Generiranje umjetnih slika lica pomoću generativnih suparničkih mreža

Ćaleta, Mislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

GENERIRANJE UMJETNIH SLIKA LICA POMOĆU GENERATIVNIH SUPARNIČKIH MREŽA

Diplomski rad

Mislav Ćaleta

Osijek, 2023.

Sadržaj

1. UVOD

U ovom radu se rješava problem generiranja umjetnih slika lica pomoću generativnih suparničkih mreža. Općenito se generativne mreže koriste za generiranje novih podataka. Generativne suparničke mreže razlikuju se od ostalih generativnih mreža po procesu učenja modela koji obavlja generiranje. Ovo učenje se obavlja na natjecateljski način, gdje generativna mreža pokušava generirati podatke koje će druga, diskriminativna mreža klasificirati kao stvarne podatke. Dok u isto vrijeme ta diskriminativna mreža pokušava naučiti razliku između stvarnih i generiranih podataka. Ova naučena generativna mreža onda se može koristiti u svrhe generiranja podataka. Prema [2], Neka od područja primjene generativnih mreža mogu biti generiranje slika u umjetničke svrhe ili primjena za pretvaranje jedne vrste slike u drugu, kao što je skice u realističnu sliku. Također se mogu pokušati koristiti za generiranje novih umjetnih podataka na kojima neki drugi modeli mogu obavljati učenje. Ovakva primjena za stvaranje ili nadopunu skupa podataka za učenje može biti korisna kada nema dovoljno podataka na raspolaganju ili je potrebna raznolikost. Prednost generativnih suparničkih mreža nad ostalim mrežama je mogućnost generiranja podataka koji više nalikuju stvarnima, kao i veća raznolikost podataka. Prilikom učenja generativnih suparničkih mreža nema potrebe za pretpostavkama distribucije stvarnih podataka, generativna mreža svakako će ju naučiti. Ovo omogućuje uzimanje raznolikih podataka iz distribucije koja je što više nalik onoj iz koje su uzeti stvarni podaci. Također, nema potrebe za postojanjem oznaka klasa u samom skupu podataka, već se jednostavno mogu postaviti u procesu učenja budući da je uvijek jasno dolaze li podaci iz skupa stvarnih podataka ili su generirani. Cilj rada je predstaviti različite modele i implementirati ih korištenjem programskog jezika Python, te modele koristiti za zadatak generiranja slika ljudskih lica i zatim ih međusobno usporediti. Na početku rada spomenuta su i sažeta neka dosadašnja postignuća i trenutna istraživanja u ovom i sličnim područjima. Nakon toga, objašnjen je način na koji umjetne neuronske mreže, generativne suparničke mreže i konvolucijske neuronske mreže rade i način na koji se te mreže mogu učiti. Zatim je objašnjen rad generativnih suparničkih konvolucijskih neuronskih mreža. Opisana je struktura modela i različiti pristupi koji se mogu koristiti kako bi se pokušali dobiti bolji rezultati. Nakon opisa modela, prikazan je i dio skupa podataka koji se u radu koristio za učenje mreže. Zatim je napravljen kratki uvod u Tensorflow Python paket koji se u radu koristio za implementiranje predstavljenih matematičkih modela. Opisane su najvažnije klase i funkcije koje su

se koristile u radu. U nastavku su predloženi teoretski modeli i zatim implementirani pomoću prijašnje objašnjenog Tensorflow paketa. U modelima su istaknute glavne razlike u odnosu na ostale modele u radu i objašnjena je njihova svrha. Na samom kraju rada prikazani su rezultati učenja svakog modela, odnosno, prikazane su slike koje je svaki model generirao. Također su prikazane neke brojčane mjere, kao što je vrijeme učenja modela. Za svaki model, ukazano je na uspješnost dodane razlike nad ostalim modelima u radu u poboljšavanju generirane slike.

2. PREGLED POSTIGNUĆA U GENERIRANJU SLIKA LICA POMOĆU GENERATIVNIH SUPARNIČKIH MREŽA

Trenutna istraživanja posvećena su tome kako bi generativne suparničke mreže generirale što realističnije slike, što bi značilo da bi slike trebale biti zadovoljavajuće rezolucije i da bi značajke na slikama trebale što više izgledati kao stvarni objekt, u ovom slučaju ljudsko lice. Postoji više pristupa tom problemu. Prema [1], mogu se vidjeti rezultati generiranja umjetnih slika spavaće sobe. Na slikama se mogu vidjeti pokazatelji podnaučenosti kao što je ponavljajuća tekstura na više generiranih primjeraka. Prema [2, str. 5], vidi se usporedba generiranja slike pomoću tri različita modela, od kojih je najuspješniji primjerak onaj generiran korištenjem modela koji je baziran na generativnim suparničkim mrežama. Prema [3], neka su istraživanja usmjerena i na upravljanje samim izgledom, odnosno stilom slike. Jedan od primjera bi bio razina pjegica na licu, a drugi bi mogao biti boja kose.

2.1. Progresivna izgradnja generativnih suparničkih mreža

Prema [4], kada se mjeri razlika između distribucije skupa za učenje i distribucije generirane pomoću generatorske mreže, ako su te dvije distribucije previše različite, može doći do prevelike nepredvidivosti u gradijentu. Tako gradijent može prestati biti dobar vodič pri učenju generativne i diskriminativne mreže. Ovaj problem ipak puno više utječe na generativnu mrežu. Budući da generativna mreža prima informaciju o kvaliteti podataka koje generira od diskriminativne mreže, nakon nekog vremena diskriminativna mreža naučit će razlikovati generirane i stvarne podatke i počet će davati značajniju informaciju generativnoj mreži. Ovaj proces detaljnije je objašnjen u 4. poglavlju. Zato je ovaj problem izraženiji pri ranijem učenju modela kada diskriminator još nije naučio razlikovati podatke. Generiranje slika visoke rezolucije otežano je zato što je lakše raspoznati lažne slike od pravih pri visokim rezolucijama. Na taj način slike visokih rezolucija još više povećavaju spomenuti problem s gradijentom. Generativna i diskriminativna mreža zato se mogu progresivno povećavati, počinjući od slika niske rezolucije, a tokom učenja se dodaju dodatni slojevi kojima se dobivaju veći detalji visokih rezolucija. Pokazalo se da ovo povećava brzinu učenja i stabilnost pri visokim rezolucijama slika.

2.2. Veliki model generativnih suparničkih mreža

Prema [5], generativne suparničke mreže pokazale su se uspješnijima pri povećanju modela i pri većim serijama skupa za učenje. Također su uočene nestabilnosti koje su karakteristične za ove mreže velikih razmjera. Ove nestabilnosti uočene su iz velike spektralne norme težina koja naglo raste. Veliki iznosi težina mogu upućivati na nestabilnost pri učenju jer se prilikom izračuna gradijenta dolazi do velikog iznosa. Ovime se rade preveliki koraci prilikom optimizacije koji uzrokuju ovu nestabilnost.

3. UMJETNE UNAPRIJEDNE NEURONSKE MREŽE

U ovom poglavlju objašnjen je rad umjetnih unaprijednih neuronskih mreža kao uvod u iduća poglavlja o generativnim suparničkim mrežama i konvolucijskim mrežama. Pokazana je osnovna teoretska pozadina neuronskih mreža, kako se najčešće dolazi do funkcije pogreške i koji su najčešći pristupi učenju neuronske mreže, odnosno minimiziranju funkcije pogreške. Također su prikazane neke aktivacijske funkcije i objašnjena je posebnost izlaznih jedinica.

3.1. O unaprijednim mrežama

Unaprijedna neuronska mreža, naziva se unaprijednom zato što unutar nje ne postoje povratne veze. Podatak **x** koji dolazi na ulaz mreže preslikava se kroz mrežu u različite prostore koristeći sve funkcije koje sačinjavaju mrežu, koja je onda i sama funkcija. Izlazna funkcija mreže primjenjuje se na prostoru u kojeg je preslikavanje obavio predzadnji sloj mreže. Prema [8, str. 164], cilj neuronske mreže je aproksimirati neku funkciju g. Ako postoji ovisnost

$$y = g(\boldsymbol{x}) \tag{3-1}$$

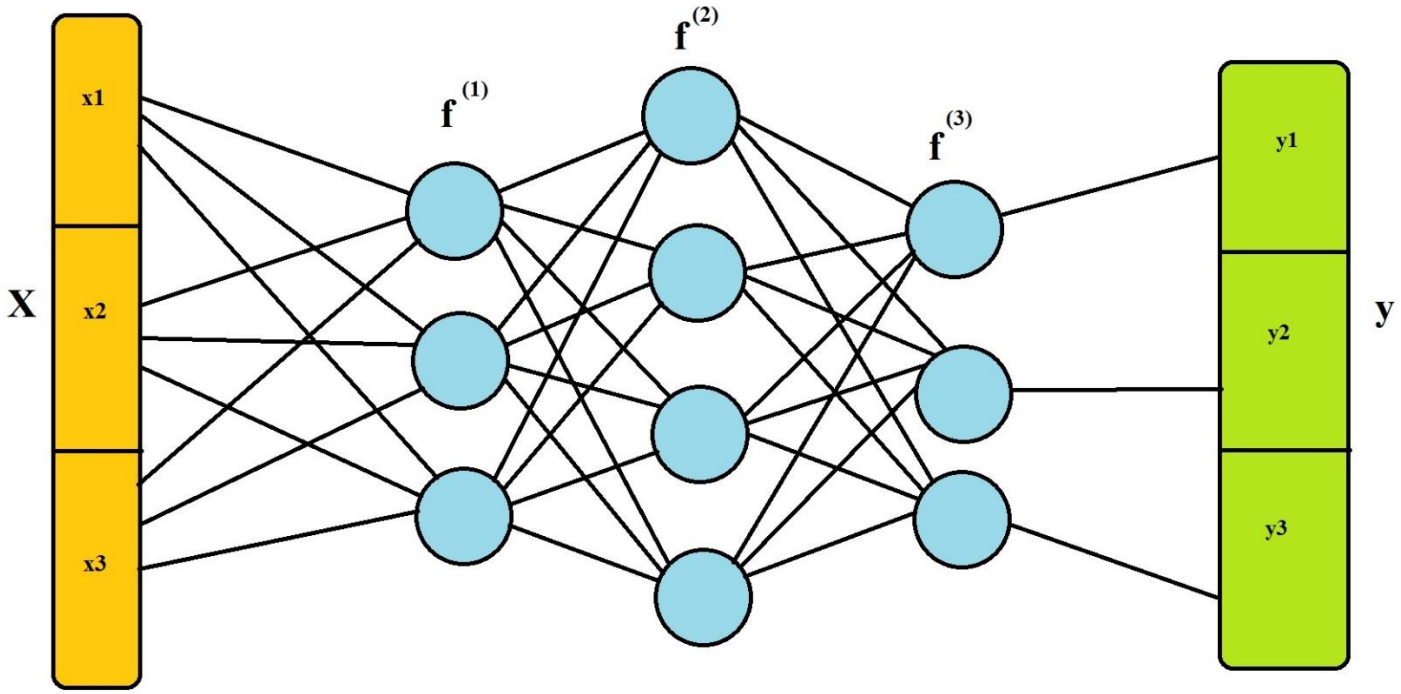
koja preslikava ulaz **x** u neku kategoriju y. Unaprijedna neuronska mreža onda će definirati preslikavanje

$$\boldsymbol{y} = f(\boldsymbol{x};\, \boldsymbol{w}) \tag{3-2}$$

gdje uči vrijednosti parametara **w**. Rezultat tog učenja su parametri koji daju najbolju aproksimaciju f funkcije g. Prema izrazu (3-3) Neuronske mreže ustvari se mogu predstaviti kao kompozicija više matematičkih funkcija, gdje svaka funkcija u kompoziciji predstavlja jedan sloj neuronske mreže. U izrazu (3-3) f^n predstavlja n-ti sloj mreže. Prvi sloj naziva se ulazni sloj, a zadnji sloj, izlazni sloj mreže.

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}^{(3)}(\boldsymbol{f}^{(2)}(\boldsymbol{f}^{(1)}(\boldsymbol{x}))) \tag{3-3}$$

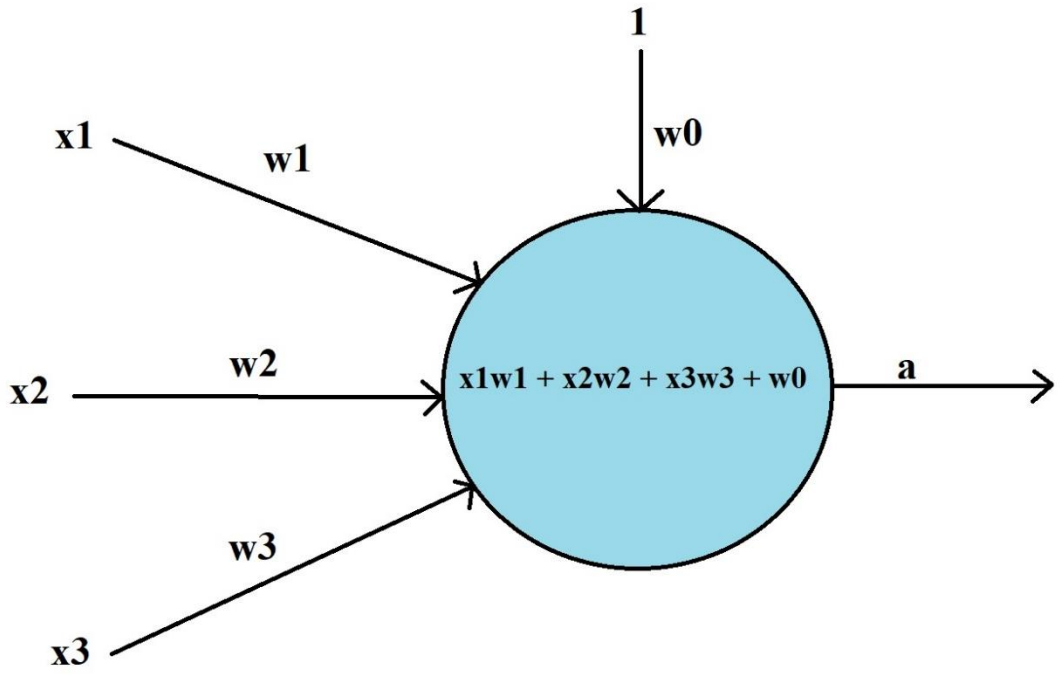
Svaki sloj mreže može se promatrati kao vektorska funkcija, koja za ulaz ima vektor, a za izlaz također daje vektor. Također se svaki sloj može promatrati kao više paralelnih jedinica ili neurona. U tom slučaju svaki neuron bi se predstavljao funkcijom više varijabli, koja za izlaz ima skalar. Odnosno preslikava vektor u skalar. Kada je sloj određen tako da je svaki ulaz povezan sa svakim izlazom prethodnog sloja preko nekog parametra w_i, onda se taj sloj naziva potpuno povezani sloj. Unaprijedne neuronske mreže, grafički se mogu prikazati kao usmjereni, aciklički graf. Kao što je prikazano na slici 3.1. Svaka linija koja ulazi u jednu jedinicu predstavlja neki parametar w_i, koji se još može nazvati i težina.



Slika 3.1 Ilustracija unaprijedne neuronske mreže s potpuno povezanim slojevima.

Prema [8 str. 165], skup za učenje pruža zašumljene primjere koji su približno jednaki $g(\mathbf{x})$, svakom primjeru iz skupa za učenje pridružena je neka oznaka y koja je približno jednaka $g(\mathbf{x})$. To znači da skup za učenje nudi željeni izlaz izlaznog sloja mreže za svaki ulazni primjer $\mathbf{x}$ u neuronsku mrežu, a to je vrijednost što bliža vrijednosti oznake y koja je pridružena tom primjeru u skupu za učenje. Izlaz ostalih slojeva neuronske mreže, onih koji nisu izlazni sloj cijele mreže, nije određen skupom za učenje, odnosno nije mu zadan željeni izlaz eksplicitno nego se koristi oznaka željenog izlaza izlaznog sloja y putem unazadne propagacije pogreške. Algoritam učenja neuronske mreže odredit će

najpogodniji izlaz za te slojeve, koji se nazivaju skrivenim slojevima. Umjetne neuronske mreže najbolje je promatrati kao univerzalne funkcijske aproksimatore. Ako bi svaki neuron neuronske mreže bio linearna funkcija svojih ulaza, onda bi, budući da je kompozicija linearnih funkcija i sama linearna funkcija, cijela mreža bila ograničena samo na aproksimaciju linearnom funkcijom. Kapacitet mreže ne bi bio dovoljan za puno zadataka. Prema slici 3.2 može se vidjeti ilustracija takve jedinice. Na slici 3.2 *w0* predstavlja pomak od ishodišta ili potpornu točku afinog potprostora.



Slika 3.2 Jedinica neuronske mreže bez aktivacijske funkcije. Na izlazu daje linearnu, točnije zbog pomaka *w0*, afinu funkciju ulaza *x1, x2 i x3*.

Kako bi se zaobišla ograničenja ovakvih linearnih jedinica radi se preslikavanje značajki. Ulaz **x** se pomoću neke nelinearne funkcije Φ preslikava u različite prostore. Svaki skriveni sloj obavlja transformaciju, dok izlazni sloj primjenjuje na tako transformiranim podacima linearni model. Ovakvo preslikavanje podataka pomoću nelinearnih funkcija omogućuje njihovu linearnu odvojivost u nekom drugom prostoru i na taj način izlazni sloj može dati željeni rezultat. U klasičnom strojnom učenju izbor funkcije Φ obavlja osoba, potragom za najboljim odrednicama te funkcije. Sama određuje na koji način će se preslikati podaci u novi prostor. Tokom učenja nema promjene i prilagođavanja te funkcije kroz algoritam učenja. Odnosno ta funkcija nije parametrizirana. Uzme li

se za primjer linearna regresija s polinomskim proširenjem, zadanim vektorskom funkcijom, određenim izrazom (3-4) onda bi model bio određen prema formuli (3-5).

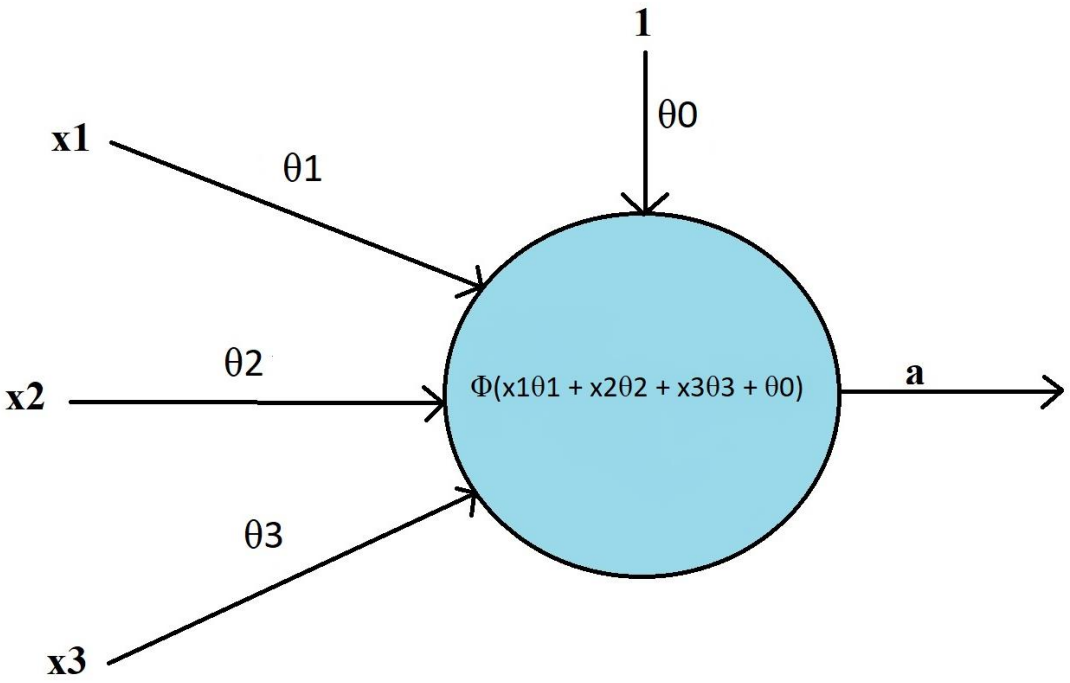
$$\boldsymbol{x} = \left[x_{1,}x_2\right],\ \boldsymbol{\Phi}(\boldsymbol{x}) = [\Phi_1(\boldsymbol{x}), \Phi_2(\boldsymbol{x}), \Phi_3(\boldsymbol{x}), \Phi_4(\boldsymbol{x}), \Phi_5(\boldsymbol{x})] = [x_1, x_2, x_1^2, x_2^2, x_1x_2] \tag{3-4}$$

$$\Phi_1(\boldsymbol{x})w_1 + \Phi_2(\boldsymbol{x})w_2 + \Phi_3(\boldsymbol{x})w_3 + \Phi_4(\boldsymbol{x})w_4 + \Phi_5(\boldsymbol{x})w_5 + w_0 = \boldsymbol{\Phi}(\boldsymbol{x})^{\boldsymbol{T}}\boldsymbol{w} + w_0 \tag{3-5}$$

Prema [8 str. 166], drugi način izbora ove nelinearne funkcije je način koji se koristi u dubokom učenju umjetnih neuronskih mreža. U ovom načinu parametriziramo nelinearnu funkciju koja transformira ulazne podatke. Na taj način se algoritmom učenja podešavaju i parametri tog preslikavanja podataka u novi prostor, a ne samo konačnog preslikavanja. Prema izrazu (3-6), zadana je neuronska mreža s jednim skrivenim slojem.

$$f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \boldsymbol{\Phi}(\boldsymbol{x}; \boldsymbol{\theta})^T\boldsymbol{w} + w_0 \tag{3-6}$$

Kada se usporede izrazi (3-5) i (3-6) može se uočiti da je glavna razlika u parametrima $\boldsymbol{\theta}$ nelinearne funkcije $\boldsymbol{\Phi}$. Ova nelinearna funkcija koja je omotana oko linearne funkcije neurona, naziva se aktivacijska funkcija. Neuron s nekom aktivacijskom funkcijom Φ prikazan je slikom 3.3. Na izlaznom sloju prilikom regresije obično se koristi samo linearna aktivacija, dok se za zadatke klasifikacije koriste logistička sigmoida ili softmax. Sigmoida se koristi kada u zadatku klasifikacije postoje samo dvije klase u koje treba svrstati podatke, a softmax funkcija koristi se kada postoji više klasa u koje treba svrstati podatke na ulazu u model. U skrivenim slojevima koristi se više vrsta aktivacijskih funkcija ovisno o zadatku. Česte su različite vrste ReLU aktivacijskih funkcija koje mogu učinkovito umanjiti problem nestajućeg gradijetna.



Slika 3.3 Ilustracija jedinice s aktivacijskom funkcijom *Φ*.

3.2. Funkcija pogreške

Prema [8 str. 174], model generira neku uvjetnu distribuciju p (**y**|**x**; **w**), gdje **x** predstavlja podatkovni primjer iz skupa za učenje, a **y** labelu koja predstavlja željeni izlaz iz skupa za učenje. U ovom potpoglavlju opisana je funkcija pogreške i gubitka modela, čijim minimiziranjem približavamo distribuciju podataka i modela. Također je pokazano kako izvesti funkciju pogreške pomoću maksimalne izglednosti i kako doći do izraza za maksimalnu log izglednost od količine informacije samo jednog događaja, entropije, Kullback Leiblerove divergencije i unakrsne entropije čijim negiranjem dobivamo maksimalnu log izglednost. Funkcija pogreške predstavlja neku funkciju *J*, koja je parametrizirana parametrima **w** modela i njenom minimizacijom dolazi se do željenog rezultata, odnosno izlaza modela. Pogreška za samo jedan primjer iz skupa za učenje, naziva se gubitak i predstavljen je funkcijom gubitka *L*. Funkcija pogreške je ustvari očekivanje funkcije gubitka. Općenito može biti zadana izrazom (3-7).

$$J(\boldsymbol{w}) = E[L(y, f(\boldsymbol{x}; \boldsymbol{w}))] \tag{3-7}$$

U izrazu (3-7) funkcija *f* predstavlja model.

3.2.1. Unakrsna entropija

Za izvod funkcije pogreške često se koristi princip maksimalne izglednosti, odnosno minimizira se unakrsna entropija između empirijske distribucije skupa podataka za učenje i distribucije koju generira model. Tom minimizacijom funkcije unakrsne entropije postiže se minimiziranje razlike između empirijske distribucije podataka i one koju generira model. Cilj je što više približiti distribuciju modela onoj iz koje je uzet skup podataka za učenje, ali nije moguće pristupiti cijeloj distribuciji, pa se ona pokušava približiti empirijskoj. Ne koriste svi modeli princip maksimalne izglednosti za izvod funkcije pogreške, ali je vrlo čest. Shannonova entropija ili samo entropija se definira kao očekivana količina informacije u događaju uzetom iz neke distribucije, odnosno Prema [10 str. 17], kao mjera za sadržaj informacije distribucije. Količina informacije jednog događaja *x* iz neke distribucije *α(x)* se može definirati prema izrazu (3-8) koji govori da manje vjerojatni događaji sadrže više informacija. Ako je baza logaritma dva, onda tu količinu informacije mjerimo u bitovima, a funkcija *I* onda predstavlja količinu informacije tog događaja.

$$I(x) = -log_2 \alpha(x) \tag{3-8}$$

Prema tome entropija te distribucije može se definirati po izrazu (3-9), gdje *E* predstavlja matematičko očekivanje.

$$H(x) = E_{x\sim\alpha}[I(x)] = -E_{x\sim\alpha}[log_2 \alpha(x)] \tag{3-9}$$

Ako postoji distribucija *q*, koju generira model i distribucija podataka *p*, onda se prema [11], pomoću Kullback Leibler divergencije može izmjeriti koliko je distribucija *p* podataka, različita od druge distribucije modela *q*. Razlika je izražena u bitovima zbog baze logaritma dva. Također je uvijek nenegativna. Kada je vrijednost nula, može se zaključiti da su distribucije identične, a što je vrijednost veća to su distribucije različitije. Kullback Leibler divergencija je dana prema izrazu (3-10).

$$D_{KL}(p||q) = \sum_i p(x_i) log_2 \frac{p(x_i)}{q(x_i)} = E_{x\sim p}\left[log_2 \frac{p(x)}{q(x)}\right] = E_{x\sim p}[log_2 p(x) - log_2 q(x)] \quad (3\text{-}10)$$

Prema [12], unakrsna entropija također se može definirati kao udaljenost između dvije distribucije, a udaljenost pomoću koje se definira unakrsna entropija je Kullback Leibler divergencija. Prema izrazu (3-11) i (3-12) zadana je unakrsna entropija između distribucije podataka *p* i distribucije modela *q*.

$$H(p,q) = H(p) + D_{KL}(p||q) = -E_{x\sim p}[log_2 p(x)] + E_{x\sim p}[log_2 p(x) - log_2 q(x)] \quad (3\text{-}11)$$

Sada iz izraza (3-11) jednostavno slijedi izraz (3-12) koji predstavlja unakrsnu entropiju.

$$H(p,q) = -E_{x\sim p}[log_2 q(x)] \quad (3\text{-}12)$$

Budući da se minimizacijom unakrsne entropije količina informacije podatka *x* za distribuciju *q* približava količini informacije podatka *x* za distribuciju *p*, odnosno približava se entropija distribucije *q* entropiji distribucije *p*, iz izraza (3-11) i (3-12) se može uočiti da je minimizacija Kullback Leiblerove divergencije između distribucija *p* i *q* ekvivalentna minimizaciji unakrsne entropije između tih distribucija.

3.2.2. Maksimalna izglednost

Prema [13 str. 93], ako postoji skup podataka $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$ koji sadrži n primjera koji su neovisni i uzeti iz Gaussove multivarijantne distribucije, parametri te distribucije mogu se estimirati pomoću maksimalne izglednosti. Prema [8 str. 129] u pitanju ne mora biti samo Gaussova distribucija, nego neka prava, ali nepoznata distribucija $p_{gen}(\mathbf{x})$ koja generira te podatke. Također postoji i familija distribucija $p_{model}(\mathbf{x}; \boldsymbol{\theta})$ koja je definirana nad istim prostorom i parametrizirana je parametrima $\boldsymbol{\theta}$. Onda će po dosadašnjem objašnjenju funkcija maksimalne izglednosti biti zadana prema izrazu (3-13).

$$\boldsymbol{\theta}_{ML} = argmax_{\boldsymbol{\theta}} \prod_{i=1}^{n} p_{model}(\boldsymbol{x_i}; \boldsymbol{\theta}) \quad (3\text{-}13)$$

Ovim izrazom dobivamo parametre distribucije, koji će maksimizirati produkt vjerojatnosti svih izmjerenih podataka iz skupa *X*. Ipak ovakav izraz nije pogodan za rad računalom i sklon je gubitku numeričke preciznosti. Zbog toga se uvodi logaritam produkta, parametri koji maksimiziraju izraz ostaju isti, ali produkt se pretvara u sumu što smanjuje taj problem. Sada se ovakva funkcija naziva, maksimalna log izglednost i zadana je izrazom (3-14).

$$\boldsymbol{\theta_{ML}} = argmax_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log\, p_{model}(\boldsymbol{x_i}; \boldsymbol{\theta}) \tag{3-14}$$

Prema [8 str. 130] ako se izraz (3-14) podjeli s brojem primjera n, iz njega slijedi očekivanje s obzirom na empirijsku distribuciju p_{emp} koju definira skup podataka za učenje. To očekivanje je dano izrazom (3-15).

$$\boldsymbol{\theta_{ML}} = argmax_{\boldsymbol{\theta}} E_{\boldsymbol{x}\sim p_{emp}} \log p_{model}(\boldsymbol{x_i}; \boldsymbol{\theta}) \tag{3-15}$$

Maksimiziranje log izglednosti može se također, kao i unakrsna entropija, promatrati kao minimiziranje Kullback Leiblerove divergencije između dvije distribucije. U ovom slučaju približavanje distribucije p_{model} distribuciji p_{emp}. To se može uočiti i iz izraza (3-15) gdje bi se uklanjanjem argmax funkcije i dodavanjem minusa dobio izraz za unakrsnu entropiju između te dvije distribucije. Što znači da je maksimizacija log izglednosti isto kao minimizacija unakrsne entropije. Zbog navedenog, bilo koja funkcija pogreške koja se sastoji od negativne log izglednosti, je unakrsna entropija između empirijske distribucije definirane podacima i vjerojatnosne distribucije definirane modelom. Budući da model generira uvjetnu distribuciju, estimacija parametara pomoću maksimalne izglednosti može se i treba raditi za uvjetnu distribuciju. Gdje bi izraz za maksimalnu log izglednost onda bio zadan prema izrazu (3-16).

$$\boldsymbol{\theta_{ML}} = argmax_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log\, p_{model}(\boldsymbol{y_i}|\boldsymbol{x_i}; \boldsymbol{\theta}) \tag{3-16}$$

Izborom različitih parametriziranih familija uvjetne distribucije za p_{model} dolazi se do različitih funkcija pogreške, ovisno o modelu. Na primjer, za linearnu regresiju može se reći da proizvodi

uvjetnu Gaussovu distribuciju. Zbog toga bi se za linearnu regresiju, p_{model} izabrao po izrazu (3-17). Gdje *N* predstavlja Gaussovu distribuciju.

$$p_{model}\left(y \mid \boldsymbol{x}\right) = N(y; f(\boldsymbol{x}; \boldsymbol{w}), \sigma^2) \tag{3-17}$$

Gdje u izrazu (3-17) f (**x**; **w**) predstavlja sam model linearne regresije koji uči srednju vrijednost Gaussove distribucije. Ovime se estimacijom parametara distribucije koristeći maksimalnu log izglednost, gdje sada srednju vrijednost predstavlja model, a standardna devijacija je konstanta i ne estimira se, ustvari pronalaze najbolji parametri modela. Razlog tomu je što je parametrima modela parametriziran sam parametar srednje vrijednosti distribucije, a time je parametrima modela onda parametrizirana i distribucija. Ne mora se parametrizirati samo srednja vrijednost pomoću parametara modela, to se može napraviti za druge parametre distribucije, ovisno o tome koji model i koja distribucija je u pitanju. Prema [8 str. 132] izraz (3-17) se može svesti na izraz koji ima iste minimizatore **w** kao srednja kvadratna pogreška koja bi se inače koristila kao funkcija pogreške za linearnu regresiju. Iz toga se zaključuje da je srednja kvadratna pogreška, samo unakrsna entropija između empirijske uvjetne distribucije i Gaussove distribucije modela. Ovime je zaključeno ovo poglavlje i pokazano je kako se pomoću estimatora maksimalne log izglednosti, uvrštavajući željenu distribuciju i parametrizirajući je pomoću modela može izvesti funkcija pogreške za veliki broj modela.

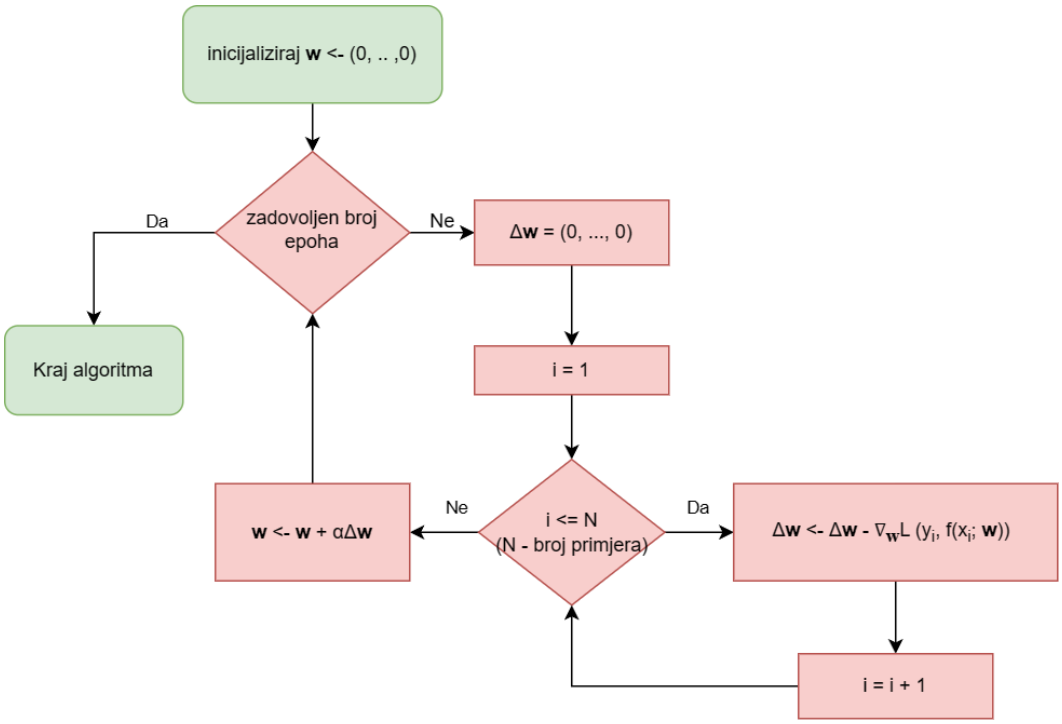
3.3. Proces učenja

U ovom potpoglavlju opisan je osnovni algoritam gradijentnog spusta i drugi iterativni optimizacijski algoritmi koji mogu poslužiti kao alternativa ili poboljšanje osnovnog algoritma gradijentnog spusta. Prema [9], gradijenti spust je način da se minimizira neka ciljna funkcija $J(\mathbf{w})$ koja je parametrizirana parametrima modela *w* tako što se ažuriraju parametri modela u suprotnom smjeru gradijenta ciljne funkcije, $\nabla_w J(\mathbf{w})$. Budući da gradijent pokazuje u smjeru najbržeg porasta funkcije, tako će negativni gradijent pokazivati u smjeru najbržeg spusta. Stopa učenja α određuje koliko velik će biti korak pri svakom ažuriranju parametara. Stopa učenja može biti stalna, promjenjiva ili se može određivati svaki

korak nekom metodom za pronalazak optimalne stope učenja. Također je opisan algoritam unazadne propagacije pogreške koji služi za ažuriranje parametara neuronske mreže.

3.3.1. Grupni gradijentni spust

Grupni gradijentni spust (engl. *Batch gradient descent*) pri izračunu vektora pomaka, odnosno gradijenta funkcije s obzirom na parametre modela, uzet će u obzir sve primjere koji se nalaze u skupu za učenje. Prema [9], Budući da se gradijent funkcije računa uzimajući u obzir cijeli skup podataka za učenje, ovakav postupak može biti vrlo spor i memorijski zahtjevan. Cijeli općeniti algoritam grupnog gradijentnog spusta može se vidjeti na slici 3.4. Na slici u obzir nije uzeto postavljanje stope učenja α budući da se po izboru može odrediti proizvoljno kao konstanta prije samog izvođenja algoritma ili se može odrediti nekom metodom tokom samog izvođenja algoritma, nakon izračunavanja vektora pomaka $\Delta\mathbf{w}$. Vektor $\mathbf{w}$ predstavlja parametre modela, a funkcija L je funkcija gubitka za pojedini primjer. Može se uočiti i sa slike kako u slučaju grupnog gradijentnog spusta vektor $\Delta\mathbf{w}$ predstavlja sumu svih negativnih gradijenata funkcije gubitka za svaki pojedini primjer, što nije slučaj kod stohastičkog gradijentnog spusta.



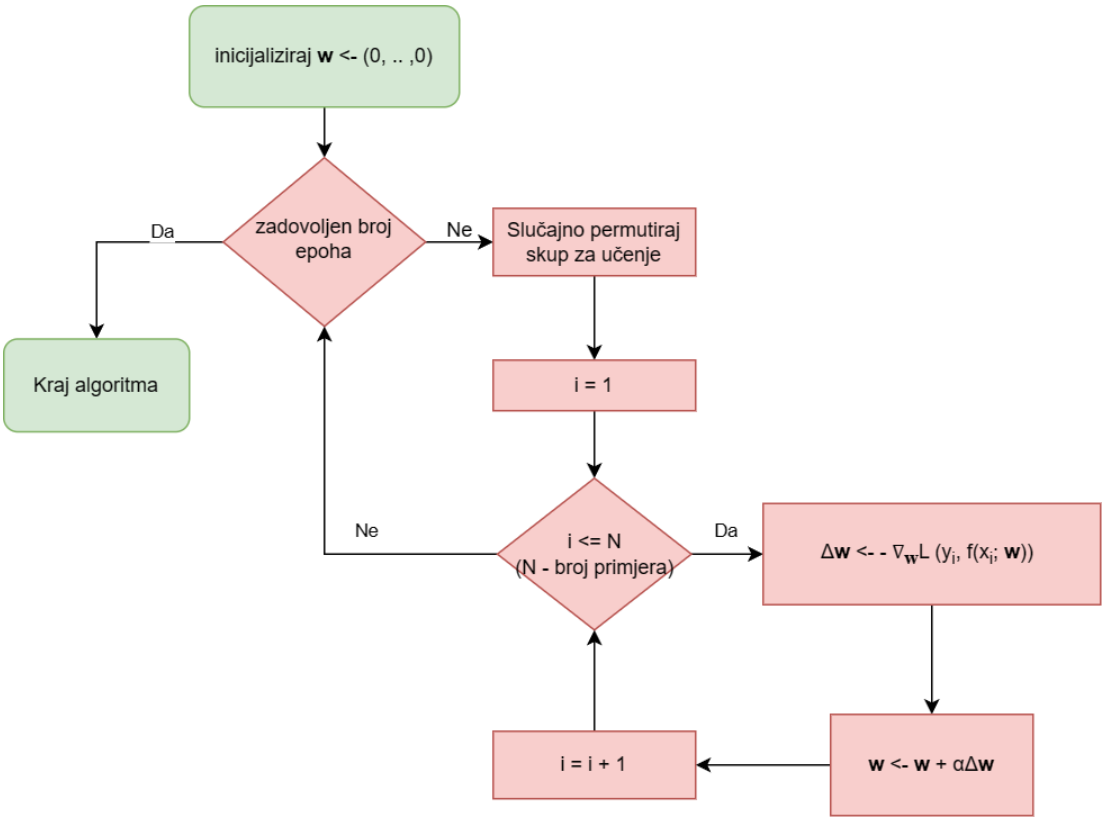
Slika 3.4 Dijagram toka grupnog gradijentnog spusta.

Ažuriranje parametara u algoritmu grupnog gradijentnog spusta može se pokazati izrazom (3-18).

$$\boldsymbol{w}^* = \boldsymbol{w} - \alpha \sum_{i=1}^{N} \nabla_{\boldsymbol{w}} L(y_i, f(\boldsymbol{x_i}; \boldsymbol{w})) \tag{3-18}$$

3.3.2. Stohastički gradijentni spust

Stohastički gradijentni spust, za razliku od grupnog gradijentnog spusta, ažuriranje parametara napravit će za svaki primjer u skupu za učenje. Pri izračunu gradijenta za jedan primjer, parametri cijelog modela odmah će biti ažurirani. Prema [9], stohastički gradijentni spust izvodi često ažuriranje parametara s visokom varijancom što uzrokuje velike fluktuacije u vrijednosti funkcije pogreške. Stohastički gradijentni spust može izazvati probleme zbog preskakanja minimuma prema kojem treba konvergirati, ali postupnim smanjivanjem stope učenja može imati slična svojstva kao grupni gradijentni spust. Prema slici 3.5 može se vidjeti dijagram toka stohastičkog gradijentnog spusta.



Slika 3.5 Dijagram toka stohastičkog gradijentnog spusta.

Ažuriranje parametara u algoritmu stohastičkog gradijentnog spusta može se pokazati izrazom (3-19).

$$\boldsymbol{w}^* = \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} L(y_i, f(\boldsymbol{x_i}; \boldsymbol{w})) \tag{3-19}$$

3.3.3. Minibatch gradijentni spust

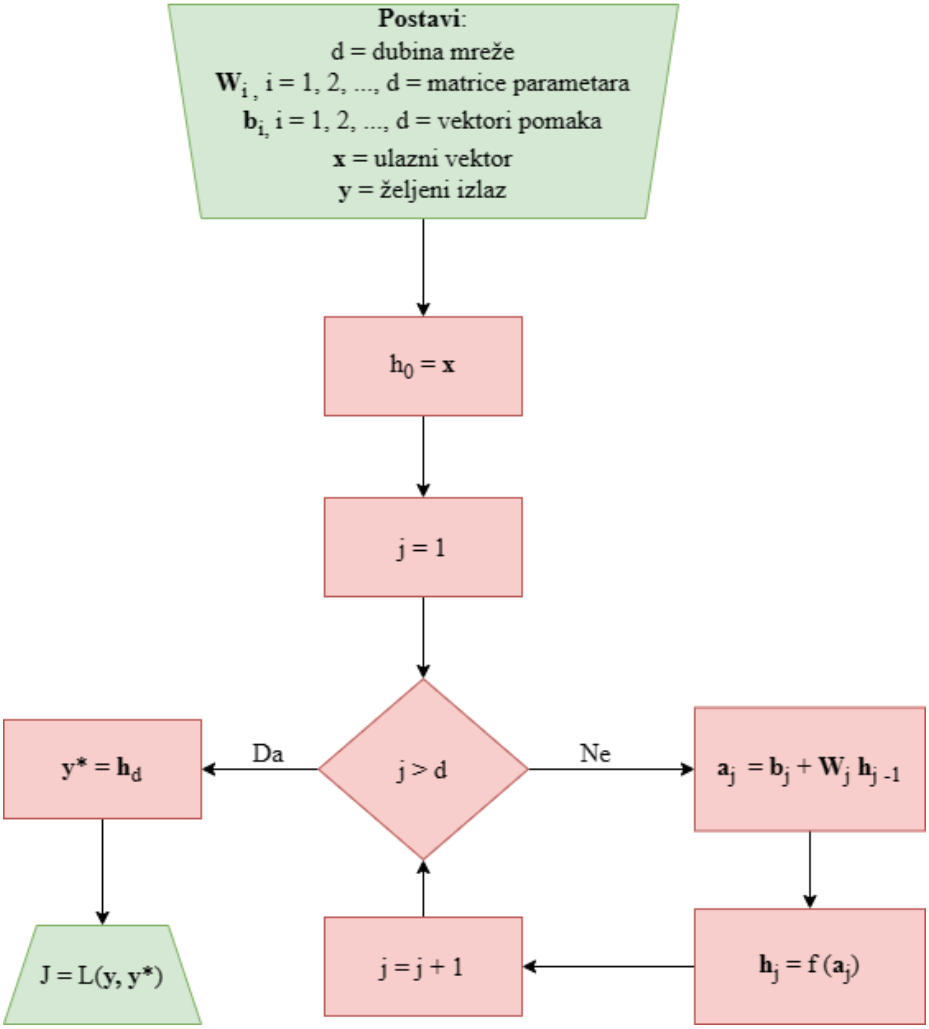
Minibatch gradijentni spust pokušava pronaći ravnotežu između algoritma grupnog gradijentnog spusta i stohastičkog gradijentnog spusta. Prema [9], ovakva vrsta gradijentnog spusta ima manju varijancu ažuriranja parametara, što vodi do stabilnije konvergencije. Ažuriranje parametara se obavlja za svakih n primjera iz skupa za učenje. Tako se skup za učenje dijeli u manje *minibatcheve* čija se veličina može prilagoditi po potrebi.

3.3.4. Unazadna propagacija pogreške

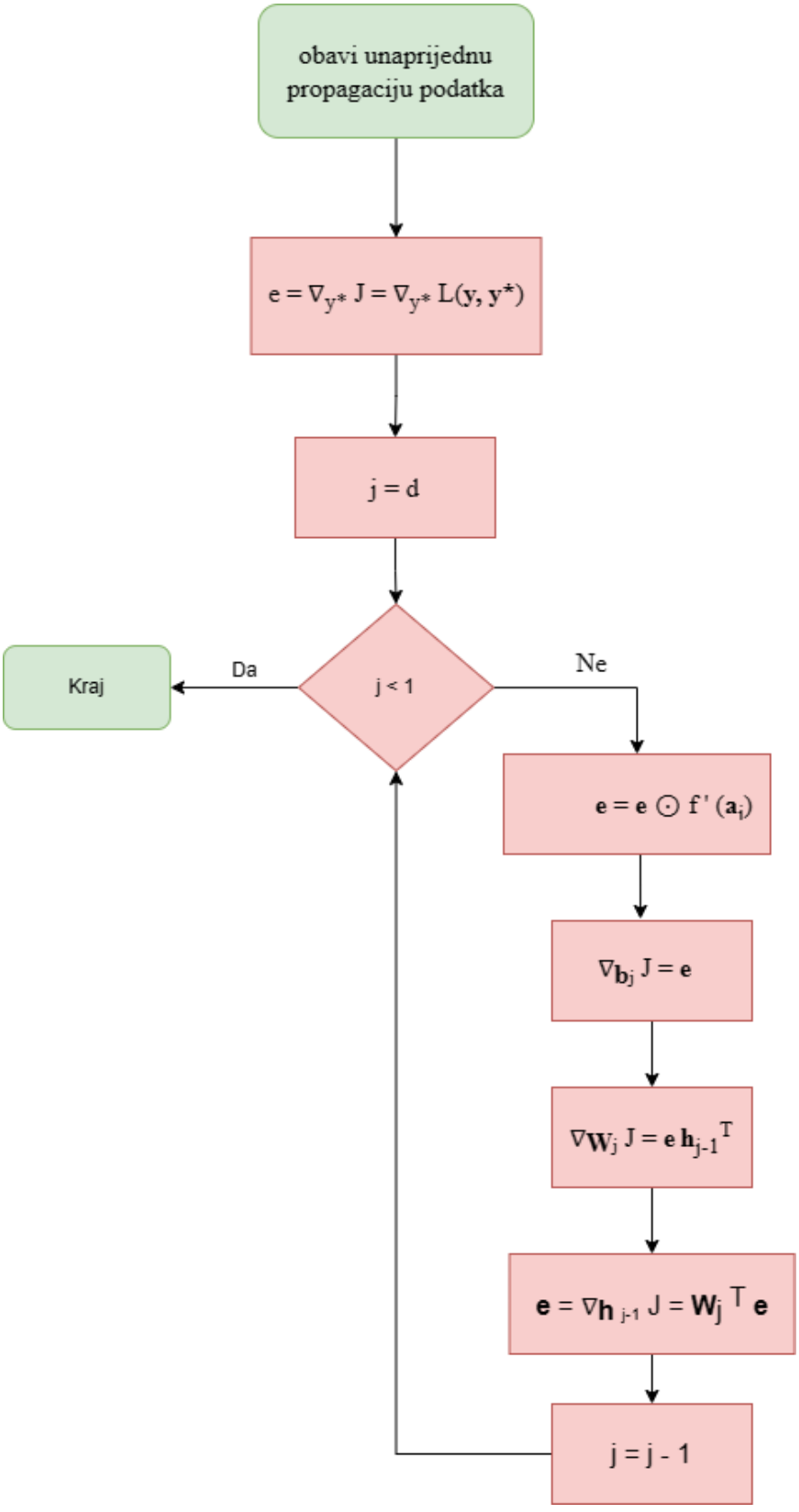
Unaprijedna propagacije informacije je širenje podatka od ulaza do izlaza mreže i na kraju računanje funkcije pogreške. Prema [8, str. 200], dok je unazadna propagacija pogreške kroz mrežu, širenje greške od izlaza mreže do samog početka, kako bi se mogli izračunati gradijenti funkcije pogreške i kako bi se ispravno obavilo ažuriranje parametara. Treba uzeti u obzir da je unazadna propagacija pogreške algoritam izračuna gradijenata funkcije pogreške s obzirom na parametre mreže. Samo ažuriranje parametara obavlja se po jednom od oblika iterativne optimizacije kao što je neki od algoritama gradijentnog spusta. Propagacija pogreške kroz mrežu unazad je ustvari deriviranje složene funkcije, ali uz posebnosti implementacije prilagođene za računalo. Postoje različite implementacije ovog algoritma ovisno o resursima koji su najviše na raspolaganju. Vrlo jednostavna neuronska mreža može se predstaviti kompozicijom funkcija, prema izrazu (3-3). Želi li se dobiti gradijent funkcije $f^{(3)}$ s obzirom na vektor **x**, treba pratiti izraz (3-20). koji nije ništa više nego pravilo deriviranja kompozicije funkcija.

$$\nabla_{\boldsymbol{x}} f^{(3)} = \frac{\partial f^{(3)}}{\partial f^{(2)}} \frac{\partial f^{(2)}}{\partial f^{(1)}} \frac{\partial f^{(1)}}{\partial \boldsymbol{x}} \tag{3-20}$$

Prema [8, str. 205], kod implementacije ovog pravila pri provođenju unaprijedne propagacije informacije kroz neuronsku mrežu, ako je dovoljno memorije na raspolaganju, što je često slučaj, vrijednosti aktivacija $f^{(1)}(\mathbf{x})$ i $f^{(2)}(f^{(1)}(\mathbf{x}))$ i ostatak vrijednosti aktivacija ovisno o dubini mreže mogu se spremiti u zasebne varijable. Na ovaj način kada se provodi algoritam unazadne propagacije pogreške nije potrebno opet računati vrijednosti aktivacija slojeva, nego se samo mogu iskoristiti već izračunate vrijednosti tih aktivacija kada je algoritam u koraku izračuna gradijenta pogreške s obzirom na parametre određenog sloja mreže. Ako je memorija ograničena aktivacije se mogu računati svaki puta od početka, ali to će usporiti učenje mreže. Dijagram toka za propagaciju informacije kroz mrežu prema naprijed može se vidjeti na slici 3.7, a dijagram toka za algoritam unazadne propagacije pogreške može se vidjeti na slici 3.8. Oba algoritma odnose se na slučaj potpuno povezane unaprijedne neuronske mreže.



Slika 3.7 Dijagram toka propagacije podatka u unaprijednoj neuronskoj mreži.



Slika 3.8 Dijagram toka unazadne propagacije pogreške za potpuno povezanu unaprijednu mrežu.

U dijagramima toka pomoću slova **h** označeni su izlazi iz slojeva mreže, **a** predstavlja linearnu aktivaciju sloja, **e** predstavlja grešku određenog sloja, a **f** nelinearnu funkciju. Simbol ⊙ predstavlja Hadamard produkt gdje se elementi lijeve strane množe odgovarajućim elementima desne strane. Član **e** koji je pogreška sloja mreže koristi se za propagaciju pogreške kroz mrežu unazad i pomoću njega se u svakom sloju izračuna gradijent funkcije pogreške s obzirom na parametre tog sloja. Nakon toga se nastavlja propagirati greška. Za lakše shvaćanje algoritma treba uzeti u obzir da vrijedi izraz (3-21).

$$\nabla_{\boldsymbol{W}_j} J = \boldsymbol{e}\boldsymbol{h}_{j-1} = \boldsymbol{e}\frac{\partial \boldsymbol{a}_j}{\partial \boldsymbol{W}_j} = \frac{\partial J}{\partial \boldsymbol{h}_d}\frac{\partial \boldsymbol{h}_d}{\partial \boldsymbol{a}_d}\frac{\partial \boldsymbol{a}_d}{\partial \boldsymbol{h}_{d-1}} \dots \frac{\partial \boldsymbol{a}_j}{\partial \boldsymbol{W}_j} \tag{3-21}$$

To bi značilo da deriviranjem linearne aktivacije *j*-tog sloja s obzirom na parametre tog sloja, dobijemo izlaze iz prošlog sloja. Iz tog razloga je produkt člana koji predstavlja grešku sloja i izlaza iz (*j-1*)-tog sloja ustvari gradijent funkcije pogreške s obzirom na parametre *j*-tog sloja.

3.4. Neke aktivacijske funkcije i njihovi efekti

U ovom potpoglavlju dan je pregled nekih od najčešćih aktivacijskih funkcija. Nelinearne aktivacijske funkcije uzrokuju nekonveksnost funkcije pogreške. To znači da ne postoji sigurnost konvergencije, ali ipak ih je vrijedno koristiti s obzirom na, u prijašnjem potpoglavlju opisanu korist koju te funkcije donose. Kao što je već opisano, Prema [14], podaci obično nisu linearno odvojivi i zbog toga je potrebno pomoću nelinearnih funkcija preslikati podatke u različite prostore.

3.4.1. Logistička sigmoida

Prema [14], biološki neuroni inspirirali su sigmoidalnu aktivacijsku funkciju. Ova funkcija dat će vrijednost između jedan i nula. Problem koji se pojavljuje kod sigmoidalne aktivacijske funkcije je to da za male i velike ulaze ima zasićen izlaz. Točnije, funkcija je vrlo ravna. Zbog toga će gradijent te funkcije pri provođenju unazadne propagacije pogreške biti vrlo zanemarivog iznosa ili će biti

potpuno nula. Ovaj problem zove se problem nestajućeg gradijenta pri provođenju algoritma unazadne propagacije pogreške. Sigmoida je zadana izrazom (3-22).

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3-22}$$

3.4.2. Ispravljena linearna jedinica (ReLU)

Prema [14], ispravljena linearna jedinica je funkcija identiteta za pozitivne ulaze, a nula za negativne ulaze. Ova funkcija vrlo je jednostavna, ali također može uzrokovati problem nestajućeg gradijenta, u ovom slučaju samo za ulaze manje od nule. Zadana je izrazom (3-23).

$$ReLU(x) = \max(0, x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3-23}$$

3.4.3. Leaky ReLU (LReLU)

LReLU je aktivacijska funkcija temeljena na ispravljenoj linearnoj jedinici. Razlika je što kod ove aktivacijske funkcije za ulaze manje od nule vrijednost funkcije neće biti nula i to će omogućiti izbjegavanje problema nestajućeg gradijenta. Prema [14], glavni problem LReLU aktivacijske funkcije je odabrati zadovoljavajući nagib u linearnoj funkciji za negativne ulaze. LReLU zadana je prema izrazu (3-24).

$$LReLU(x) = \begin{cases} x, & x \geq 0 \\ 0.01 \cdot x, & x < 0 \end{cases} \tag{3-24}$$

3.5. Konvolucijske umjetne neuronske mreže.

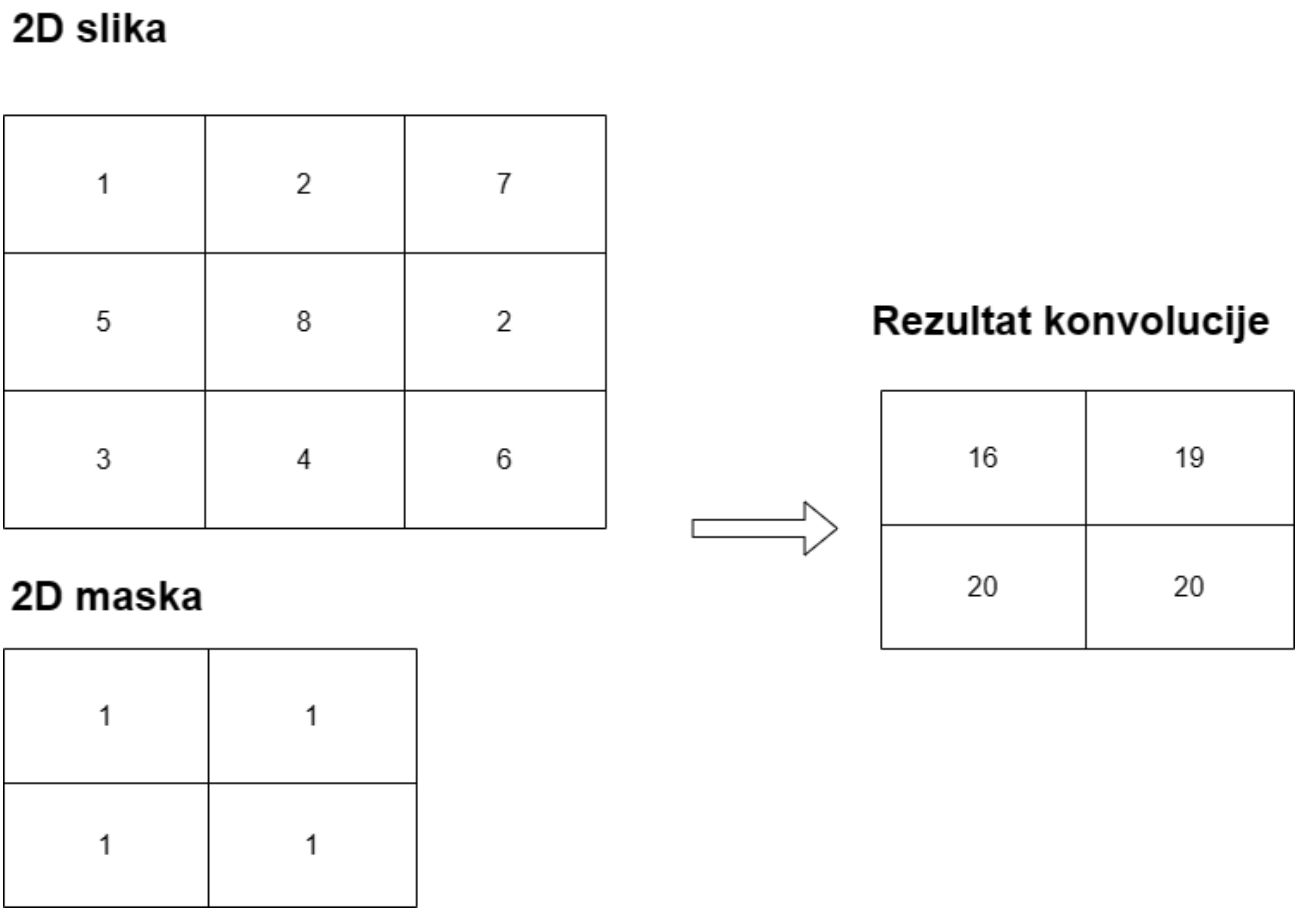
U ovom poglavlju objašnjena je teorija iza konvolucijskih neuronskih mreža. Pojašnjeni su različiti slojevi koji se u ovim mrežama koriste i zašto se baš oni koriste. Ukratko je prikazan operator konvolucije i ukazano je na njegovu ulogu u ovim mrežama. Također je zadana funkcija pogreške

koja se može koristiti pri klasifikaciji slika pomoću konvolucijskih mreža i objašnjena je njena optimizacija. Svrha konvolucijske neuronske mreže je iz slika naučiti različite značajke koje će kroz sve više slojeva postajati sve složenije i složenije. Pomoću tih naučenih značajki model bi trebao moći prepoznati klasu nove slike koja se nađe na ulazu koja prije nije viđena. U ovom radu konvolucijska neuronska mreža također se koristi u te svrhe, odnosno koristi se u svrhu diskriminativne mreže koja će služiti kao jedan od glavnih dijelova učenja generativne mreže. Pozitivna strana konvolucijske neuronske mreže je što ne postoji poravnanje slike u vektor brojeva, što bi bilo potrebno napraviti za potpuno povezanu neuronsku mrežu, čime bi se uništila prostorna informacija. Konvolucijska neuronska mreža pomoću operatora konvolucije ovaj problem zaobilazi. Maska ili filtar koji je predstavljen tenzorom u operaciji konvolucije sadrži težine. U ovom modelu to su težine koje se uče. Filtrirana slika ili mapa značajki predstavlja novu sliku gdje svaki piksel slike govori koliko je snažna prisutnost značajke koju predstavlja maska u određenim pikselima ulazne slike. Odnosno, što je veće preklapanje nekog susjedstva piksela u ulazu i maske to će vrijednost piksela koji se nalazi u mapi značajki i kojeg je proizveo operator konvolucije s odgovarajućom maskom biti veći. Budući da model ne može donijeti odluku na temelju samo jedne značajke iz slike, u mreži postoji više maski koje onda proizvode i više mapi značajki. Dimenzija slike se uz pokušaj očuvanja prostorne informacije može smanjiti po potrebi korištenjem MaxPool operatora. Na kraju je potreban potpuno povezani sloj koji će donijeti potrebnu odluku o klasi kojoj slika pripada. Sve navedeno je detaljnije objašnjeno u narednim potpoglavljima.

3.5.1. Funkcija konvolucije i konvolucijski sloj mreže

Prema [16], kao što je navedeno u uvodu ovog poglavlja, kovolucija se obavlja na način da se maska koja je predstavljena tenzorom trećeg reda, dimenzije $H_{m0} \times W_{m0} \times D_0$, pomiče po slici. Slika je također predstavljena tenzorom trećeg reda, dimenzija $H_0 \times W_0 \times D_0$, gdje svaka matrica u tenzoru predstavlja jedan kanal za boje. Treba primijetiti kako je treća dimenzija maske jednaka trećoj dimenziji slike. Npr. za sliku s tri kanala koji predstavljaju tri komponente boje: crvenu, zelenu i plavu (engl. *red, green, blue – RGB*), $D_0 = 3$. Svaka težina maske se množi odgovarajućim pikselom slike, zatim se svi ti umnošci zbrajaju i također je moguće dodati pomak. Nakon toga se na rezultat primijenjuje nelinearna aktivacijska funkcija i na taj se način dobije jedan piksel mape značajki. Nakon ove operacije, maska se po slici pomiče prema dolje za veličinu koraka. Kada maska dođe do dna, vraća

se na vrh slike i pomiče udesno za veličinu koraka. Veličina koraka ne mora uvijek biti jedan i može se podesiti po potrebi. Proces se obavlja sve dok maska ne obiđe cijelu sliku. Obavlja se dvodimenzionalna konvolucija, odnosno maska se pomiče samo u dvije dimenzije. Konvolucija se može vidjeti na slici 3.9 gdje su umjesto trodimenzionalnim tenzorima slika i maska predstavljeni matricama zbog jednostavnijeg prikaza, a pomak maske po slici iznosi jedan.



Slika 3.9 Rezultat konvolucije.

Budući da u konvolucijskoj mreži u sloju ne postoji samo jedna maska, već svaka maska predstavlja jednu naučenu značajku, onda se skup svih maski može predstaviti kao tenzor četvrtog reda, dimenzije $N_{m0} \times H_{m0} \times W_{m0} \times D_0$. Ako kroz jedan takav sloj prolazi jedna slika, onda će na izlazu postojati jednak broj mapa značajki koliko ima i maski, odnosno postojat će N_{m0} mapa značajki koje se mogu zapisati kao matrice. Zajedno sve mape značajki mogu biti zapisane kao tenzor trećeg reda dimenzije $H_1 \times W_1 \times D_1$, gdje je $D_1 = N_{m0}$. Novi skup maski onda će se opet moći predstaviti kao tenzor četvrtog reda N_{m1}

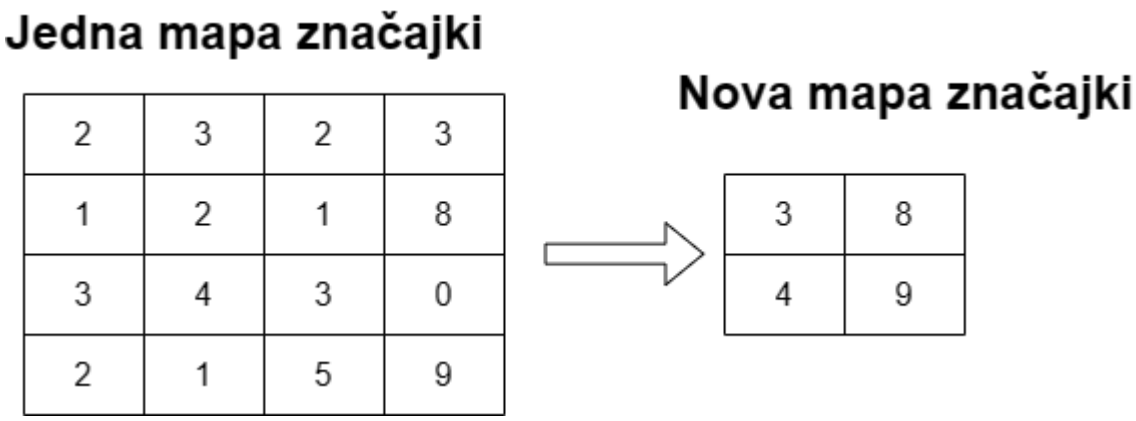
$x\ H_{m1}\ x\ W_{m1}\ x\ D_{1.}$ Za mape značajki onda se opet ponavlja proces dvodimenzionalne konvolucije, odnosno treća dimenzija tenzora maske i tenzora mapi značajki se poklapa, pa se svaki tenzor maski opet kreće samo u dva smjera. Na ovaj opisani način može se dodati potrebni broj konvolucijskih slojeva u mrežu. Na izlazu iz svakog konvolucijskog sloja, u svrhu uvođenja nelinearnosti, na svaki element mape značajki primjenjuje se neka vrsta ispravljene linearne aktivacije, kao što je ReLU. Ovo se može promatrati i kao zaseban sloj. Prema [15], duboke konvolucijske neuronske mreže uče se puno brže kada se koristi vrsta ReLU aktivacije, nego s nekom drugom aktivacijom koja zasićuje. Prema [16], prednost konvolucije je što se postepeno izdvajaju sve složenije značajke. U prvom sloju moguće je naučiti maske koje će detektirati vrlo jednostavne značajke. U idućem sloju gdje se kovolucija obavlja po novim mapama značajki, naučit će se maske koje će detektirati značajne kombinacije i prostorne odnose osnovnijih značajki, odnosno koje će detektirati kompleksnije značajke. Na taj način svaki će idući sloj učiti sve složenije značajke kao kombinacije onih detektiranih u prošlom sloju. Prema [16], još jedna od prednosti konvolucije je što sve prostorne lokacije dijele maske. To znači da se pojavljivanjem više istih obrazaca na ulaznoj slici, detektira ista značajka samo na različitim lokacijama. Opisana konvolucija za jedan piksel u mapi značajki zadana je izrazom (3-25). Gdje lijeva strana predstavlja jedan piksel u mapi značajki. Promjenom vrijednosti i^{l+1} i j^{l+1}, gdje indeks $l+1$ označava da se pomicanje obavlja po izlazu iz trenutnog sloja l, pomiče se po redovima i stupcima mape, također označavaju trenutni pomak maske na slici. Simbol n označava broj mape, odnosno broj maske. Njegovom promjenom pomiče se po mapama i maskama. Simbol m označava jednu težinu unutar maske. Unutarnja suma obavlja iteriranje kroz kanale maske i slike. Središnja suma obavlja iteriranje kroz stupce maske i slike, a vanjska suma kroz redove maske i slike. Simbol x predstavlja trenutni piksel slike po kojoj se obavlja konvolucija. Odgovarajućom promjenom i^{l+1}, j^{l+1} može se dobiti svaki piksel jedne mape značajki, a zatim promjenom broja mape n, može se dobiti svaka mapa značajki.

$$y_{i^{l+1},j^{l+1},n} = \sum_{i=0}^{H_{m^l}} \sum_{j=0}^{W_{m^l}} \sum_{k=0}^{D_l} m_{i,j,k,n} \cdot x_{i^{l+1}+i,j^{l+1}+j,k} \tag{3-25}$$

3.5.2. MaxPooling sloj mreže i način izračuna gradijenta

MaxPool operator također se sastoji od pomicanja prozora, koji je predstavljen tenzorom trećeg reda određenih dimenzija, preko slike. Ovaj prozor umjesto množenja i zbrajanja obavit će biranje

maksimalnog elementa unutar tog prozora. Izlaz iz ovog sloja bit će tenzor trećeg reda, budući da MaxPool operator svaki kanal gleda zasebno. Ovaj sloj nema parametre i ne obavlja se nikakvo učenje. Razlog za korištenje ovog sloja je smanjivanje veličine podatka na ulazu u sloj, a pri tom očuvanje prostorne informacije. Prenošenjem maksimalnih vrijednosti u novu mapu značajki na izlazu, ustvari se prenose najznačajniji podaci u ulazu, a njihov prostorni raspored je većinom sačuvan. MaxPool operator prikazan je na slici 3.10, gdje se primjenjuje dvodimenzionalni prozor na jednu mapu značajki zbog jednostavnije ilustracije. Prozor koji se pomiče je dimenzije 2 x 2, i njegov pomak je jednak vlastitim dimenzijama.



Slika 3.10 Jedan kanal MaxPool operatora primjenjen na jedan kanal tenzora mapi značajki.

Gradijenti potrebni za učenje mreže unutar konvolucijske mreže računaju se po principu unazadne propagacije pogreške. Princip širenja pogreške unazad i izračuna parametara je isti kao kod unaprijedne neuronske mreže. Razlika je što su detalji i matematika ovog algoritma prilagođeni za konvolucijsku mrežu.

3.5.3. Potpuno povezani sloj i funkcija pogreške

Potpuno povezani sloj u konvolucijskoj neuronskoj mreži koristi se kao izlazni sloj koji će donjeti konačnu odluku pri problemu klasifikacije. Kao ulaz sloj prima vektorizirani tenzor mapa značajki, odnosno cijeli tenzor se poravnava u jedan vektor. Aktivacijska funkcija pri klasifikaciji na izlaznom sloju je sigmoida ako se radi o samo dvije klase ili softmax funkcija ako se radi o višeklasnom

problemu. Principom maksimalne log izglednosti, koji je objašnjen u poglavlju o potpuno povezanim unaprijednim neuronskim mrežama, može se izvesti funkcija pogreške mreže. Ako se radi o mreži koja treba obaviti prepoznavanje samo dvije klase, pretpostavlja se Bernoulijeva distribucija podatka. Funkcija pogreške ovakve mreže naziva se unakrsna entropija, baš kao i općenitiji pojam po kojem je funkcija pogreške dobila ime, kojim se može doći do funkcija pogreške modela kada se pretpostavlja neka distribucija podataka i pokušava se približiti stvarnoj empirijskoj distribuciji podataka. Funkcija pogreške zadana je izrazom (3-26) i ustvari predstavlja unakrsnu entropiju binarnog klasifikatora kada je izlazna aktivacija sigmoida kao u izrazu (4-1), gdje **W** predstavlja parametre, **x** je izlaz iz prethodnog sloja koji bi se također trebao promatrati kao vrijednost parametrizirane funkcije, a *y* je ciljana vrijednost klasifikacije. Funkcija *f* je nelinearna aktivacija izlaznog sloja, odnosno sigmoida, a P_f je distribucija modela koja se želi približiti onoj podataka.

$$J(\boldsymbol{W}) = -E_{\boldsymbol{x}\sim p_{dat}} \log P_f(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N}(-y^i \ln f(\boldsymbol{x^i};\boldsymbol{W}) - (1-y^i)\ln\,(1-f(\boldsymbol{x^i};\boldsymbol{W}))) \quad (3\text{-}26)$$

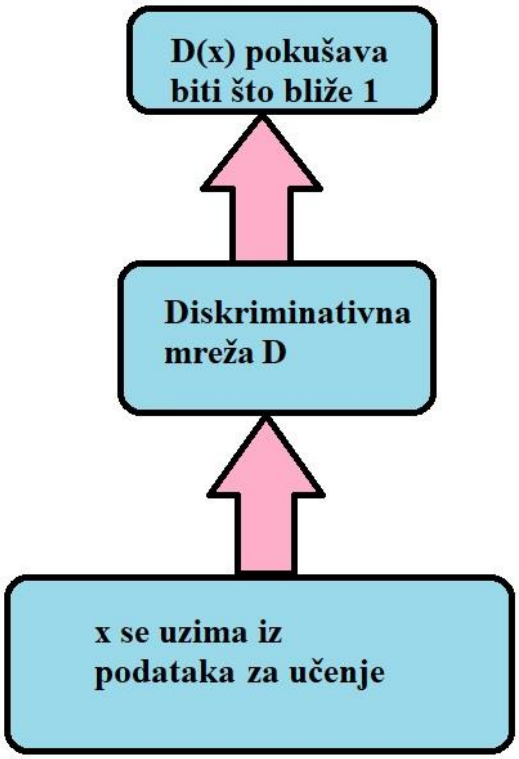
4. GENERATIVNE SUPARNIČKE UMJETNE NEURONSKE MREŽE

Cilj ovog poglavlja je objasniti detaljniju teoretsku pozadinu iza generativnih suparničkih mreža i objasniti postupak učenja ovih modela. Ovi modeli oslanjaju se na postupak učenja u kojem sudjeluju dvije umjetne neuronske mreže. Prva mreža koja generira nove podatke naziva se generatorska mreža, a druga mreža naziva se diskriminativna mreža. Cilj generatorske mreže je generirati podatke koji će biti što sličniji podacima iz empirijske distribucije skupa za učenje. Cilj diskriminativne mreže je prepoznati koji podatak dolazi iz distribucije skupa za učenje, a koji dolazi iz distribucije koju generira generatorska mreža. Kako diskriminativna mreža uči što bolje razlikovati lažne podatke od stvarnih, tako se i generativna uči što bolje generirati lažne podatke.

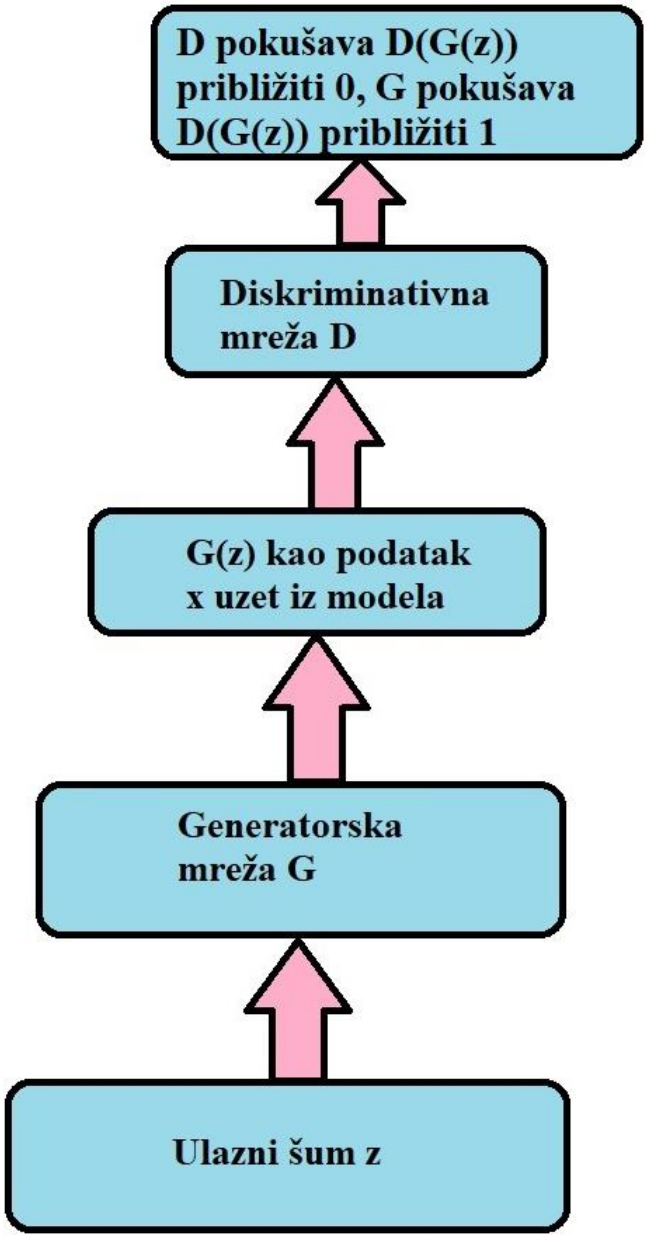
4.1. Osnovna ideja generativnih suparničkih mreža

Prema [6], kako bi generatorska mreža mogla generirati podatke što sličnije podacima iz distribucije koju određuje skup podataka za učenje koji imamo na raspolaganju, na ulaz generatorske mreže dovodi se slučajna varijabla iz neke vjerojatnosne distribucije, $p_z(\mathbf{z})$. Preslikavanje u prostor podataka koji dolaze na ulaz diskriminativne mreže izvodi derivabilna funkcija koja je najčešće neuronska mreža, $G(\mathbf{z}; \mathbf{w_g})$, gdje G označava generatorsku mrežu, a $\mathbf{w_g}$ označava parametre generatorske mreže. Generatorska mreža implicitno generira distribuciju p_g iz koje se zatim uzorkuju lažni podaci. Ova distribucija p_g i distribucija stvarnih podataka p_{dat} su definirane nad istom domenom jer je generatorska mreža preslikala varijablu $\mathbf{z}$ u tu domenu. Druga mreža koja sudjeluje u postupku učenja, diskriminativna mreža $D(\mathbf{x}; \mathbf{w_d})$, na izlazu daje vjerojatnost da podatak koji se pojavio na njenom ulazu dolazi iz skupa podataka za učenje, a ne iz distribucije koju je na izlazu generirala generatorska mreža. Obe neuronske mreže imaju funkcije pogreške koje su funkcije parametara generatorske i diskriminativne mreže. Generatorska mreža pokušava minimizirati svoju funkciju pogreške $J_g(\mathbf{w_g}, \mathbf{w_d})$, pri čemu ima pristup samo parametrima $\mathbf{w_g}$, a diskriminativna mreža pokušava minimizirati svoju funkciju pogreške $J_d(\mathbf{w_g}, \mathbf{w_d})$ pri čemu ima pristup samo parametrima $\mathbf{w_d}$. Prema [2, 7], ovakav način dobivanja željenih parametara više bi predstavljao postizanje Nashovog ekvilibrija, nego što bi predstavljao klasični optimizacijski problem gdje se dolazi do minimuma. Nashov ekvilibrij je pojam

iz matematičke teorije igara i predstavlja skup strategija gdje nijedan od igrača neće napredovati ako pokuša promijeniti svoju strategiju, što znači da su svi igrači došli do najbolje moguće strategije u ovisnosti o ostalim igračima. U slučaju generativnih suparničih mreža pri postizanju Nashovog ekvilibrija dobije se uređeni par parametara ($\mathbf{w_g}$, $\mathbf{w_d}$), to jest lokalni minimum J_d s obzirom na $\mathbf{w}_d$ i lokalni minimum J_g s obzirom na $\mathbf{w}_g$. Prema slici 4.1. može se vidjeti opisani postupak prikazan grafičkim jednostavnim modelom za diskriminativnu mrežu kada joj se na ulazu nalazi podatak iz skupa za učenje, odnosno kada se na ulazu nalazi stvarni podatak **x**. U tom slučaju cilj diskriminativne mreže bit će proizvesti izlaz što bliže broju jedan. Prema slici 4.2. također se može vidjeti opisani postupak učenja za slučaj kada se na ulazu diskriminativne mreže nalazi podatak iz distribucije modela p_g, odnosno podatak $G(\mathbf{z})$. U ovom slučaju diskriminativna mreža pokušava $D(G(\mathbf{z}))$ približiti što više broju nula, ali u ovom slučaju sudjeluje i generativna mreža koja isti $D(G(\mathbf{z}))$ pokušava što više približiti broju jedan. Prema [2], ako oba modela imaju dovoljno velik kapacitet onda se postizanjem Nashovog ekvilibrija dolazi do toga da su podaci $G(\mathbf{z})$ i podaci koji se nalaze u skupu za učenje i određuju neku empirijsku razdiobu vjerojatnosti, uzeti iz iste vjerojatnosne razdiobe.



Slika 4.1 Učenje kada je na ulazu u diskriminativnu mrežu podatak iz skupa za učenje.



Slika 4.2 Učenje kada je na ulazu u diskriminativnu mrežu podatak iz distribucije modela, p_g.

Prema [2], pri učenju potrebno je obaviti istovremeni gradijentni spust. Pri svakom koraku uzima se podskup podataka **x** iz skupa stvarnih podataka i podskup **z** iz a priori distribucije p_z latentnih varijabli. Nakon toga se rade dva optimizacijska koraka istovremeno. Jedan ažurira parametre $\mathbf{w_d}$, i minimizira J_d, a drugi ažurira parametre $\mathbf{w_g}$, i minimizira J_g.

4.2. Funkcije pogreške

Funkcija pogreške diskriminativne mreže može se zapisati izrazom (4-4), ovaj izraz predstavlja unakrsnu entropiju koja se koristi kod binarnog klasifikatora sa sigmoidalnom aktivacijom izlaza. Općenito se takva unakrsna entropija može prikazati izrazom (4-1), a ovaj se izraz može izvesti na način objašnjen u potpoglavlju 3.2. ako se pretpostavi uvjetna Bernoulijeva distribucija oznaka ulaznih podataka.

$$\frac{1}{N}\sum_{i=1}^{N}(-y^i \ln y_{mod}^i - (1-y^i)\ln(1-y_{mod}^i)) \tag{4-1}$$

Jedina razlika je što podaci ne dolaze iz istih razdioba. Podaci koji dolaze iz skupa podataka, odnosno p_{dat} svi su označeni s jedan. Podaci koji dolaze iz p_g označeni su s nula. Pa kada je na ulazu u diskriminator podatak $\mathbf{x^i}$ iz stvarnog skupa podataka onda $y^i{}_{mod}$ postaje jednak $D(\mathbf{x^i})$. Pa izraz (4-1) postaje izraz (4-2).

$$\frac{1}{N}\sum_{i=1}^{N} -\log D(\boldsymbol{x^i}) = -E_{\boldsymbol{x}\sim p_{dat}} \log D(\boldsymbol{x}) \tag{4-2}$$

Kada je na ulazu generirani podatak $G(\mathbf{z^i})$ onda $y^i{}_{mod}$ postaje $D(G(\mathbf{z^i}))$ i izraz (4-1) postaje izraz (4-3).

$$\frac{1}{N}\sum_{i=1}^{N} -\log\left(1 - D\left(G(\boldsymbol{z^i})\right)\right) = -E_{\boldsymbol{z}\sim p_z}\log\left(1 - D\left(G(\boldsymbol{z^i})\right)\right) \tag{4-3}$$

Uzme li se u obzir mogućnost jedne ili druge situacije dolazi se do izraza (4-4). Još jasnije, za podatke iz stvarnog skupa pokušava se izlaz iz diskriminatora što više približiti jedinici i time minimizirati prvi dio izraza, a za podatke koji su generirani izlaz iz diskriminatora se pokušava što više približiti nuli i time minimizirati drugi dio izraza. Na ovaj način se distribucija diskriminatora približava pravoj uvjetnoj distribuciji oznaka.

$$J_D(\boldsymbol{W}_d, \boldsymbol{W}_g) = -E_{\boldsymbol{x}\sim p_{dat}} \log D(\boldsymbol{x}) - E_{\boldsymbol{z}\sim p_z}\log\left(1 - D\big(G(\boldsymbol{z})\big)\right) \tag{4-4}$$

Prema [2], ako je izraz J_D funkcional onda se njegovim minimiziranjem s obzirom na D može dobiti izraz za diskriminativnu mrežu (4-5) koji govori da diskriminator estimira zadani omjer.

$$D^*(\boldsymbol{x}) = \frac{p_{dat}(\boldsymbol{x})}{p_{dat}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{4-5}$$

Budući da bi p_g i p_{dat} trebali biti jednaki, iz izraza (4-5) vidi se da ako je izlaz iz diskriminativne mreže prevelik, p_g je premal. Ako je izlaz iz diskriminativne mreže premal, p_g je prevelik. Ovu informaciju generatorska mreža koristi u učenju kako bi se distribucija p_g koja se generira na izlazu generatorske mreže približila što više distribuciji p_{dat}. Ako se postigne da je $p_g = p_{dat}$, izlaz iz diskriminativne mreže će po izrazu (4-5) za svaki podatak, lažni ili stvarni biti 0.5. Treba uzeti u obzir da je izraz (4-5) idealni diskriminator i da stvarni diskriminator ovaj izraz ne mora dobro estimirati i može proizvesti vrijednosti 0.5 i za podatke koje generira generatorska mreža koji nisu sasvim zadovoljavajući. Zato postizanje vrijednosti 0.5 na izlazu iz diskriminatora ne može biti konačan cilj. Ipak diskriminator na ovaj način kroz funkciju pogreške generatorske mreže, koja sadrži izraz diskriminativne mreže, može generatorskoj mreži prenijeti informaciju koliko dobro generatorska mreža uspijeva zavarati diskriminativnu mrežu. Zbog ovoga je bitno izgraditi kvalitetnu diskriminativnu mrežu koja će dobro estimirati omjer (4-5). Za generativnu mrežu može postojati više izbora funkcije pogreške, ali u ovom radu koristi se ona zadana izrazom (4-6). Prema [2], generativna mreža po ovom izrazu pokušava povećati vjerojatnost da diskriminativna mreža pogriješi u svojoj klasifikaciji.

$$J_G(\boldsymbol{W}_d, \boldsymbol{W}_g) = -E_{\boldsymbol{z}\sim p_z} \log\left(D\big(G(\boldsymbol{z})\big)\right) \tag{4-6}$$

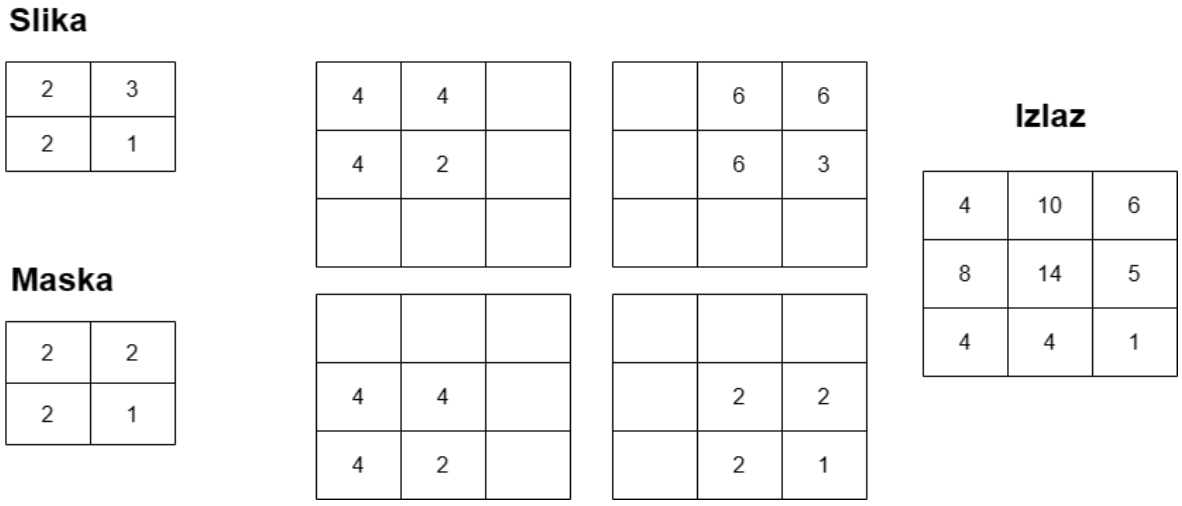
4.3. Generativne suparničke konvolucijske umjetne neuronske mreže

U ovom poglavlju opisane su generativne suparničke konvolucijske neuronske mreže, što je i glavni model o kojemu se u ovom radu govori. Opisani su slojevi transponirane konvolucije i spojeni su koncepti iz prijašnjih poglavlja kako bi se u potpunosti objasnio ovaj tip modela.

4.3.1. Transponirana konvolucija

Transponirana konvolucija omogućit će povećanje rezolucije slike. U ovom radu koristi se kako bi se slika niske rezolucije, gdje svaki piksel predstavlja podatak uzet iz a priori distribucije p_z generativne

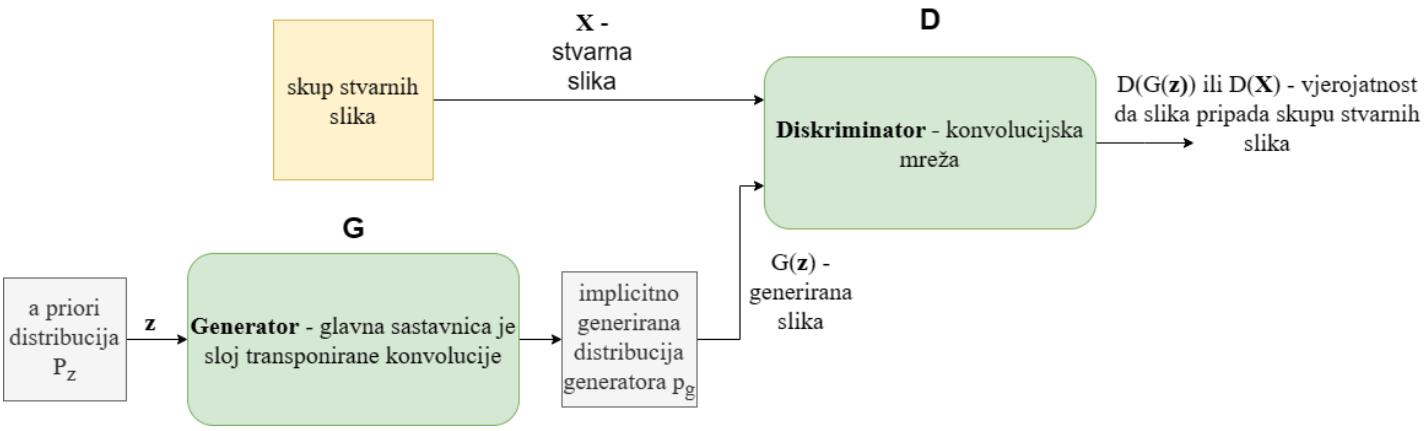
mreže, kroz više ovakvih slojeva dovela do potrebne rezolucije za ulaz u diskriminativnu mrežu. U operaciji transponirane konvolucije također se kao i u konvoluciji koristi maska. Ako bi se uzeo najjednostavniji primjer ove operacije, gdje bi pomak maske bio jedan, a popunjavanje se ne bi koristilo, onda bi se transponirana konvolucija mogla pokazati slikom 4.3. U tom slučaju svaki element ulazne slike množi se svakim elementom maske i za svaki element ulazne slike dobije se dio izlazne slike koji je dimenzije maske. Zatim se odgovarajući pikseli tih dijelova zbroje i dobije se izlazna slika. Objašnjeni postupak i slika odnose se na matrice, ali kao i kod konvolucije operacija transponirane konvolucije može proširiti na tenzore.



Slika 4.3 Transponirana konvolucija s matricama.

4.3.2. Pregled općeg modela

Opća struktura modela može se vidjeti na slici 4.4. Sve do sada već objašnjeno na slici se može prepoznati. U generativnoj mreži iz distribucije p_z na ulaz dolazi vektor **z** i preoblikuje se u sliku niske rezolucije. Zatim se kroz transponiranu konvoluciju ta slika postepeno povećava i dolazi na ulaz diskriminatora. Isto tako se uzima slika iz skupa stvarnih slika i dolazi na ulaz diskriminatora. Diskriminator, koji je samo konvolucijska mreža, za svaku sliku određuje je li slika stvarna ili je slika lažna. Razlika je što će se u praktičnom dijelu ovi podaci na ulaz diskriminatora dovoditi u skupovima, a ne pojedinačno. Nakon obavljenog učenja odbacuje se diskriminativni dio i ostavlja se samo generativna mreža koja onda iz distribucije p_z proizvodi željene slike.



Slika 4.4 Pregled rada generativne suparničke konvolucijske mreže.

5. REALIZACIJA I UČENJE GENERATIVNE SUPARNIČKE NEURONSKE MREŽE

U ovom poglavlju prikazani su podaci koji su korišteni za učenje. Također su objašnjene osnove Tensorflow i Keras paketa. Pokazane su najvažnije funkcije i postupci koji se mogu primijeniti u ovom Python paketu. Na kraju su najdetaljnije opisane funkcije i klase koje su važne za ovaj rad.

5.1. TensorFlow i Keras

Prema [19], Keras je biblioteka na razini modela, što znači da pruža način izgradnje modela na visokoj razini. Detalje kao što su operacije s tenzorima ostavlja nekoj biblioteci koja je za to napravljena, kao što je TensorFlow. Kako bi se izgradila i naučila mreža kroz ove biblioteke prvo će biti potrebno definirati skup podataka koji će se koristiti i napraviti potrebnu predobradbu. To se može napraviti korištenjem gotovih TensorFlow funkcija stvaranjem objekta klase Dataset ili se mogu koristiti neke druge biblioteke koje Keras modeli podupiru. Zatim je potrebno izgraditi model. Korištenjem Kerasa to se radi tako da se sloj po sloj mreže dodaje u model koji je objekt klase Sequential. Jednostavno dodavanje sloja u mrežu može se vidjeti na slici 5.1.

```
import tensorflow as tf
from tensorflow.keras import layers

model = tf.keras.Sequential()
model.add(layers.Dense(units = 20), activation = "relu", input_shape=(300,))
```

Slika 4.1 Dodavanje sloja u mrežu.

Nakon toga bi trebalo definirati funkcije pogreške i optimizacijski algoritam, dodavajući ih u compile() metodu koja se poziva na objektu modela. Konačno se može pozvati fit() metoda na objektu modela, zajedno s podacima koji su određeni. Ako se želi imati bolja kontrola nad procesom učenja, funkcija pogreške se može definirati i propagacija pogreške se može izvoditi pomoću GradientTape klase. Pogreška koja se dobije propagacijom podataka kroz mrežu će se zapisati u objekt ove klase i

zatim će se izvesti deriviranje po parametrima modela pomoću gradient() metode objekta klase GradientTape. Dobiveni gradijenti će se onda iskoristiti u optimizacijskom algoritmu za minimizaciju funkcije pogreške. Ovaj postupak se onda ponavlja za svaki podskup podataka za učenje. U ovom radu koristi se drugi način, budući da je potrebno definirati dvije funkcije pogreške i ažurirati parametre dva modela istovremeno. Primjer ovakvog načina učenja može se vidjeti na slici 5.2.

```
def train_step(train_batch) {
    with tf.GradientTape() as tape:
      predictions = model(train_batch, training = True)
      loss = modelLoss(predictions)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
}
```

Slika 5.2 Korištenje GradientTape klase za definiranje koraka učenja.

Svaki korak učenja može se ponavljati petljom kao što je prikazano na slici 5.3, gdje se za svaki podskup podataka poziva korak učenja.

```
def train(dataset, epochs):
  for epoch in range(epochs):
    tf.keras.backend.clear_session()
    start = time.time()
    for batch in dataset:
      train_step(batch)

    print("epoch {} took {} seconds".format(epoch + 1, time.time() - start))
```

Slika 5.3 Petlja učenja.

Evaluacija modela jako ovisi o vrsti modela. U ovom radu model generira slike i može se reći da se procjena uspješnosti može obaviti po zadovoljstvu korisnika uspoređujući rezultate sa stvarnim podacima, vremenu učenja i potrebnim podacima za učenje modela. Također se može koristiti mjera kao što je indeks strukturne sličnosti.

5.2. Učitavanje i predobrada podataka za učenje

Za početak je bilo potrebno uvesti sve Python pakete koji će se koristiti u radu, ovi paketi mogu se vidjeti na slici 5.4.

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras import layers
import gdown
import zipfile
import os
import matplotlib.image as img
import time
```

Slika 5.4 Potrebni paketi.

Zatim je potrebno skinuti skup podataka koji je opisan ranije i raspakirati ga u jedan direktorij. To se može vidjeti na slici 5.5.

```
# download dataset of celeb images
gdown.download(output = "images_download.zip",
               url = "https://drive.google.com/file/d/1C3vHqEY97eSUmEL_x0u565dXZE-MIVR6/view?usp=sharing",
               quiet = False,
               fuzzy = True)
```

```
Downloading...
From: https://drive.google.com/uc?id=1C3vHqEY97eSUmEL_x0u565dXZE-MIVR6
To: /content/images_download.zip
100%|██████████| 1.44G/1.44G [00:17<00:00, 82.1MB/s]
'images_download.zip'
```

```
# extract downloaded file
with zipfile.ZipFile("images_download.zip", "r") as zip_ref:
  zip_ref.extractall("./celeb_a")
```

Slika 5.5 Skidanje i raspakiravanje skupa podataka.

Za skidanje .zip datoteke u kojoj se nalaze slike, koristi se download() metoda biblioteke gdown. Ova biblioteka koristi se za dobavljanje datoteka s Google Drive usluge. Za raspakiravanje koristi se zipfile biblioteka, gdje se na privremenom objektu klase ZipFile poziva extractall metoda. Nakon ovog postupka treba se provjeriti valjanost skupa podataka. Na slici 5.6 može se vidjeti spremanje putanje do skupa podataka u jednu varijablu, zatim ispis imena prvih pet slika i ispis broja svih slika unutar direktorija. Na kraju se prikazuje neka slika po želji kako bi se provjerio izgled slika.

```
# create path to dataset
dataset_folder = os.path.join("celeb_a", "img_align_celeba")

# list first 5 images and print total number of images
list_of_images = os.listdir(dataset_folder)
print(list_of_images[:5])
print(len(list_of_images))
```

```
['186891.jpg', '026288.jpg', '115673.jpg', '079140.jpg', '117632.jpg']
202599
```

```
# view image
first_image = img.imread(os.path.join(dataset_folder, "007449.jpg"))

im_fig, im_ax = plt.subplots()
im_ax.set_xticks([])
im_ax.set_yticks([])
im_ax.imshow(first_image)
```

```
<matplotlib.image.AxesImage at 0x7ace31d8aa10>
```

Slika 5.6 Provjera skupa podataka.

Nastavlja se s provjerom oblika slike kako bi se utvrdilo treba li ju povećati ili smanjiti. Kao što je rečeno u poglavlju o TensorFlow biblioteci, izrađuje se dataset objekt u kojem se nalaze podaci uz zadana svojstva. U ovom slučaju slike su raspoređene u podskupove od 128 slika, veličina im je promijenjena na 256 redaka i 256 stupaca zbog lakšeg rukovanja sa slikama tokom prolaska kroz model. Suffle parametar postavlja se na vrijednost True, kako bi se slike permutirale prije izrade podskupova, a validation_split parametar koristi se za kontrolu postotka slika koji će se koristiti za

učenje. Na cijeli skup podataka primjenjuje se sloj koji se zove rescaling, odnosno primjenjuje se jednostavna funkcija koja će svaki piksel slike podijeliti s 255 i tako svaki piksel postaviti unutar intervala između nula i jedan. Na kraju ovog koraka provjerava se oblik jednog podskupa slika koji bi trebao biti tenzor četvrtog reda. Sve opisano se može vidjeti na slici 5.7.

```
# inspect image dimensions
first_image_array = np.array(first_image)
print(first_image_array.shape)
```

```
(218, 178, 3)
```

```
# load dataset into an tensorflow.dataset.Dataset object
dataset = tf.keras.utils.image_dataset_from_directory(directory = dataset_folder,
                                                      labels = None,
                                                      label_mode = None,
                                                      batch_size = 128,
                                                      image_size = (256, 256),
                                                      shuffle = True,
                                                      validation_split = 0.49,
                                                      subset = "training",
                                                      seed = 52
                                                      )
```

```
Found 202599 files belonging to 1 classes.
Using 103326 files for training.
```

```
# rescale images
rescaling = layers.Rescaling(scale = 1./255.)
dataset = dataset.map(lambda x: rescaling(x))
```

```
# check if batch has good shape in dataset

first_batch = next(iter(dataset))
print(first_batch.shape)
```

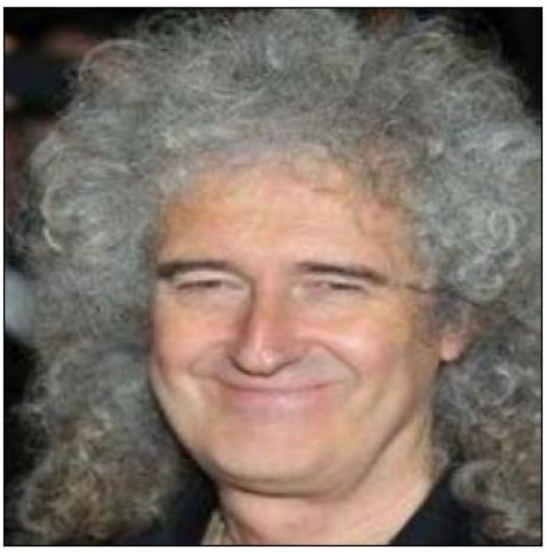
```
(128, 256, 256, 3)
```

Slika 5.7 Izrada upotrebljivog skupa podataka.

Obavlja se još jedna provjera izgleda slike nakon što joj se promijenila rezolucija, što se vidi na slici 5.8.

```
# check the look of resized image from dataset
resized_image = first_batch[0]
print(resized_image.shape)

im_fig2, im_ax2 = plt.subplots()
im_ax2.set_xticks([])
im_ax2.set_yticks([])
im_ax2.imshow(resized_image)
```

```
(256, 256, 3)
<matplotlib.image.AxesImage at 0x7c809a97b910>
```



Slika 5.8 Provjera preoblikovane slike.

Zadnji korak rada s podacima se sastoji od pozivanja cache() metode na dataset objektu. Ovime se pri svakom prvom pristupu podatku on učitava u memoriju i u svim idućim pristupanjima ovom istom podatku on se ne mora opet učitati u memoriju. Razlog ovoga je brži pristup podacima, a time i sam postupak učenja. Također se poziva prefetch() metoda na istom skupu podataka, pozivom ove metode ostvaruje se učitavanje idućeg skupa podataka i njegova priprema za obradu, tokom same obrade trenutnog skupa podataka, odnosno njegovog prolaska kroz mrežu. Argument buffer_size postavlja se na tf.data.AUTOTUNE kako bi TensorFlow sam odredio kolika je veličina međuspremnika u kojega će učitati podatke dok drugi dio prolazi kroz mrežu. Ovime se također ubrzava proces učenja. Ovaj postupak se vidi na slici 5.9.

```
# cache and prefetch data
dataset = dataset.cache().prefetch(buffer_size = tf.data.AUTOTUNE)
```

Slika 5.9 Postavljanje postavki učitavanja podataka.

5.3. Učenje neuronske mreže

Funkcije pogreške mogu se definirati baš kao što su opisane u teoriji po izrazima (4-4) i (4-6). Pogreška diskriminatora bit će zbroj unakrsnih entropija gdje se distribucija koju generira diskriminator pokušava približiti empirijskoj uvjetnoj distribuciji oznaka. Dok se minimizacijom pogreške generativne mreže, koja je isto unakrsna entropija, pokušava maksimizirati vjerojatnost pogreške diskriminatora. Optimizacijski algoritam koji se koristi je Adam sa stopom učenja od 10^{-4}. Ove funkcije pogreške i definirani optimizacijski algoritmi mogu se vidjeti na slici 5.10.

```
crossentropy = tf.keras.losses.BinaryCrossentropy(from_logits = True)

# define discriminator loss
def discriminator_loss(fake_output, real_output):
  real_loss = crossentropy(tf.ones_like(real_output), real_output)
  fake_loss = crossentropy(tf.zeros_like(fake_output), fake_output)
  total_loss = real_loss + fake_loss

  return total_loss

# define generator loss
def generator_loss(fake_output):
  return crossentropy(tf.ones_like(fake_output), fake_output)

# define optimizers
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

Slika 5.10 Funkcije pogreške i optimizacijski algoritam.

Tokom učenja mreže mogu se spremiti željena svojstva modela pomoću Checkpoint klase unutar TensorFlow biblioteke. U ovom radu spremaju se oba modela, diskriminativna i generatorska mreža,

zajedno sa svojim optimizatorima. Radi se objekt klase Checkpoint koji će sadržavati te informacije. Ovo se vidi na slici 5.11.

```
# define checkpoint information
checkpoint_dir = "./checkpoint_dir"
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer = generator_optimizer,
                                 discriminator_optimizer = discriminator_optimizer,
                                 generator = generator,
                                 discriminator = discriminator)
```

Slika 5.11 Izrada objekta Checkpoint klase.

Konačno prije procesa učenja treba definirati kako će se proces učenja odvijati. Na slici 5.12 može se vidjeti jedna takva definicija. Prvo se određuje broj epoha, određuje se dimenzija šuma odnosno kolike će vrijednosti imati vektor **z** uzet iz a priori distribucije p_z. Nakon toga se određuje veličina podskupa podataka nakon kojeg će se ažurirati težine mreže. Jedan korak učenja odvija se tako da se prvo iz standardne normalne razdiobe uzima matrica koja ima broj redaka određen veličinom podskupa podataka, a broj stupaca brojem varijabli u vektoru **z**. Zatim je potrebno definirati objekte GradientTape klase. Definiraju se dva objekta zato što se odvojeno želi optimizirati funkcija gubitka diskriminativne mreže i funkcija gubitka generativne mreže. Generatorskoj mreži se na ulaz dovodi matrica vektora **z**, uz parametar training postavljen na vrijednost True. Ovo nije potrebno za generatorsku mrežu i postavljeno je zbog konzistentnosti, ali u nekim verzijama diskriminatora tokom rada bit će potrebno zbog određenih slojeva koji su potrebni samo tokom faze učenja mreže. Generatorska mreža generira slike i one se dovode na ulaz diskriminativne mreže. Njegovo predviđanje se sprema u varijablu. Također se dovodi i podskup stvarnih podataka koji je u funkciju poslan kao argument. Rezultati diskriminatora se spremaju u drugu varijablu. Ove dvije varijable klasifikacije lažnih i stvarnih slika daju se kao argumenti prethodno definiranoj funkciji gubitka diskriminatora. Klasifikacija lažnih slika daje se definiranoj funkciji gubitka generatorske mreže budući da generatorskoj mreži nije potreban rezultat klasifikacije stvarnih slika. Nakon izračunatih vrijednosti gubitka računaju se gradijenti pomoću gradient() metode objekta klase GradientTape. Ti gradijenti se na kraju koriste unutar apply_gradients() metode optimizacijskih algoritama kako bi se ažurirale vrijednosti trenutnih parametara. Dekorator @tf.function koji se nalazi na vrhu funkcije koja

definira jedan korak učenja služi za ubrzavanje izvođenja funkcije. Dekorator u Pythonu je funkcija koja mijenja ponašanje druge funkcije. Ovaj dekorator omogućava konstruiranje računskog grafa koji odgovara potrebnim operacijama. Zatim TensorFlow ovu strukturu može koristiti za brže izvođenje funkcije. Ovo je jedna od metoda za ubrzavanje učenja neuronske mreže. Iduća funkcija definira cjelokupni proces učenja i poziva funkciju koja definira jedan korak učenja. Funkciji se predaje cijeli skup podataka i broj epoha. Funkcija prolazi kroz svaku epohu i na početku poziva funkciju clear_session(). Ova funkcija je riješila problem pretjeranog zauzimanja memorije jer ova funkcija oslobađa nepotrebno zauzetu memoriju od prošlog treniranja modela. Prije početka učenja također se sprema trenutno vrijeme kako bi se moglo ispisati provedeno vrijeme učenja za jednu epohu. Unutarnja petlja tokom svake epohe iterira kroz cijeli skup podataka, izdvaja podskup po podskup i predaje ga funkciji zaduženoj za obavljanje jednog koraka učenja. U vanjskoj petlji svake petnaeste epohe spremaju se modeli i njihovi optimizatori pozivanjem metode save() na prijašnje definiranom objektu klase Checkpoint. Na kraju svake epohe ispisuje se koliko je epoha trajala.

```
epochs = 150
noise_dim = 200
batch_size = 128

# define one training step
@tf.function
def train_step(images):
  noise = tf.random.normal([batch_size, noise_dim])

  with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
    generated_images = generator(noise, training = True)

    fake_output = discriminator(generated_images, training = True)
    real_output = discriminator(images, training = True)

    disc_loss = discriminator_loss(fake_output, real_output)
    gen_loss = generator_loss(fake_output)

  gen_gradients = gen_tape.gradient(gen_loss, generator.trainable_variables)
  disc_gradients = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

  generator_optimizer.apply_gradients(zip(gen_gradients, generator.trainable_variables))
  discriminator_optimizer.apply_gradients(zip(disc_gradients, discriminator.trainable_variables))

# define training loop

def train(dataset, epochs):
  for epoch in range(epochs):
    tf.keras.backend.clear_session()
    start = time.time()
    for image_batch in dataset:
      train_step(image_batch)

    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print("epoch {} took {} seconds".format(epoch + 1, time.time() - start))
```

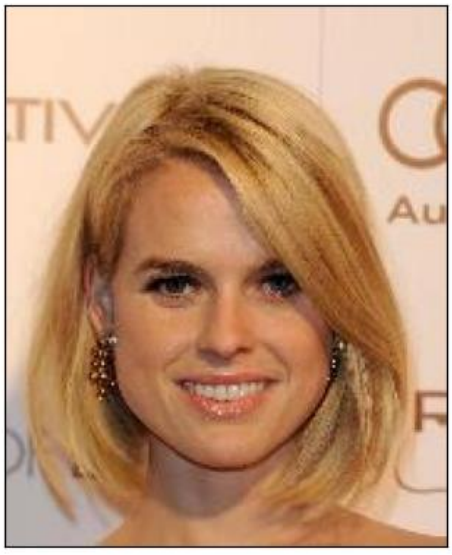
Slika 5.12 Definicija procesa učenja.

6. IZRAĐENI MODELI I REZULTATI

Prvi cilj ovog poglavlja je opisati potrebni kod prije same izgradnje i učenja modela, a zatim prikazati svaki izrađeni model pomoću Tensorflow paketa i objasniti njegovu arhitekturu i istaknuti najvažnije značajke. Drugi cilj poglavlja je za svaki model prikazati i komentirati rezultate. Svi modeli učeni su na vrlo moćnoj Nvidinoj grafičkoj kartici A100, korištenjem Google Colab Pro+ usluge u oblaku.

6.1. Skup podataka

Za učenje mreže korišten je već napravljeni skup podataka koji se sastoji od slika ljudskih lica. Ovaj skup zove se CelebA skup podataka. To je skup podataka koji se sastoji od 202599 slika poznatih osoba. U ovom radu korištena je varijanta ovog skupa podataka gdje su slike poravnate i napravljeno je potrebno rezanje slika. Ne koriste se sve slike za učenje. Broj slika koji je korišten za učenje mijenjan je po potrebi. Razlog za promjenom je svaki puta pojašnjen. Primjer slike iz skupa podataka se može vidjeti na slici 6.1.



Slika 6.1 Primjer slike iz skupa podataka.

6.2. Prvi model

Model diskriminatora je u ovom slučaju konvolucijska neuronska mreža koja ima samo dva konvolucijska sloja i na kraju potpuno povezani sloj. Prvi sloj sadrži 64 maske, koja se sastoji od 5 redaka i 5 stupaca. Pomak maske u oba sloja je za dva piksela. Opisani model vidi se na slici 6.2. Model generatorske mreže vrlo je jednostavan model. Na početku se sastoji od potpuno povezanog sloja na ulazu, gdje je između aktivacije i tog sloja dodana grupna normalizacija. Grupna normalizacija ubrzava učenje mreže i to je glavni razlog dodavanja ove operacije. Prema [17], grupna normalizacija smanjuje unutarnji kovarijantni pomak, što uzrokuje brže učenje. Prema [18], ipak je glavni uzrok ubrzanja učenja neuronskih mreža efekt zaglađivanja funkcije gubitka koji uzrokuje grupna normalizacija. Na sam ulaz dolazi 200 nasumičnih vrijednosti uzetih iz normalne distribucije. Kroz cijelu mrežu koristi se LeakyReLU aktivacijska funkcija za uvođenje nelinearnosti, a također rješava problem nestajućeg gradijenta. Na izlazu iz prve aktivacije, potrebno je napraviti preoblikovanje podataka kako bi se postepeno doveli do formata slike. Pomoću assert naredbe provjerava se je li oblik slike trenutno valjan pri izgradnji modela. Ostali slojevi su transponirana konvolucija, ovim slojevima se izgrađuje slika iz šuma koji dolazi na ulaz modela. Zadnji sloj ima sigmoidalnu aktivacijsku funkciju kako bi se vrijednosti piksela ograničile između nula i jedan, kako bi odgovarale vrijednostima piksela slika koje se nalaze u skupu za učenje. Opisana arhitektura može se vidjeti na slici 6.3.

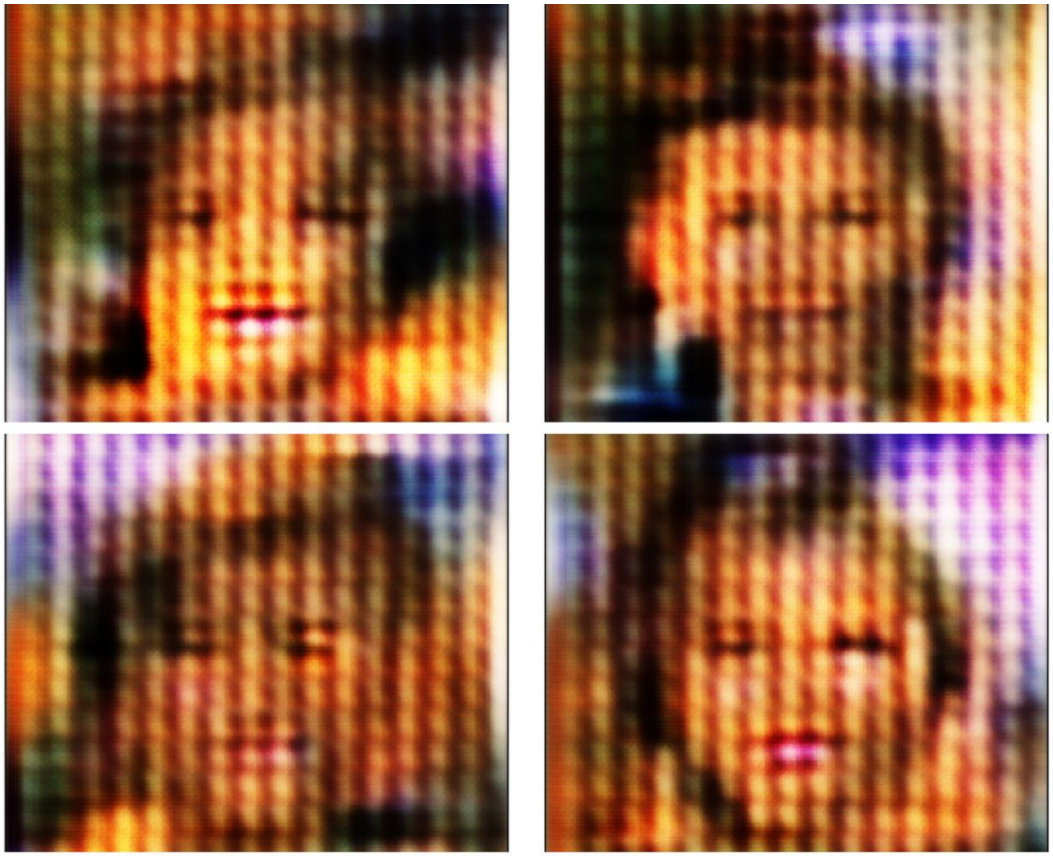
```
def make_discriminator_model():
  model = tf.keras.Sequential()

  model.add(layers.Conv2D(filters = 64, kernel_size = (5, 5), strides = 2,
                          padding = "same", input_shape = (256, 256, 3)))
  model.add(layers.LeakyReLU())

  model.add(layers.Conv2D(filters = 128, kernel_size = (5, 5), strides = 2,
                          padding = "same", ))
  model.add(layers.LeakyReLU())

  model.add(layers.Flatten())
  model.add(layers.Dense(units = 1))

  return model
```

Slika 6.2 Diskriminativni dio prvog modela.

```python
def make_generator_model():
  model = tf.keras.Sequential()

  model.add(layers.Dense(units = 16 * 16 * 256, use_bias = False, input_shape = (200,)))
  model.add(layers.BatchNormalization())
  model.add(layers.LeakyReLU())
  model.add(layers.Reshape((16, 16, 256)))
  assert model.output_shape == (None, 16, 16, 256)

  model.add(layers.Conv2DTranspose(filters = 128, kernel_size = (5, 5), strides = 2,
                                   padding = "same", use_bias = False))
  assert model.output_shape == (None, 32, 32, 128)
  model.add(layers.BatchNormalization())
  model.add(layers.LeakyReLU())

  model.add(layers.Conv2DTranspose(filters = 64, kernel_size = (5, 5), strides = 2,
                                   padding = "same", use_bias = False))
  assert model.output_shape == (None, 64, 64, 64)
  model.add(layers.BatchNormalization())
  model.add(layers.LeakyReLU())

  model.add(layers.Conv2DTranspose(filters = 32, kernel_size = (5, 5), strides = 2,
                                   padding = "same", use_bias = False))
  assert model.output_shape == (None, 128, 128, 32)
  model.add(layers.BatchNormalization())
  model.add(layers.LeakyReLU())

  model.add(layers.Conv2DTranspose(filters = 3, kernel_size = (5, 5), strides = 2,
                                   padding = "same", use_bias = False, activation = tf.keras.activations.sigmoid))
  assert model.output_shape == (None, 256, 256, 3)

  return model
```

Slika 6.3 Generativni dio prvog modela.

Rezultati prvog modela mogu se vidjeti na slici 6.4. Iz slika se može uočiti kako rezultati nisu zadovoljavajući za rješavanje problema generiranja slika lica ljudi. Mogu se razaznati obrisi lica i neke od glavnih značajki. Na slikama se može uočiti ponavljanje određenog uzorka, što prema [1], može ukazivati na podnaučeni model. Cijeli model se učio kroz 70 epoha, a trajanje učenja je bilo 3 minute. Glavna značajka poboljšanja bi trebalo biti povećanje broja podataka za učenje, zbog podnaučenosti modela.



Slika 6.4 Primjer generiranih slika prvog modela.

6.3. Drugi model

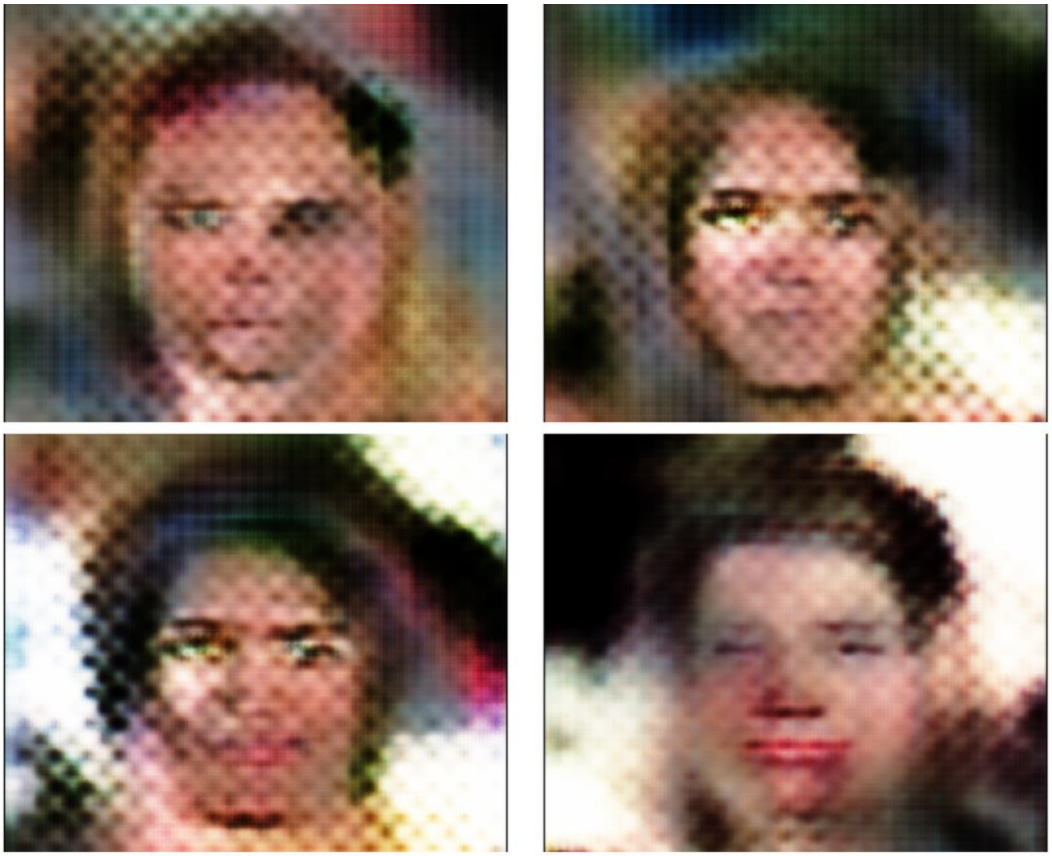
Drugi model raspodijeljen je na tri verzije zato što je arhitektura modela ostala ista kroz verzije modela jedan, dva i tri. Ono što se mijenja po verzijama su postavke kao broj epoha i broj podataka koji se koristi za učenje. Glavna promjena u arhitekturi drugog modela je dodavanje sloja isključivanja neurona u diskriminator s 20% isključivanja neurona. Generatorska mreža je ostala ista kao i u prvom modelu. Prema [20], isključivanje neurona djeluje tako da se određeni neuroni privremeno isključe iz procesa učenja zajedno sa svojim ulazećim i izlazećim vrijednostima. Ova metoda također ima utjecaj regularizacije što smanjuje efekt prenaučenosti. Uz povećanje broja podataka kroz verzije drugog modela, upravo ovo je potrebno za poboljšanje prvog modela. Definicija diskriminatora vidi se na slici 6.5.

```
# define discriminator model
def make_discriminator_model():
  model = tf.keras.Sequential()

  model.add(layers.Conv2D(filters = 64, kernel_size = (5, 5), strides = 2,
                          padding = "same", input_shape = (256, 256, 3)))
  model.add(layers.LeakyReLU())

  model.add(layers.Conv2D(filters = 128, kernel_size = (5, 5), strides = 2,
                          padding = "same", ))
  model.add(layers.LeakyReLU())

  model.add(layers.Flatten())
  model.add(layers.Dropout(0.2))
  model.add(layers.Dense(units = 1))

  return model
```

Slika 6.5. Diskriminator drugog modela.

6.3.1. Model 2.1

U modelu 2.1 broj slika za učenje povećao se na 10130. Također se je povećan broj epoha na 100. Promjene u arhitekturi diskriminatora i promjene ova dva parametra rezultirale su boljim slikama, ali i povećanjem vremena učenja na 16 minuta. Rezultati ove generatorske mreže mogu se vidjeti na slici 6.6. Kao i po rezultatima prvog modela još uvijek se može uočiti ponavljanje jedne teksture na slici što ukazuje na podnaučenost, ali ne u toliko velikoj mjeri kao kod prvog modela. Slike nisu zadovoljavajuće, ali uspoređujući slike sa slikama koje je generirao prvi model mogu se uočiti jasnije oči ljudi, kao i detaljnije usne. Također oblik glave se može bolje raspoznati od ostatka okoline na slici.



Slika 6.6 Primjer generiranih slika modela 2.1.

6.3.2. Model 2.2

U modelu 2.2 povećan je broj podataka za učenje kako bi se pokušala smanjiti podnaučenost modela. U ovom modelu koristi se 81040 slika. Također je smanjen broj epoha na 30. Model 2.2 učio se 38 minuta. Smanjenje epoha nije bio potreban korak, jer kao što će se vidjeti model 2.3 postići će bolje rezultate od modela 2.2 s istim brojem slika, a brojem epoha kao i model 2.1. Rezultati ovog modela mogu se vidjeti na slici 6.7. Lica se puno jasnije prepoznaju nego na prijašnjim slikama. Također su bolje vidljivi detalji lica kao što su nos, oči, usne, kosa i obrve. Još uvijek se može vidjeti ponavljanje određenih uzoraka na slici.

Slika 6.7 Primjer generiranih slika modela 2.2.

6.3.3. Model 2.3

U modelu 2.3 broj epoha je vraćen na 100, nisu se radile druge promjene. Ovo je uzrokovalo povećanje vremena učenja na 60 minuta, ali i poboljšanje rezultata koji se mogu vidjeti na slici 6.8. Na slikama se više ne vidi uzorak koji se ponavlja, ali generatorska mreža još uvijek generira vrlo pojednostavljene slike ljudskih lica. Ovo može ukazivati na nedovoljni kapacitet diskriminativne mreže koja će generatorskoj mreži omogućiti ovakvu slobodu. Odnosno, diskriminator neće dobro estimirati omjer distribucija dan izrazom (4-5) i na taj način će generatorska mreža unutar funkcije pogreške imati krivu informaciju o uspješnosti učenja. Zato je u idućem modelu potrebno opet mijenjati arhitekturu diskriminatora kako bi mu se povećao kapacitet.

Slika 6.8 Primjer generiranih slika modela 2.3.

6.4. Treći konačni model

U ovom modelu povećan je kapacitet modela tako što se diskriminatoru dodao još jedan konvolucijski sloj. Ulazni konvolucijski sloj sada ima 32 maske, dok drugi sloj ima 64 maske. Ovim povećanjem broja slojeva povećan je i broj operacija što je uzrokovalo povećanje vremena učenja na 102 minute. Diskriminator se može vidjeti na slici 6.9., a rezultati ovog modela mogu se vidjeti na slici 6.10. Lica ljudi sada su vrlo prepoznatljiva uz kompleksnije detalje koji se mogu uočiti na licima. Također se mogu uočiti sjene i osvjetljenja. Slike još uvijek ne izgledaju u potpunosti kao stvarne slike ljudi, ali u opsegu ovog rada možemo reći da su postignuti željeni rezultati.

```python
# define discriminator model
def make_discriminator_model():
  model = tf.keras.Sequential()

  model.add(layers.Conv2D(filters = 32, kernel_size = (5, 5), strides = 2,
                          padding = "same", input_shape = (256, 256, 3)))
  model.add(layers.LeakyReLU())

  model.add(layers.Conv2D(filters = 64, kernel_size = (5, 5), strides = 2,
                          padding = "same"))
  model.add(layers.LeakyReLU())

  model.add(layers.Conv2D(filters = 128, kernel_size = (5, 5), strides = 2,
                          padding = "same", ))
  model.add(layers.LeakyReLU())

  model.add(layers.Flatten())
  model.add(layers.Dropout(0.2))
  model.add(layers.Dense(units = 1))

  return model
```

Slika 6.9. Diskriminator trećeg modela.

Slika 6.10 Primjer generiranih slika trećeg modela.

6.5. Usporedba modela

Tablica 6.1 Usporedba svih modela.

	Model 1	Model 2.1	Model 2.2	Model 2.3	Model 3
Broj epoha	70	100	30	100	100
Broj slika	3039	10130	81040	81040	81040
Vrijeme učenja	3 minute	16 minuta	38 minuta	60 minuta	102 minute
rezultati					
Model 1			Model 2.2		
Model 2.1			Model 2.3		
Model 3					

U tablici 6.1 može se vidjeti usporedba svih modela u ovom radu. Broj korištenih slika u procesu učenja raste s novijim modelima. Također raste i vrijeme učenja zbog većeg broja podataka, ali i zbog kompleksnosti modela. Broj epoha koji je najviše korišten je 100. Jasan napredak kroz poboljšanje modela se također vidi u tablici u dijelu gdje su pokazani svi rezultati modela jedan pored drugog. Pogotovo se vidi napredak u generiranim slikama ako se usporede rezultati prvog modela i konačnog modela koji daje ciljane rezultate ovog rada.

7. ZAKLJUČAK

Cilj ovog rada bilo je napraviti model generativnih suparničkih mreža, kojim će se moći naučiti generatorsku mrežu koja će proizvoditi zadovoljavajuće slike ljudskih lica. Na početku rada je objašnjena teorijska podloga neuronskih mreža, konvolucijskih neuronskih mreža, generativnih suparničkih mreža i konačno objašnjena je osnovna teorija konvolucijskih suparničkih generativnih mreža. Zatim su pojašnjeni paketi koji se koriste unutar Python programskog jezika i prikazan je skup podataka čiji su se djelovi koristili za učenje modela. Zadnji dio ovog rada, koji je i glavni dio rada, sastoji se od prikaza postavki koje su bile potrebne za učenje i izgradnju modela kao što je funkcija koja upravlja učenjem modela ili funkcija koja upravlja jednim korakom učenja. Nakon prikaza postavki, izložena je arhitektura prvog modela iz kojeg su se onda gradili i ostali modeli. Odnosno prvi model je poboljšavan u samoj arhitekturi, ali i u svojstvima kao što je broj epoha i broj podataka za učenje. Također su prikazani rezultati svakog modela, opisani su nedostatci tih rezultata i napredak s obzirom na prošle modele. Konačno je napravljena tablica usporedbe svih modela i njihovih rezultata. Pokazalo se da je bilo više puta potrebno povećati broj podataka kako bi se izbjegla podnaučenost modela. Također je korištena metoda isključivanja neurona iz učenja što je uzrokovalo regularizaciju, odnosno bolja svojstva generalizacije i izbjegavanje prenaučenosti. Konačna promjena arhitekture je bilo povećanje kompleksnosti diskriminatora kako bi bolje pomogao u učenju generatorske mreže i tako doveo do kvalitetnijih i realnijih slika ljudskih lica. Bilo je potrebno napraviti tri modela, uz tri verzije drugog modela. Konačni rezultati ne izgledaju potpuno isto kao stvarne slike ljudskih lica. Ipak, vrlo je jasno da su to ljudska lica i slike imaju određenu razinu kompleksnosti i detalja lica. Zbog ovih svojstava slike dobivene trećim modelom su uzete kao konačni rezultati ovog rada. Rezultati bi se mogli poboljšati uz dulji pristup boljem sklopovlju. Time bi se skup podataka za učenje mogao povećati, kao i kompleksnost modela, a brzina učenja bi se povećala i to bi omogućilo testiranje više različitih modela. Također bi se mogli razmotriti još neki postupci biranja hiperparametara modela, kao što je na primjer promijenjiva stopa učenja koja bi se postepeno smanjivala. Za uočavanje nestabilnosti mogla bi se pratiti promjena funkcija gubitka tokom učenja.

LITERATURA

[1] A. Radford., L. Metz., S. Chintala., unsupervised representation learning with deep convolutional generative adversarial networks, International Conference on Learning Representations, San Juan, 2016.

[2] I. Goodfellow, Generative Adversarial Networks, Neural information processing systems, Barcelona, 2016.

[3] T. Karras, S. Laine, T. Aila, A Style-Based Generator Architecture for Generative Adversarial Networks, IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), str. 4396 – 4405, Long Beach, 2019.

[4] T. Karras, S.Laine, T. Aila, J. Lehtinen, Progressive Growing Of Gans For Improved Quality, Stability, And Variation, International Conference on Learning Representations, Vancouver, 2018.

[5] A. Brock, J. Donahue, K. Simonyan, International Conference on Learning Representations, New Orleans, 2019.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adverserial Nets, Neural information processing systems, Montréal, 2014.

[7] M. O. Jackson, A Brief Introduction to the Basics of Game Theory, Social Science Research Network, prosinac 2011.

[8] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT press, Cambridge, 2016.

[9] S. Ruder, [1609.04747] An overview of gradient descent optimization algorithms, Cornell University, Galway, 2016., dostupno na: https://doi.org/10.48550/arXiv.1609.04747

[10] I. S. Pandžić, A. Bažant, Ž. Ilić, Z. Vrdoljak, M. Kos, V. Sinković, Uvod u teoriju informacija i kodiranja, Element, Zagreb, 2012.

[11] J. Shlens, [1404.2000] Notes on Kullback-Leibler Divergence and Likelihood, Cornell University, Mountain View, 2014., dostupno na: https://doi.org/10.48550/arXiv.1404.2000

[12] R. J. Moss, [2009.09043] Cross-Entropy Method Variants for Optimization, Cornell University, Stanford, 2020., dostupno na: arXiv:2009.09043

[13] C. M. Bishop, Pattern recognition and machine learning, Springer, New York, 2006.

[14] S. R. Dubey, S. K. Singh, B. B. Chaudhuri, Activation functions in deep learning: A comprehensive survey and benchmark, Neurocomputing, br. 7, sv. 503, str. 92-108, rujan 2022.

[15] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, Neural information processing systems, Lake Tahoe, 2012.

[16] J. Wu, Introduction to Convolutional Neural Networks, National Key Lab for Novel Software Technology. Nanjing University. China, br. 23, sv. 5, str.495, svibanj 2017.

[17] S. Ioffe, C. Szegedy, [1502.03167] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Cornell University, Google inc., 2015., dostupno na: arXiv:1502.03167

[18] S. Santurkar, D. Tsipras, A. Ilyas, A. Madry, [1805.11604] How Does Batch Normalization Help Optimization?, Cornell University, Cambridge, 2019., dostupno na: arXiv:1805.11604

[19] François Chollete, Deep Learning with Python, Manning Publications Co, Shelter Island, 2018.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskevar, R. Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research, sv. 15, str. 1929−1958, lipanj 2014.

SAŽETAK

Glavni zadatak rada je bio napraviti model suparničkih generativnih mreža čija će generatorska mreža nakon učenja uspješno generirati slike ljudskih lica. Na početku je objašnjena sva potrebna teorija za shvaćanje modela koji su u radu napravljeni i korišteni. Nakon toga je prikazana potrebna priprema za sam proces izgradnje modela i njihovog učenja. Na kraju su svi modeli objašnjeni i pojašnjeni su razlozi za promjene koje su napravljene na modelima da bi se dobili bolji rezultati. Također su prikazani rezultati i objašnjeni su nedostatci rezultata i kako pristupiti poboljšanju. Poboljšanja koja su primijenjena u radu kako bi se postigli željeni rezultati proizlaze iz teorije strojnog učenja, točnije metoda iz dubokog učenja koje su se pokazale kao dobre metode za rješavanje problema koji se pojavljuju kroz rad. Svako poboljšanje je objašnjeno i naveden je jasan razlog zašto je napravljeno i kako bi teoretski trebalo utjecati na rezultate modela. Budući da su slike kroz verzije modela znatno bolje može se zaključiti da ovi postupci nisu učinkoviti samo u teoriji već su pomogli u generiranju kvalitetnijih slika.

Ključne riječi: generativne suparničke neuronske mreže, generiranje slika ljudskog lica, tensorflow, strojno učenje, umjetna inteligencija

GENERATING ARTIFICIAL IMAGES OF FACES USING GENERATIVE ADVERSARIAL NETWORKS

The main task of this paper was to create a model of adversarial generative networks whose generator network after learning will successfully generate images of human faces. At the beginning, all the necessary theory for understanding the models that were created and used in the paper was explained. After that, the necessary preparation for the process of building the models and their learning is presented. At the end, all models are explained and the reasons for the changes made to the models for the purpose of getting the better results are clarified. The results are also presented. The shortcomings of the results and how to approach the improvements are explained. The improvements that have been applied in this paper to achieve the desired results stem from machine learning theory, more specifically deep learning methods that have proven to be good methods for solving problems that arise through this work. Each improvement is explained and a clear reason is given why it was made and how it should theoretically affect the results of the model. Because the images with different model versions are significantly better it can be concluded that these procedures are not efficient only in theory but also that they helped to generate higher quality images.

Keywords: generative adversarial neural networks, human facial image generation, tensorflow, machine learning, artificial intelligence