(Dropout(0.2, name='Dropout-Layer-3'))

    model.add(Flatten(name='Flatten-Layer'))
    model.add(Dense(16, activation='sigmoid', name='Hidden-Layer-1',
kernel_initializer='HeNormal'))
    model.add(Dense(4, activation='softmax', name='Output-Layer'))


model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rat
e),
              loss='SparseCategoricalCrossentropy',
              metrics=['accuracy'],
              loss_weights=None,
              weighted_metrics=None,
              run_eagerly=None,
              steps_per_execution=None
              )
    return model
```

```
seed = 7
np.random.seed(seed)

model_random_search = KerasClassifier(build_fn=model_1, verbose=1)

batches = [64, 128]
epochs = [60, 80]
learning_r = np.arange(0.001, 0.005, 0.001)
filters = [32, 64]

hyper_param = dict(epochs=epochs, batch_size=batches,
learning_rate=learning_r, filter=filters)

random_rf = RandomizedSearchCV(estimator=model_random_search,
param_distributions=hyper_param, n_iter=5, scoring='f1_macro', cv=3,
refit=True)

early_stop_cb = tf.keras.callbacks.EarlyStopping(
    monitor="loss",
    min_delta=0,
    patience=10,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=True
)

early_stop_mc = keras.callbacks.ModelCheckpoint(
    "model_random.h5",
    monitor="loss",
    save_best_only=True,
    save_weights_only=False,
    save_freq="epoch"
)

random_result = random_rf.fit(X_train, y_train, callbacks=[early_stop_cb,
early_stop_mc])

print('Best hyperparameters are: '+str(random_result.best_params_))

ucitani_model = load_model('model_random.h5')

pred_labels_tr = np.array(tf.math.argmax(ucitani_model.predict(X_train),
axis=1))
pred_labels_te = np.array(tf.math.argmax(ucitani_model.predict(X_test),
axis=1))

cm_train = confusion_matrix(y_train, pred_labels_tr)
cm_disp_train = ConfusionMatrixDisplay(confusion_matrix=cm_train,
display_labels=['dalmatian', 'hedgehog', 'llama', 'panda'])
cm_disp_train.plot()
plt.show()

cm_test = confusion_matrix(y_test, pred_labels_te)
cm_disp_test = ConfusionMatrixDisplay(confusion_matrix=cm_test,
display_labels=['dalmatian', 'hedgehog', 'llama', 'panda'])
cm_disp_test.plot()
plt.show()

print("")
```

```
print('------------------------ Model Summary ------------------------')
ucitani_model.summary()
print("")

print("")
print('------------------------- Encoded Names -------------------------')
for i in range(0, len(enc.categories_[0])):
    print(i, ": ", enc.categories_[0][i])
print("")

print('------------------ Evaluation on Training Data ------------------')
print("")

print(classification_report(y_train, pred_labels_tr))
print("")

print('-------------------- Evaluation on Test Data --------------------')
print(classification_report(y_test, pred_labels_te))
print("")

mydog = cv2.imread(main_dir+'\pythonProject\data\panda.JPG')

mydog = cv2.resize(mydog, (128, 128))

mydog = mydog / 255.0

mydog = mydog[np.newaxis, ...]

print("Shape of the input: ", mydog.shape)
print("")

pred_mydog =
enc.inverse_transform(np.array(tf.math.argmax(ucitani_model.predict(mydog),
axis=1)).reshape(-1, 1))
print("DCN model prediction: ", pred_mydog)

pred_probs=ucitani_model.predict(mydog)

print("")
print("Probabilities for each category:")
for i in range(0,len(enc.categories_[0])):
    print(enc.categories_[0][i], " : ", pred_probs[0][i])
```