SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

Uklanjanje šuma iz slika primjenom dubokih neuronskih mreža

Završni rad

Dijana Ivezić

Osijek, 2020.

Sadržaj

1. UVOD

Šum na slici je bilo koja nasumična varijacija svjetline slike koja smanjuje kvalitetu slike ili onemogućava očitavanje željenih podataka. Predstavlja neželjeni signal koji je dodan čistom signalu slike. Slika može imati toliko šuma da su detalji ili čak cijela slika neprepoznatljivi. Prilikom fotografiranja ne dobije se slika bez šuma, no za svakodnevne potrebe to nije toliko važno. Značajno je u znanstvenom radu gdje čista slika (slika bez šuma) može spasiti nečiji život ili otkriti nam nekakve nove spoznaje. Neka rješenja za uklanjanje šuma samo zamute sliku i ti važni detalji se više ne vide; gubimo značajne informacije zbog kojih je ta slika snimljena. Primjeri takvih rješenja su medijan filter ili Gaussovo izglađivanje. Računala se mogu naučiti da otklanjaju šum na drugačiji način, primjenom dubokog učenja. Duboko učenje je grana znanosti umjetne inteligencije koja nam omogućava računalno razumijevanje govora, prepoznavanje slika, donošenje odluka na temelju podataka i sl. Nešto što bi osobi bilo lako za napraviti, računalima je vrlo teško izvršiti. Ako pokušamo intuitivno programirati računalo da prepoznaje lica ili objekte sa slike, vidjeli bi kako bi to bilo gotovo nemoguće bez upotrebe znanja i vještina iz grane znanosti duboko učenje. Duboko učenje je dobilo svoje ime po tome što slojevi korištene neuronske mreže idu u dubinu, u više slojeva koje možemo shvatiti tako da zamislimo kako je svaki sloj zadužen da prepoznavanje određenih značajki slika. Takvih značajki ima mnogo. Ako želimo napraviti sustav za prepoznavanje životinja sa slika, više slojeva bi bilo zaduženo za više vrsta očiju, neki bi bili zaduženi za prepoznavanje različitih vrsta krzna i sl. Znamo kako je životinjski svijet raznolik i intuitivno je jasno da takva mreža može samo rasti kako bi dobro obavila svoj posao. Ovaj rad se bazira na korištenju dubokog učenja u svrhu uklanjanja šuma iz slika. Zamislimo li sliku sa šumom, vidimo da dijelove slike možda uopće ni ne vidimo ispod smetnji. Te dijelove moramo pretpostaviti kako bi na kraju dobili čistu i cjelovitu sliku. Pretpostavljamo ih pomoću ostatka slike i to slijepo ili ne slijepo. Kod ne slijepog popunjavanja slike algoritam ne zna točno gdje se nepravilnosti nalaze što zahtjeva ljudski rad. Slijepo popunjavanje zahtjeva od algoritma automatsko prepoznavanje šuma koje treba otkloniti čime je taj sustav složeniji, no zanimljiviji sa točke automatizacije uklanjanja šuma što je vrlo važno kod velikog broja slika. Duboko učenje uči parametre sustava iz podataka bez ljudske intervencije. Ovaj rad se bazira na dvije vrste šuma: aditivni bijeli Gaussov šum i šum soli i papra. Aditivni bijeli Gaussov šum je nasumični šum koji pobliže opisuje razne smetnje koje se mogu dogoditi prilikom dohvaćanja slika. Šum soli i papra predstavlja skokove intenziteta na slici što je obično uzrokovano pogreškama u prijenosu podataka. Oštećeni pikseli imaju ili minimalnu ili maksimalnu vrijednost, dajući slici izgled soli i

papra. Kako bi se uspjeh modela za uklanjanje šuma prikazao kvantitativno koristi se PSNR (engl. *Peak Signal to Noise Ratio*). Što je veći PSNR, to je bolja kvaliteta rekonstruirane slike.

U idućem poglavlju je prikazan trenutni znanstveni doseg iz područja teme, a u 3. poglavlju slijedi teorijska podloga zadatka i korištene tehnologije, dok u 4. poglavlju je predstavljeno uklanjanje šuma iz slika u programskom jeziku Python koristeći aplikacijsko korisničko sučelje Keras i U-Net arhitekturu.

1.1. Zadatak završnog rada

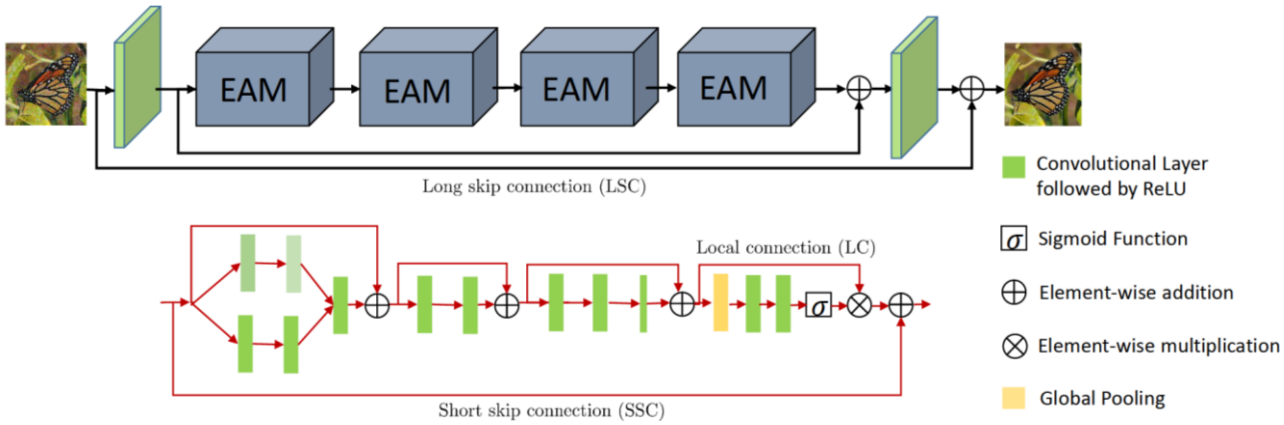
Koristeći programski jezik Python i aplikacijsko korisničko sučelje Keras razviti model koji će ukloniti dodani bijeli Gaussov šum i šum soli i papra različitih intenziteta sa slika iz baza slika CIFAR-10 i MNIST. Prikazati PSNR za razne intenzitete šuma i primjere slika prije šuma, sa i bez šuma. Predstaviti trenutno stanje teme, te objasniti principe dubokog učenja.

2. PREGLED PODRUČJA TEME

2.1 Uklanjanje šuma iz stvarnih slika sa pozornošću na značajke

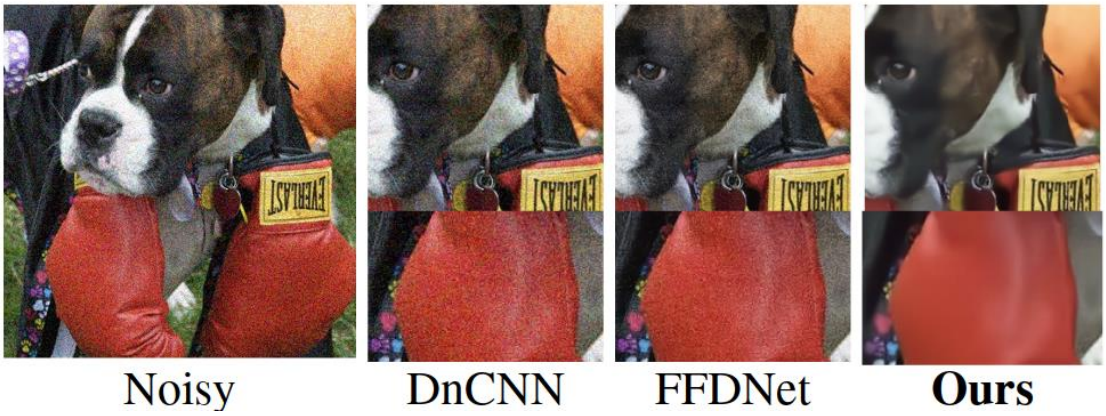
(engl. *„Real Image Denoising with Feature Attention“* [1])

Po Anwar i Barnes [1] modeli neuronskih mreža često dobro rade na kreiranim podacima prilikom treniranja, podaci koji su dobiveni umjetnim dodavanjem šuma na čistu sliku, te su manje uspješni na stvarnim podacima zbog čega su mreže i trenirane. Praktičnost takvih mreža je smanjena na što su se bazirali u svom radu predstavljajući mrežu slijepog uklanjanja šuma RIDNet. Uvode model koji osigurava vrhunske rezultate koristeći modularnu mrežu gdje povećanje broja modula pomaže poboljšati performanse, za razliku od većine dotadašnjih rješenja.



Slika 2.1: Predložena arhitektura [1]

Kazalo na desnoj strani slike 2.1 redom prikazuje: konvolucijski sloj popraćen ispravljačem, sigmoid funkcija, zbrajanje elemenata u matrici direktno element sa elementom, množenje elemenata u matrici direktno element sa elementom, globalno udruživanje. Gornji dio slike prikazuje suštinu rada modela. Zeleno su označeni moduli za ekstrakciju svojstava i rekonstrukciju; na početku i na kraju. Oni su sačinjeni od samo jednog konvolucijskog sloja. Prateći strelice, vidimo da nakon ekstrakcije svojstava dolaze EAM (engl. *„enhancement attention modules“*) blokovi koji su glavni dio zasebnog modula učenja značajki i njihova uloga je poboljšanje pozornosti na značajke cijelog modela. Vidimo da su EAM blokovi prikazani jedan nakon drugog i u predloženoj mreži ih ima 4, moguće je dodati još radi poboljšanja izvedbe. Dakle, mreža podržava proširivanje tvoreći tako po potrebi jako duboku mrežu. [1]



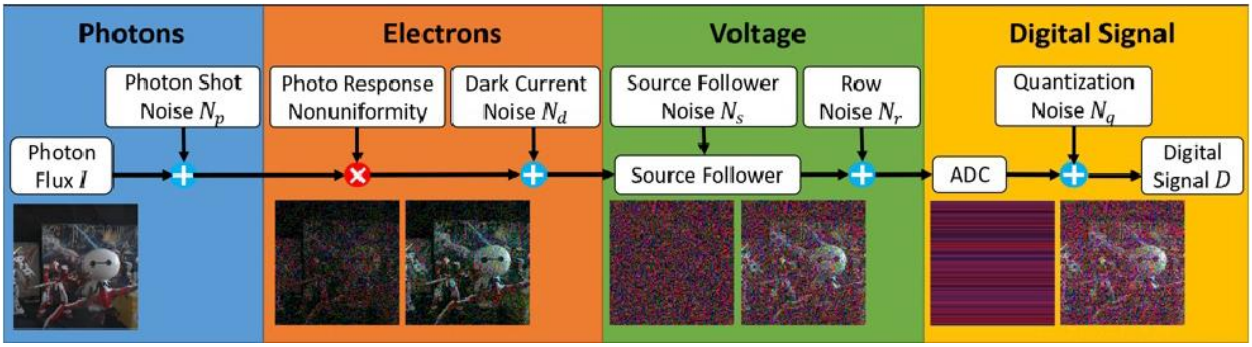
Slika 2.2: Usporedba sa drugim arhitekturama [1]

Usporedba predstavljenog modela sa ostalim modelima (predstavljeni model je prikazan pod *„Ours”*, posljednja slika) je prikazana na slici 2.2. Prva slika (prikazana pod *„Noisy”*) predstavlja sliku sa šumom. Slika je iz RNI15 baze podataka [1]. Uspjeh predložene mreže je jasno vidljiv, posebno u detaljima koji su prikazani na slici. U njihovom radu [1] prikazane su detaljnije usporedbe, kao i PSNR naspram drugih rješenja.

2.2 Model kreiranja šuma baziran na fizici za uklanjanje šuma iz ekstremno nisko osvjetljenih slika

(engl. *„A Physics-based Noise Formation Model for Extreme Low-light Raw Denoising”* [2])

Naučeni algoritmi uklanjanja šuma iz slika nailaze na poteškoće u upotrebi na stvarnim slikama iz fotoaparata ili drugih izvora jer su trenirani na slikama za trening koje ne prate dobro stvaran šum. Problematika kojima se istraživači [2] bave je modeliranje šuma koje pobliže prati šum na stvarnim slikama, ponajviše šum uzrokovan elektronikom u digitalnim aparatima i šum koji se javlja pri niskom osvjetljenju. Jasno je da je skupljanje stvarnih podataka dugotrajan i skup posao, te da kvaliteta modeliranih podataka uvelike utječe na iskoristivost sustava treniranih tim podacima na stvarnim slikama. Proučavajući fizikalne procese stvaranja stvarne slike, napravili su bolji model dodanog šuma na slikama koji se bazira na CMOS foto senzorima koji dobro opisuje stvarni šum. Dobiveni podaci pokazuju da su postojeći modeli uklanjanja šuma iz slika, trenirani na dobivenim realističnijim podacima, uspješniji. Wei, Fu, Yang i Huang [2] također uvode podešavanje parametara stvaranja njihovog šuma kako bi što bliže opisivao šum nastao na kameri odabranog proizvođača. [2]



Slika 2.3: Vizualizacija šuma, te prikaz krajnje slike u pojedinim etapama [2]

Na slici 2.3 su vidljive 4 različite etape pri kojima se na fotone iz prirode unutar digitalne kamere neizbježno dodaju različite vrste šuma. U plavom dijelu na priljev fotona u leću kamere se dodaje šum pucanja zbog prirode čestica svjetlosti. U narančastom dijelu, koji se odnosi na elektrone, signal iz prethodne etape se množi sa veličinom koja opisuje grešku u signalu koji izlazi iz senzora te se dodaje šum takozvane tamne struje koja nastaje u senzorima svjetlosti onda kada fotona zapravo nema. Zeleni dio prikazuje dodatak šuma izazvanog nasumičnim kretanjem elektrona u vodiču što izaziva promjene napona. Posljednji dio predstavlja dodatak šuma uzrokovanog kvantizacijom u analogno-digitalnom pretvaraču.

2.3 ŠumDoNičega

(engl. *„Noise2Void“* [3])

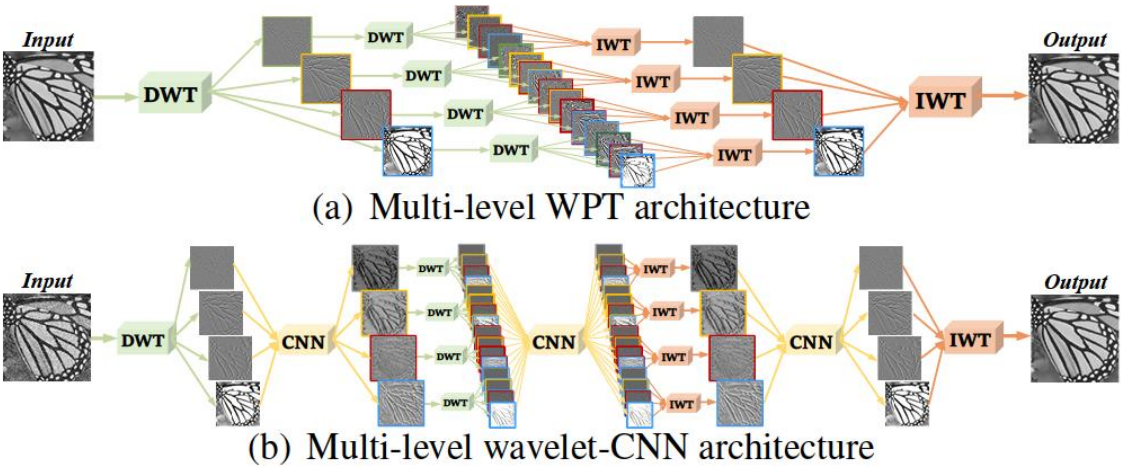
Duboke neuronske mreže se tradicionalno treniraju korištenjem dva skupa slika. Jedan skup slika sadrži čiste slike i te slike se shvaćaju kao apsolutna istina, koriste se kao referenca pri treniranju. Drugi skup slika su slike sa šumom koji želimo onda ukloniti. Nekada je teško, skupo ili čak nemoguće doći do čistih slika sa kojima bi trenirali mrežu, te takve mreže imaju svoja ograničenja. Zbog tih ograničenja, istraživači [4] su predložili model mreže koja se trenira sa dva skupa sa slikama sa šumom. Ta dva skupa ne sadrže jednake slike, nego je šum na slikama iz prvog skupa drugačiji od šuma na slikama iz drugog skupa. Takav pristup je nazvan ŠumDoŠuma (engl. *„Noise2Noise“*). To je odlično ukoliko možemo više puta snimiti sličnu sliku, te prilikom dohvaćanja tih slika nasumičnim fizikalnim procesima dobiti različite šumove koji bi morali biti jedina razlika između slika ta dva skupa. Idući istraživači [3] imaju još manje zahtjeva tako što im je potreban samo jedan skup za treniranje, skup sa slikama sa šumom. Jasno im je da je nekada nemoguće dobiti dvije slike koje se razlikuju samo u šumu ili da je čak općenito nemoguće dobiti

dvije slike. Vide primjenu u medicini ili slikama iz laboratorija gdje je važno ukloniti šum iz te jedne dostupne slike, ne postoji čista verzija te slike ili verzija sa drugačijim šumom. Ističu kako je njihova verzija nekada manje uspješna od prijašnjih rješenja, no kako je razlika vrlo mala. Ta razlika je i shvatljiva zbog toga što njihova mreža jednostavno radi sa manje dostupnih informacija. Model su nazvali ŠumDoNičega (engl.*"Noise2Void"*). [3] Nadam se kako će se njihovo istraživanje dalje razgranati u još bolje metode uklanjanja šuma zbog obećavajuće primjene na razne grane znanosti.

2.4 Konvolucijska neuronska mreža bazirana na valovima konačnoga trajanja za restauraciju slika

(engl. *„Multi-level Wavelet-CNN for Image Restoration"* [5])

Prilikom dizajniranja modela neuronskih mreža, znanstvenici često moraju odlučiti između duljine izvođenja i treniranja mreže i količine detalja koje žele sačuvati u obrađenoj slici. Kako bi povećali količinu detalja na slici, istraživači često čine svoje mreže dubljima; imaju više slojeva ili povećavaju veličinu filtera koji se primjenjuju na sliku prilikom treniranja. Obje metode imaju za posljedicu produljivanje trajanja programa. U svom radu istraživači [5] pokušavaju barem djelomično ukloniti taj problem predstavljajući svoj novi model koji jako dobro čuva teksture u obrađenim slikama, te ima visok PSNR. Duljina izvođenja pri predloženoj arhitekturi, jasno nije najbrža do sada, no cilj je nekakva ravnoteža između očuvanih značajki i brzine izvođenja. Glavna ideja je korištenje diskretne transformacije valovima konačnog trajanja (engl. *„discrete wavelet transform"*), koja je slična Fourierovoj transformaciji, no umjesto da signal predstavimo kao sumu beskonačnih sinusoida, koristimo signale kratkog trajanja u točno određenim trenutcima određenih frekvencija. Takva transformacija je reverzibilna. Njihov model se može koristiti ne samo za uklanjanje šuma, nego i za povećanje rezolucije slike i uklanjanje nedostataka pri JPEG kompresiji.



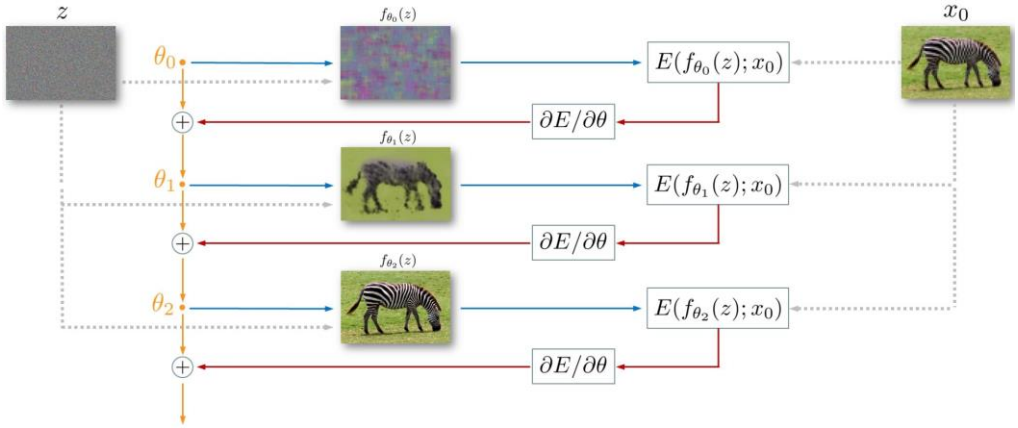
Slika 2.4: Prikaz diskretne transformacije valovima (a) i predložena arhitektura (b) [5]

Dobivaju se 4 slike koje predstavljaju prethodnu primjenom diskretne transformacije valovima i čine jedan sloj. Na tim novim slikama se opet može primijeniti transformacija dobivajući još 4 manje slike (slika 2.4 (a)). [5] Dobivene slike predstavljaju važne značajke originalne slike na kojima se mreža trenira. Slika 2.4 (a) također prikazuje reverzibilnost diskretne transformacije valovima. Predložena arhitektura je bazirana na U-Net arhitekturi s tim da koriste diskretnu transformaciju valovima umjesto originalno predloženih metoda za transformaciju slike. Nakon svake primjene transformacije, nalazi se konvolucijska neuronska mreža (slika 2.4 (b)). [5]

2.5 Prije duboke slike

(engl. *„Deep Image Prior“* [6])

Ulyanov et al. [6] smatraju kako duboka neuronska mreža može bez treniranja uvidjeti značajke slike, te kako uspjeh mreže ovisi o prilagodbi arhitekture mreže podacima. Kod mreža koje se treniraju, parametri mreže su postavljeni na nasumične vrijednosti i mijenjaju se računanjem iz informacija sa velike količine slika. Naučene mreže dobro odrađuju svoj posao na slikama sličnima onima na kojima su trenirani. Predloženi model koristi jednu sliku kako bi mreža iz nasumično postavljenih parametara očitala nove parametre kojima se dobiva nova poboljšana slika u vidu uklanjanja šuma ili povećanja rezolucije. Autori iskorištavaju strukturu mreže kao izvor informacija, te pokazuju jako dobre rezultate bez potrebe za velikim brojem slika na kojima se trenira. [6]



Slika 2.5: Prikaz temeljne ideje modela [6]

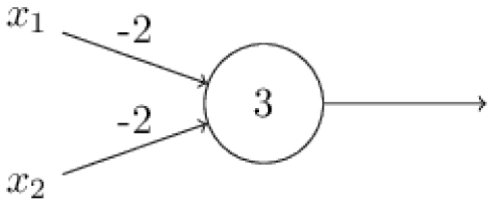
Prilikom svake iteracije neuronske mreže tražimo minimalnu razliku između dobivene slike iz mreže i početne slike počevši od mreže sa nasumičnim parametrima. Dobivene parametre smanjivanjem tih razlika koristimo u oslikavanju nove slike dok ne dobijemo zadovoljavajući rezultat. x_0 označava početnu sliku, $f_\theta(z)$ označavaju dobivene slike sa različitim parametrima mreže, z označava fiksni tenzor na kojem vršimo oslikavanje novih slika. E označava razlike između slika, a θ označava parametre mreže. [6] Navedeno je prikazano na slici 2.5.

3. TEORIJSKA PODLOGA I KORIŠTENE TEHNOLOGIJE

3.1 Neuronske mreže

3.1.1 Ideja neurona

Poznato je da računala rade sekvencijalno izvršavajući naredbu po naredbu već napisanog koda. Te instrukcije računalu piše čovjek koji je problem opisao algoritmom i taj algoritam prilagodio tom računalu. Smisliti algoritam za računanje zbroja dva broja nije teško, svima nam je intuitivan, no kako bi objasnili računalu da prepozna objekt sa slike ili da prepozna dio slike na kojem se nalazi šum? Ako razmislimo o tome, vidimo da je tako nešto teško egzaktno opisati zbog vrlo promjenjive prirode takvih problema. Dakle, raspoznati objekt sa slike čovjeku je u pravilu jednostavan zadatak, no algoritam dolaska do rješenja ne može opisati. Možemo reći da je raspoznati taj objekt čovjek jednostavno naučio. Učimo iz prijašnjih iskustava, ohrabrenja drugih i dobivenih rezultata; naučili smo da računala također mogu učiti. Proučavajući vlastiti mozak i živčani sustav dolazimo do novih saznanja koja nam omogućavaju unaprjeđenje tehnologije. Vodeći se idejom bioloških neurona dolazimo do tehničkih neurona kojima simuliramo rad našeg mozga, time i učenje. Tehnički neuroni (slika 3.1) nisu ni približno kompliciran kao oni u našem tijelu, no i ta pojednostavljena verzija daje izvanredne rezultate. Općenito, neuroni relevantni za ovaj rad imaju ulaz koji je vektor, te izlaz koji je skalar. Ti ulazi se moraju akumulirati na neki način kako bi došli do jednog broja na izlazu, što se postiže zbrajanjem. Cijelo pamćenje mreže je prikazano u težinama svakog ulaza, brojevi kojima se svaki ulaz množi, koje predstavljaju važnost pojedinog ulaza u danom neuronu. Rezultat zbroja vrijednosti sa težinama može i ne mora nam biti dovoljno dobar, to odlučujemo brojem koji nazivamo prag (engl. „*threshold*“).



Slika 3.1: Primjer neurona [7]

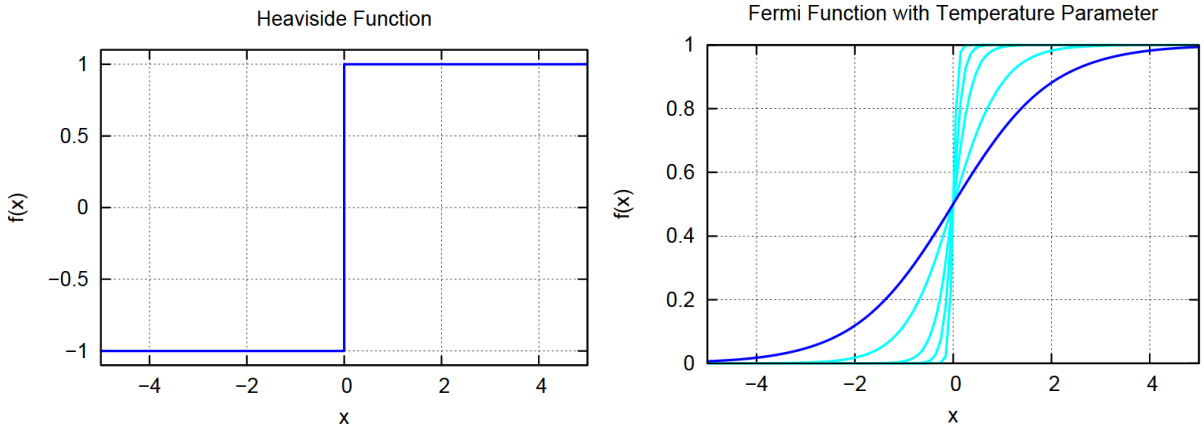
Na ovom jednostavnom primjeru (slika 3.1) prikazani su ulazi x_1 i x_2, težine -2 za svaki ulaz, te prag 3. Koristeći 0 ili 1 za x_1 i x_2 dobivamo rezultate koji jasno ukazuju da neuron predstavlja logički sklop NI. Ako kao ulaze koristimo 0 i 0, kada svaki ulaz pomnožimo sa -2 i dodamo prag od 3, kao rezultat dobivamo 3, što je pozitivan broj i označava u ovom slučaju logičku

jedinicu. Negativan rezultat bi označavao logičku nulu. I za ostale kombinacije brojeva 0 i 1 za ulaze dobili bi dobre rezultate sa ovim modelom logičkog sklopa. Baš kao i logičke sklopove, možemo kombinirati neurone kako bi postigli željene rezultate; složenije sklopove. [7]

3.1.2 Dijelovi neuronske mreže

Neuronska mreža se sastoji od neurona povezanih vezama koje imaju neku težinu. Izlaz iz neurona y za svaki ulaz x označavamo sa $y = f_{act}(z)$, a $z = \sum_i w_i x_i + b$ gdje w_i označavaju težine veze sa danim ulazom x_i, b označava prag na neuronu, a funkcija f_{act} aktiviranost neurona; je li neuron za dani ulaz aktivan ili ne. Zovemo ju aktivacijska funkcija. z označava ulaz u neuron kojemu je dodan prag. Ulaz u neuron označiti ćemo sa $net = \sum_i w_i x_i$ i predstavlja sumu ulaza pomnoženih sa odgovarajućim težinama. Kako bi mreža mogla učiti, važno je da ulazi x_i poprimaju vrijednosti između 0 i 1, kao i izlaz iz neurona y. Želimo omogućiti malim promjenama težina ili praga malu promjenu na izlazu, također je važno matematički opisati te promjene; derivacije. Nederivabilnu step funkciju od 0 do 1 najčešće aproksimiramo sigmoidnom funkcijom koja ima oblik: [7,8]

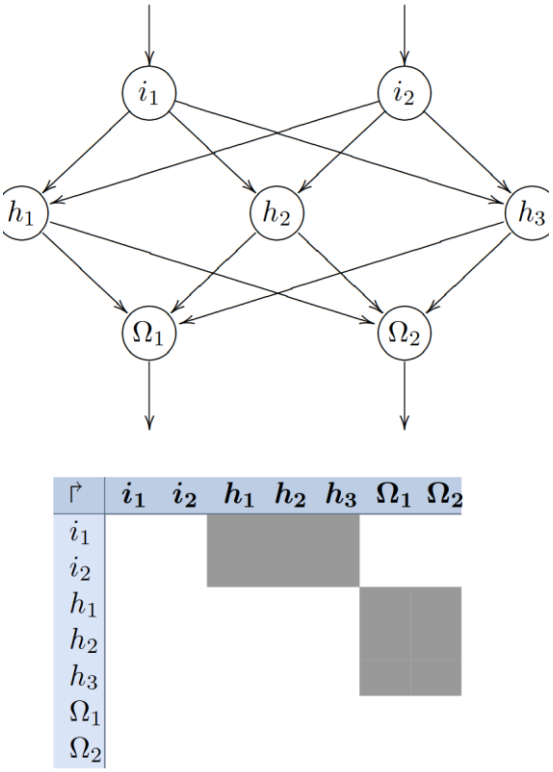
$$f_{act} = \frac{1}{1 + e^{-z}} \tag{3-1}$$



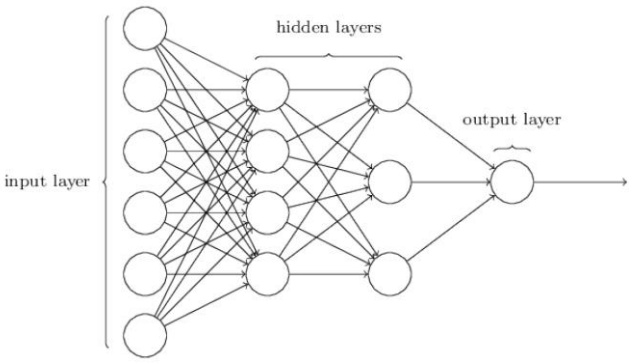
Slika 3.2: Prikaz aproksimacije step funkcije (lijevo) sigmoidnom (desno) [8]

Povezanost neurona se može prikazati Hintonovim dijagramom; kao matrica kojoj redak određuju polazni neuroni, a stupac neuroni u kojima veza završava. Neuronske mreže mogu imati razne topologije, no ovdje ćemo se osvrnuti samo na jednu: *feedforward* neuronsku mrežu. U takvoj mreži neuroni su poredani u slojeve koji mogu biti ulazni sloj, izlazni sloj ili skriveni sloj. Skriveni slojevi se tako nazivaju zbog toga što ih se izvana „ne vidi“; ne možemo na njih direktno utjecati, nisu ulaz ili izlaz mreže. Skrivenih slojeva može biti nenegativan cijeli broj. Kod

feedforward mreže, neuroni se mogu povezati samo sa neuronima iz sljedećih slojeva. Informacija putuje s ulaza, preko skrivenih neurona do izlaza. Ukoliko je svaki neuron povezan sa svakim iz idućeg sloja, tada je ta mreža potpuno povezana mreža. [8]



Slika 3.3: Feedforward mreža i Hintonov dijagram [8]



Slika 3.4: Mreža sa prikazanim ulaznim i izlaznim slojevima i dva skrivena sloja [7]

3.1.3 Učenje i greške

Učenje mreže je algoritam koji možemo opisati i prenijeti u programski jezik. Najčešće učenje simuliramo promjenom težina na odgovarajućim vezama između neurona. Mreže mogu učiti na više načina: sa i bez nadzora, te ohrabrivanjem (engl. *reinforcement*) kod kojeg se mreža nagradi ili kazni sukladno uspješnosti izvršenja željenog zadatka. Ako mreža uči bez nadzora, sama pokušava iz ulaza dobiti željeni rezultat za koji se trenira, a kod učenja sa nadzorom, mreža konstantno dobiva informaciju o svom napretku u obliku vektora greški kroz kojeg jasno vidi mijenjaju li se parametri mreže u dobrom smjeru. Kako bi uspješno proveli učenje sa nadzorom mreža mora jasno imati za svaki svoj ulaz željeni izlaz kako bi to dvoje mogli usporediti. Važno je prilikom treniranja da mreža ne zapamti gotovo doslovce što se očekuje kao izlaz za svaki ulaz, nego da ta trenirana mreža daje dobre rezultate na drugim sličnim ulazima iz iste klase. Često podatke za treniranje razdvajamo u više skupova. Skup koji mreži stalno prikazujemo i na kojemu

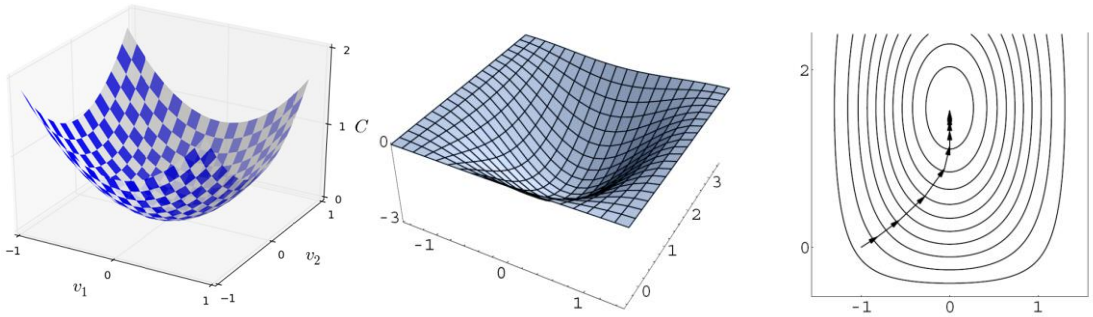
se trenira kroz etape, te skup koji predstavljamo mreži tek onda kada smo zadovoljni sa rezultatima mreže tijekom treniranja nakon čega možemo reći da je treniranje gotovo ako je mreža ponovno postigla dobre rezultate. Kako bi mogli pratiti trend grešaka mreže, a želimo da mreža s vremenom sve manje i manje griješi, moramo moći matematički odrediti greške. Greške se najčešće prikazuju pomoću srednje kvadratne pogreške (engl. *mean squared error*). [8]

$$Err_p = \frac{1}{n}\sum_{\Omega \in O}(t_\Omega - y_\Omega)^2 \tag{3-2}$$

Err_p predstavlja specifičnu grešku koja se računa nakon svakog ulaza, te predstavlja različitost između očekivane vrijednosti izlaza (t_Ω) i vrijednosti izlaza neurona (y_Ω) iz skupa izlaznih neurona (O). Ukupna greška (Err) predstavlja zbroj svih specifičnih grešaka i računa se nakon jedne etape; većeg broja ulaza. Graf ukupnih grešaka po etapama nam govori o uspješnosti mreže, u idealnom slučaju prati padajuću eksponencijalnu funkciju. Iz grešaka treniramo mrežu. Ukoliko je razlika između izlaza iz mreže i očekivanog izlaza velika, tada prilagodljive težine veza i pragove moramo puno promijeniti, u suprotnom ih malo mijenjamo. Kako bi se točno odredio taj pomak koristimo algoritme spuštanje gradijentom (engl. *gradient descent*) kojim tražimo minimume funkcije grešaka i propagacije unatrag (engl. *backpropagation*) koji svakom neuronu mijenja težine i pragove.

3.1.4 Spuštanje gradijentom

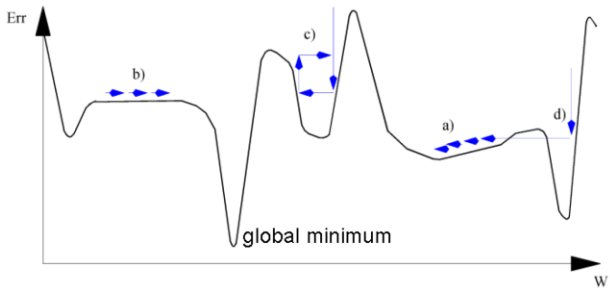
Spuštanje gradijentom traži globalni minimum složene funkcije. Ako prikažemo graf nekakve proizvoljne funkcije, možemo zamisliti kako spuštamo lopticu niz graf. Znamo kako će stati u „najdubljem“ dijelu funkcije; minimumu.



Slika 3.5: Intuitivan prikaz spuštanja gradijentom [7,8]

Za neuronske mreže bi funkcija bila vrlo složena (zbog svih težina i pragova), no za intuitivno objašnjenje to ovdje nije važno. Tu lopticu spuštamo od nasumično odabranog mjesta i pratimo kroz njen put kako se funkcija mijenja pomoću derivacija što nam govori gdje i kako se

loptica kreće. [7] Matematički, gradijent predstavlja vektor svih parcijalnih derivacija u određenoj točki neke n-dimenzionalne derivabilne funkcije i iz te točke pokazuje u maksimum te funkcije. Iz točke se putuje u suprotnom smjeru od gradijenta za udaljenost jednaku njegovom negativnom iznosu, kako bi došli do minimuma.



Slika 3.6: Problemi spuštanja gradijentom [8]

Baš kao i zamišljena loptica, spuštanje gradijentom može imati svoje probleme. Na slici 3.6 su prikazani neki od problema. Moguće je da se loptica zaustavi tamo gdje ne bi trebala (pronađen je minimum, no ne globalni; a)), da se zaustavi ili jako sporo kreće na „ravnim“ dijelovima funkcije (derivacija je 0 ili gotovo 0; b)), da stalno oscilira i nikada se ne smiri (c)) ili može izaći iz dobrog minimuma ili ga preskočiti (d)). Kako bi izbjegli takve probleme izračunati pomak (promjena težina i pragova) množimo sa faktorom η koji označava stopu učenja. Najčešće se nalazi u intervalu [0.01, 0.9]. [8] Postoje mnoge uporabe stope učenja: konstantna, promjenjiva ili pak za svaki sloj druga stopa učenja. Kod konstantne stope je vrlo teško odrediti ispravnu i ne postoji algoritam određivanja, nego čovjek svojim iskustvom određuje proučavajući različite arhitekture i uporabe mreža, dok je kod promjenjive važno da promjene dobro prate funkciju. Ako je stopa premalena, najčešće je učenje vrlo sporo no preciznije, dok ako je prevelika, učenje je puno brže no neprecizno. Početne vrijednosti težina bi trebale biti nasumične i nalaziti se u intervalu [-0.5, 0.5] ne uključujući 0 ili vrijednosti blizu 0 jer u tom slučaju se vrijednosti ne bi mijenjale. Također, ako su sve vrijednosti jednake, tada bi se jednako i mijenjale što onemogućava učenje. [8] Inicijalizacija mreže je problem za sebe. Gradijent možemo shvatiti i kao operaciju koja ima svoju oznaku zvanu nabla (∇).

3.1.5 Propagacija unatrag

Kao što je rečeno u potpoglavlju 3.1.2, za aktivacijsku funkciju koristimo derivabilne funkcije kako bi mogli odrediti derivacije, tim derivacijama učimo mrežu. Učimo je primjenom spuštanja gradijentom kako bi promijenili težine i pragove što se naziva propagacija unatrag.

Nadalje često deriviramo koristeći lančano pravilo (engl. *chain rule*) što je prikazano jednadžbom 3-3. Ako su z i y zavisne varijable, gdje z ovisi o y, a y o nezavisnom x, tada z također ovisi o x. Derivacija z u ovisnosti o x se može raspisati kao umnožak derivacije z u ovisnosti o y i derivacije y u ovisnosti o x. To pomaže jer nekada točno znamo što pojedini čimbenik predstavlja.

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \tag{3-3}$$

Neki odabrani neuron h ima svog prethodnika k i sljedbenika l, te oznake opisuju na koje se težine točno misli. Primjerice, $w_{k,h}$ označava težinu između prethodnog i trenutno odabranog neurona. Važno je matematički opisati promjenu svake težine nakon pojedine etape treniranja. Poznato je kako promjena težina ovisi o greškama; različitosti izlaza iz mreže i očekivanog izlaza što je prikazano jednadžbom 3-4. Err predstavlja funkciju greški, $w_{k,h}$ težinu između prethodnog i odabranog neurona, net_h predstavlja sumu sa težinama u trenutni neuron.

$$\frac{\partial Err(w_{k,h})}{\partial w_{k,h}} = \frac{\partial Err}{\partial net_h} \cdot \frac{\partial net_h}{\partial w_{k,h}} \tag{3-4}[8]$$

Primjenom lančanog pravila rastavljamo ovisnost određene težine o greškama na dva čimbenika. Prvi čimbenik se često označava sa $-\delta_h$. Drugi čimbenik je jednostavno izračunati (jednadžba 3-5); ovisnost određene težine $w_{k,h}$ o sumi sa težinama (ulaz u neuron) net_h jednaka je iznosu izlaza iz prethodnog neurona o_k. U sumi svih izlaza iz prethodnih neurona pomnoženima sa svim odgovarajućim težinama veza (svaki k iz skupa prethodnih neurona K) koje idu u promatrani neuron jedino izlaz pomnožen sa traženom težinom $w_{k,h}$ deriviran po $w_{k,h}$ opstaje; derivacija je različita od 0 i ta derivacija iznosi o_k.

$$\frac{\partial net_h}{\partial w_{k,h}} = \frac{\partial \sum_{k \in K} w_{k,h} o_k}{\partial w_{k,h}} = o_k \tag{3-5}[8]$$

Kako bi izrazili prvi čimbenik (δ_h), ponovno koristimo lančano pravilo (jednadžba 3-6). Uzimajući u obzir već navedene oznake h, k i l, da se zaključiti da o_h predstavlja izlaz trenutno promatranog neurona.

$$\delta_h = -\frac{\partial Err}{\partial net_h} = -\frac{\partial Err}{\partial o_h} \cdot \frac{\partial o_h}{\partial net_h} \tag{3-6}[8]$$

Drugi čimbenik jednadžbe 3-6 opisuje promjenu izlaza danog neurona u ovisnosti o ulaznoj sumi sa težinama. Izlaz iz danog neurona predstavlja primjenu aktivacijske funkcije na ulaz. Ovdje se radi o jednostavnoj derivaciji funkcije (3-7).

$$\frac{\partial o_h}{\partial net_h} = \frac{\partial f_{act}(net_h)}{\partial net_h} = f'_{act}(net_h) \tag{3-7}[8]$$

Slično dosadašnjim jednadžbama se izvodi prvi čimbenik jednadžbe 3-6, stoga se neće detaljno opisivati svaka nadolazeća jednadžba. Umjesto da pratimo prethodne neurone, pratimo iduće neurone. Struktura jednadžbi je slična. Važno je napomenuti da promjena funkcije greški u ovisnosti o izlazima promatranog neurona ovisi o sljedećem sloju (sloju nakon promatranog); o ulazima u iduće neurone (sume sa težinama). Posljednji put u jednadžbi 3-8 koristimo lančano pravilo.

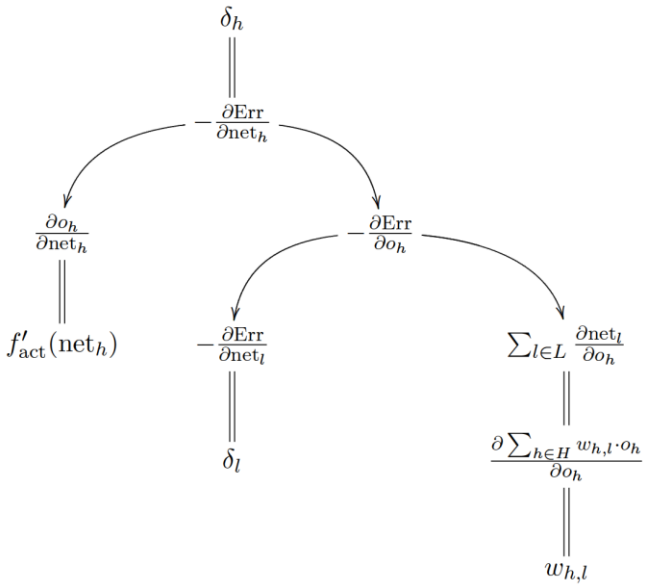
$$-\frac{\partial Err}{\partial o_h} = \sum_{l \epsilon L} \frac{\partial Err}{\partial net_l} \cdot \frac{\partial net_l}{\partial o_h} \tag{3-8}[8]$$

Prvi čimbenik u 3-8 nazivamo $-\delta_l$ po uzoru na δ_h. Drugi čimbenik možemo izračunati (3-9), slično kao u 3-5:

$$\frac{\partial net_l}{\partial o_h} = \frac{\partial \sum_{h \in H} w_{h,l} o_h}{\partial o_h} = w_{h,l} \tag{3-9}[8]$$

Konačno, prvi čimbenik u 3-6 je predstavljen jednadžbom 3-10 čime su svi čimbenici definirani.

$$-\frac{\partial Err}{\partial o_h} = \sum_{l \epsilon L} \delta_l w_{h,l} \tag{3-10}[8]$$



Slika 3.7: Grafički prikaz jednakosti i grananja danih jednadžbi [8]

Strelice prikazuju primjenu lančanog pravila, a dvostruke linije predstavljaju znak jednakosti (slika 3.7). [8] Na posljetku se dolazi do rekurzivne formule za propagaciju unatrag (3-11). Posljednji neuroni na izlaznom sloju prvi mijenjaju svoje težine uspoređujući svoj izlaz sa očekivanim izlazom, tada se ta promijenjena težina posljednjih neurona prosljeđuje unatrag kroz

mrežu utječući na prethodne neurone. Zbog toga se ovaj algoritam zove propagacija unatrag koji u suštini objašnjava način na koji male promjene više težina utječu na izlaz i kako ih možemo potaknuti da odrade posao koji želimo. Jasno se vidi i implementacija stope učenja kod promjene težina. Gornji slučaj δ_h se odnosi na izlazne neurone, dok se donji slučaj odnosi na ostale neurone.

$$\Delta w_{k,h} = \eta o_k \delta_h \tag{3-11}[8]$$

$$\delta_h = \begin{cases} f'_{act}(net_h) \cdot (t_\mathrm{h} - y_\mathrm{h}) \\ f'_{act}(net_h) \cdot \sum_{l \epsilon L} \delta_l w_{h,l} \end{cases}$$

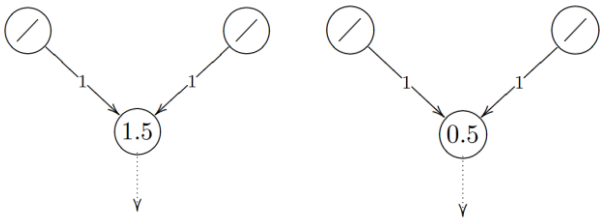
Važno je napomenuti da δ_h ovisi o odabranoj funkciji grešaka, u ovom slučaju je odabrana kvadratna udaljenost koja u određenim situacijama uzrokuje sporo početno učenje. Iz slike 3.2 možemo zaključiti da derivacija sigmoidne aktivacijske funkcije se bliži 0 kada je izlaz iz neurona blizu 1 što utječe na δ_h, pa onda i na $\Delta w_{k,h}$. Promjenom funkcije grešaka možemo ukloniti ovisnost o članu $f'_{act}(net_h)$ što rješava navedeni problem i omogućava da što smo dalje od traženog izlaza mreža brže uči. Primjer takve funkcije je funkcija križne entropije (engl. *cross-entropy function*) koja je dana jednadžbom 3-12, gdje p predstavlja uređeni par ulaza u mrežu i očekivanog izlaza t_p iz skupa takvih parova P, a y_p predstavlja izlaz iz mreže za dani ulaz p. n predstavlja broj ulaza.

$$Err = -\frac{1}{n}\sum\nolimits_{\mathrm{p}\epsilon\mathrm{P}} t_\mathrm{p} \ln(y_\mathrm{p}) + (1 - t_\mathrm{p})\ln(1 - y_\mathrm{p}) \tag{3-12}$$

Poseban slučaj propagacije unatrag bi bilo delta pravilo koje se odnosi na propagaciju grešaka kod SLP vrste mreža (potpoglavlje 3.1.6) kod kojih vrijedi: $\delta_h = (t_\mathrm{h} - y_\mathrm{h})$. [8]

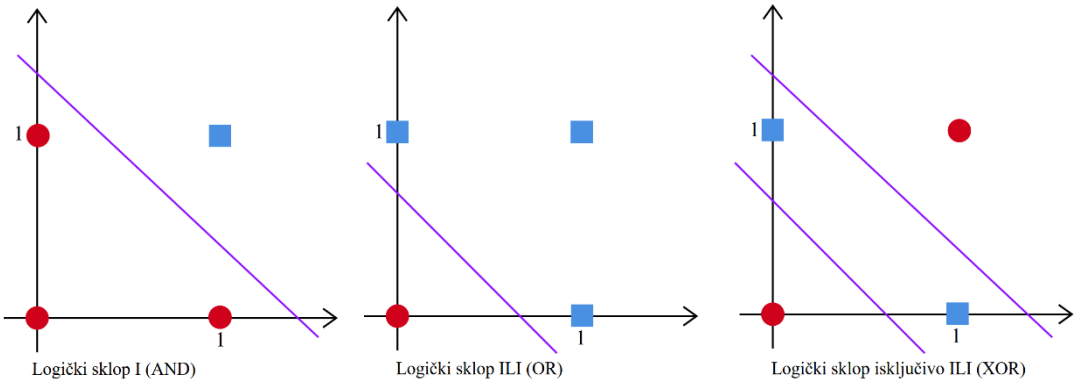
3.1.6 Perceptron

Perceptron označava algoritam treniranja mreža sa nadzorom. Perceptron se prikazuje neuronskom mrežom sa ulazima koji prosljeđuju ulazne podatke dalje u mrežu točno onakvima kakvi jesu, izlazim neuronima i skrivenim neuronima prateći *feedfoward* tip mreže sa potpuno povezanim slojevima. Mreža mora imati barem jedan sloj sa vezama koje se mogu trenirati. Postoji više vrsta perceptrona: jednoslojni perceptron (engl. *singlelayer perceptron*, SLP) i višeslojni perceptron (engl. *multiplayer perceptron*, MLP). Kod SLP vrste mreža sadrži samo ulazni i izlazni sloj, samim time sadrži i samo jedan sloj sa vezama čije se težine mogu mijenjati, dok kod MLP vrste mreža može sadržavati jedan ili više skrivenih slojeva.



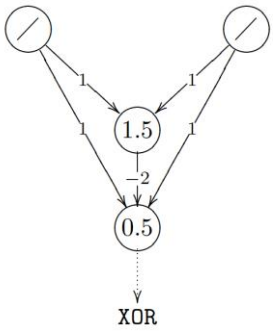
Slika 3.8: Prikaz logičke funkcije I (lijevo) i ILI (desno) pomoću SLP mreže [8]

SLP je jedna od najjednostavnijih tipova mreža i jako je korisna kod aproksimacije linearnih funkcija ili jednostavnijih logičkih funkcija (slika 3.8). Na slici su prikazane mreže sa dva ulazna neurona označena sa „/“ i jednim izlaznim neuronom sa pragom upisanim u njega. Prikazane su i težine na vezama.



Slika 3.9: Odvajanje očekivanih izlaza prikazanih logičkih sklopova

Ako prikažemo ulaze i očekivane izlaze u koordinatnoj ravnini (slika 3.9), za logičke sklopove I i ILI možemo linearno odvojiti očekivane 0 (krugovi) i 1 (kvadrati), dok za logički sklop isključivo ILI to ne možemo napraviti. Za prva dva primjera je potreban jedan pravac, dok u posljednjem primjeru su potrebna dva pravca kako bi izlaze odvojili. Pošto SLP dobro aproksimira linearne funkcije, trenirati mrežu da aproksimira I ili ILI nije teško i mreža konvergira određenom rješenju, dok kod aproksimacije isključivo ILI to nije moguće dobiti, mreža nikada ne konvergira. Nadalje, ako SLP mreža ima tri ulaza, problemi koje može riješiti moraju biti linearno odvojivi sa jednom ravninom. Ako označimo broj ulaza sa n, tada ulaze i izlaze možemo prikazati u n-dimezionalnom prostoru, a izlazi moraju biti odvojivi sa $n-1$ dimenzionalnom hiperravninom. Možemo smatrati da SLP predstavlja jedan pravac. [8] Jasno je kako SLP mreža ima svoja ograničenja. Kako bi aproksimirali isključivo ILI funkciju, potrebna nam je MLP mreža (slika 3.10).



Slika 3.10: Prikaz logičke funkcije isključivo ILI pomoću MLP mreže [8]

MLP mreža ima više od jednog sloja veza sa promjenjivim težinama, te takva mreža može aproksimirati bilo koju neprekidnu funkciju. Dodavanjem neurona, povećavamo preciznost aproksimacije ukoliko se radi o složenoj funkciji.

Bilo bi dobro da izlaz iz neuronske mreže možemo prikazati pomoću distribucije vjerojatnosti; izlazi poprimaju vrijednosti između 0 i 1, te je njihov zbroj uvijek 1. Trenirali bi mrežu tako da za nama povoljan ishod mreža sa velikom sigurnošću tvrdi da je točan, dok ostali izlazi imaju vjerojatnost da su točni jako blizu 0. To se postiže sa *softmax* slojem neurona koji imaju *softmax* aktivacijsku funkciju koja je dana jednadžbom 3-13. Takva aktivacijska funkcija uzima u obzir ostale izlaze, te povećanjem jednog izlaza drugi se smanjuju. z_i predstavlja ulaz u promatrani neuron, dok J predstavlja skup svih ostalih neurona.

$$f_{act} = \frac{e^{z_i}}{\sum_{j \in J} e^{z_j}} \tag{3-13}$$

3.2 Duboko učenje

3.2.1 Konvolucijske neuronske mreže

Za potrebe raspoznavanja objekata sa slika, uklanjanja šuma ili nekih sličnih problema obrade slike konvolucijske neuronske mreže pokazuju jako dobre rezultate. Bazirane su na MLP tipu mreža sa nekoliko izmjena. Omogućavaju vrlo duboke mreže na način da otklanjaju neke od problema dubokih MLP mreža[1] kao što je prevelika prilagodba podacima, te veliki broj podesivih

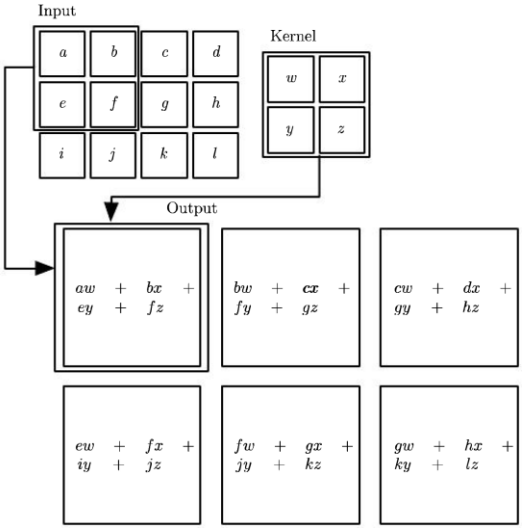
[1] mreža može umjesto rješavanja problema (generalizacije) doslovce zapamtiti povoljan ishod podataka za treniranje, čime smanjuje preciznost pri izvršavanju na testnim podacima što je najčešće uzrokovano odabirom pogrešne mreže za određeni problem (najčešće mreža sa previše parametara)

parametara čime se ubrzava njihovo treniranje. Svoje su ime dobile po linearnoj matematičkoj operaciji konvolucije koja zamjenjuje matrične operacije prilikom treniranja. Takva operacija se označava sa $*$. Za kontinuirane funkcije (signale) se računa pomoću jednadžbe 3-14. Funkcija f se često naziva ulaz, funkcija g jezgra (engl. *kernel*) operacije, a izlaz operacije karta značajki (engl. *feature map*). Prilikom konvolucije jedna od funkcija se obrće i pomiče za parametar t. Operacija je komutativna i prikazana je na slici 3.11 gdje *input* predstavlja ulaz, a *kernel* jezgru.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{3-14}$$

Pošto računala ne rade sa kontinuiranim, nego sa diskretnim vrijednostima, diskretna konvolucija nam je u ovom slučaju zanimljivija i prikazana je jednadžbom 3-15.

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau) \tag{3-15}$$



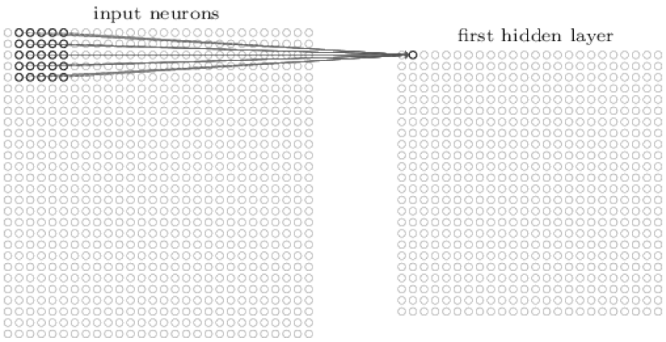
Slika 3.11: Konvolucija [9]

Pri obradi slike kao ulaz nemamo jednodimenzionalnu funkciju nego konvoluciju radimo na dvodimenzionalnoj slici sa dvodimenzionalnom jezgrom. Sliku označimo kao I, te jezgru kao K; konvolucija je prikazana jednadžbom 3-16.

$$(I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{3-16}[9]$$

Intuitivno, možemo zamisliti neurone poredane u retke i stupce kao piksele slike. Svaki neuron kao ulaz prima intenzitet određenog piksela. Kao i kod MLP vrste mreža, te ulazne neurone spajamo sa idućim skrivenim slojem, no ne spajamo svaki sa svakim nego određeni neuron iz idućeg sloja spajamo sa manjim brojem neurona iz ulaznog sloja (slika 3.12). *Input neurons* predstavljaju ulazne neurone, a *first hidden layer* prvi skriveni sloj. Lokalno recepcijsko polje

(engl. *local receptive field*) predstavlja taj fiksni broj neurona sa kojima je neuron iz prvog skrivenog sloja povezan i pomičemo ga po slici (ulaznim neuronima) za određeni korak (engl. *stride*) tako da svaki neuron iz prvog skrivenog sloja proučava dio slike omeđen tim poljem. Svi neuroni u prvom skrivenom sloju imaju jednak prag i jednake težine na vezama sa prethodnim slojem. To omogućava da svaki skriveni sloj prepoznaje određenu značajku koja se može nalaziti bilo gdje na slici. Preslikavanje između ulaznog neurona u idući skriveni sloj predstavlja kartu značajki, a dijeljene pragove i težine nazivamo jezgrom ili filterom. Jedan konvolucijski sloj je definiran sa više karata značajki. [7]

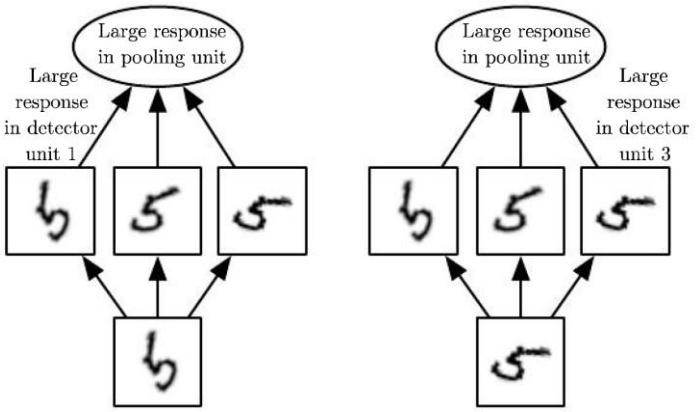


Slika 3.12: Intuitivan prikaz konvolucije [7]

Pošto neuroni nisu spojeni svaki sa svakim dolazi do velikog smanjenja potrebnih parametara koje definiraju mrežu što je omogućeno jezgrom manje veličine od ulaza. Smanjeno je trajanje izvođenja i, dodatno, potrebna količina memorije da se pohrane potrebni podaci zbog korištenja jednakih pragova i težina. Tražena značajka koju traži određeni sloj može se nalaziti bilo gdje; translacija ne utječe na uspješnost konvolucijske mreže, no ne može biti drugačije skalirana ili rotirana zbog prirode konvolucije. [9] Primjer toga bi bilo prepoznavanje lica na slici. Ako je mreža trenirana na sličnim slikama kao što je ulazna slika, no lice na ulaznoj slici je pomaknuto, mreža će vjerojatno dobro odraditi posao. Ukoliko je to lice naopako, mreža neće moći prepoznati da se uopće radi o licu (osim ako je mreža trenirana na naopakim licima).

Svaki sloj konvolucijske mreže se sastoji od različitih operacija na ulazu u pojedini sloj. Nakon opisane konvolucije, informacija putuje kroz aktivacijsku funkciju pojedinih neurona, te nakon toga se obrađuje izlaz iz te aktivacijske funkcije pomoću udruživanja vrijednosti (engl. *pooling*). To udruživanje omogućava čuvanje samo bitnih informacija za danju obradu u ostatku mreže što omogućava dodatno smanjivanje parametara potrebnih za definiciju pojedine mreže. U konvolucijskim mrežama često je važno raspoznati sadrži li slika tražene značajke, a ne gdje se one nalaze točno u jedan piksel. Sloj za udruživanje obrađuje izlaze iz pojedinih karata značajki i

opisuje nalazi li se tražena značajka u određenom dijelu slike. Primjer udruživanja bi bilo *max pooling*, udruživanje u kojem sloj prepoznaje maksimalne izlaze u određenom dijelu slike. Omogućavaju neovisnost o translaciji zbog toga što se translacijom određenih značajki ne mijenjaju znatno izlazi iz slojeva udruživanja, te također omogućavaju neovisnost o drugačijim transformacijama značajki tako što obrađuju izlaze iz različitih konvolucijskih slojeva koji su drugačije parametrizirani; u tom slučaju bi imali manje slojeva za udruživanje od konvolucijskih slojeva (slika 3.13). [7,9]



Slika 3.13: Sloj udruživanja omogućuje neovisnost o različitim transformacijama značajki [9]

Slika 3.13 intuitivno opisuje prethodnu rečenicu. Ako kao zadatak imamo raspoznavanje brojeva iz MNIST baze slika, jasno je da svaka „petica“ može biti drugačije orijentirana. Konvolucijski sloj nije neovisan o rotaciji, dakle drugačiji parametri su potrebni za svaku rotaciju broja. U zadnjem redu slike imamo dva primjera broja 5 koji su različito rotirani, a u redu iznad konvolucijski sloj koji prepoznaje određenu rotaciju broja. Lijevu rotaciju broja 5 je prepoznao prvi konvolucijski sloj, dok je desnu rotaciju prepoznao treći konvolucijski sloj. Pozitivna stvar kod slojeva udruživanja je ta što nije potrebno imati zaseban sloj za svaku rotaciju, nego dobivamo dobru reakciju na bilo koju od rotacija koja ukazuje da se broj 5 nalazi na danoj slici. Time se mreža pojednostavljuje; smanjuje se naredni broj neurona u mreži.

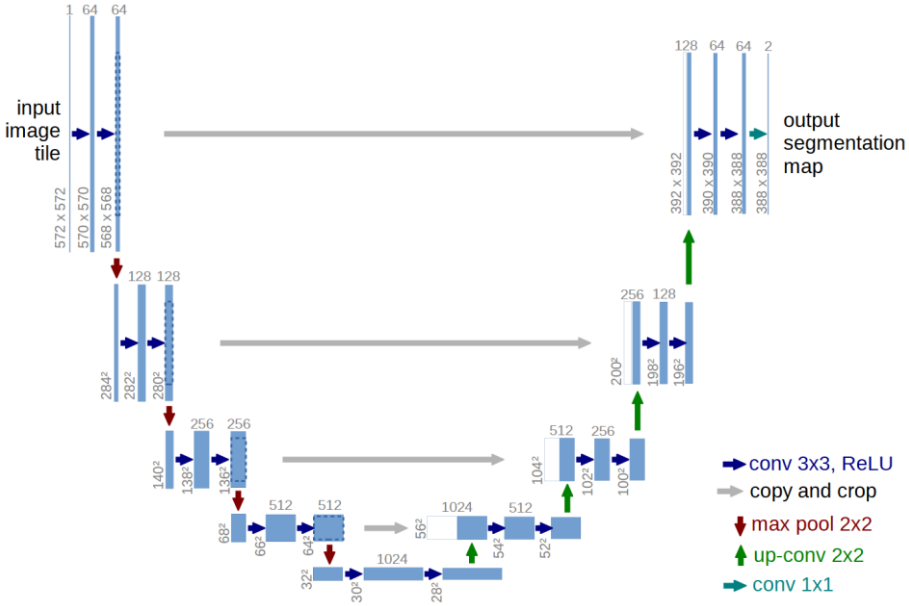
Osim već spomenute sigmoidne aktivacijske funkcije, postoje i mnoge druge. Za konvolucijske neuronske mreže se u novije vrijeme često koristi linearni ispravljač (engl. *rectified linear unit*, ReLU). Koji je definiran kao $\max(0, z)$. Zanimljiv je zbog otpornosti na zasićenost ukoliko se izlazi iz neurona približavaju 0 ili 1 (derivacija se bliži 0, spomenuto u potpoglavlju 3.1.5) što sprječava usporavanje učenja.

3.2.2 Autokoder

Autokoderi su konvolucijske mreže koje kao cilj imaju preslikati ulaz na izlaz gotovo identično i treniraju se poput bilo koje druge *feedforward* mreže. Sastoje se od dijela koji kodira podatke, te od dijela koji dekodira podatke. Autokoder koji nauči doslovce kopirati ulaz na izlaz je naučio simulirati funkciju identiteta što nije korisno. Svrha im je da nauče korisne značajke slika što se postiže ograničavanjem njihove arhitekture kako ne bi bili u mogućnosti naučiti navedenu jednostavnu funkciju. Ograničenja prisiljavaju mrežu da neke od informacija sa ulaza zanemari što je korisno u uklanjaju šuma. Spuštanjem gradijentom autokoderi minimiziraju funkciju greški poput općenitih neuronskih mreža, no sa drugačijim parametrima čime dolazimo do autokodera za uklanjanje šuma (engl. *denoising autoencoder*, DAE) koji minimizira razliku između izlaza (obrađena slika sa šumom) i predočene slike bez šuma. Dakle, DAE za zadatak imaju uklanjanje šuma umjesto slijepog kopiranja ulaza. [9]

3.2.3 U-Net arhitektura

Autori [10] predstavljaju brzu mrežu koja dobro iskorištava ulazne slike i omogućava dobro pronalaženje traženih značajki kao i njihovu lokalizaciju.



Slika 3.14: U-Net arhitektura [10]

Mreža se sastoji od dva dijela: sužavajućeg i proširujućeg dijela koji je promjena naspram ostalih rješenja. Sužavajući dio prati uobičajenu konvolucijsku mrežu sa konvolucijom i udruživanjem značajki koje se bazira na *max pooling* udruživanju sa korakom od 2. Nakon svakog

udruživanja značajki broj karata značajki se udvostručuje što omogućava propagaciju konteksta drugim slojevima. Aktivacijska funkcija je već spomenuti ReLU. Proširujući dio također prati uobičajenu konvolucijsku mrežu, no umjesto udruživanja koristi inverznu konvoluciju koja umjesto sažimanja značajki, proširuje sažete informacije od prethodnih slojeva i prepolavlja broj karata značajki. Pri svakoj inverznoj konvoluciji informacije od odgovarajućeg sužavajućeg sloja se pridodaju trenutnom sloju što uvodi informacije o položaju značajki te tvori arhitekturu koja podsjeća na slovo U (slika 3.14). Plave strelice prikazuju konvoluciju sa jezgrom veličine 3x3, sive strelice pridodavanje prethodnih odgovarajućih slojeva, crvene strelice sažimanje značajki, zelene strelice inverznu konvoluciju, te svijetlo plava strelica konvoluciju sa jezgrom veličine 1x1.

3.3 Python

Python je objektno orijentirani programski jezik opće namjene koji je često korišten zbog lakoće pisanja programskoga koda, zbog toga se vrlo često uči u školama kao uvod u programske jezike. Dostupan je na svim većim operacijskim sustavima (Linux, macOS, Windows i drugi), često se koristi i na Raspberry Pi-ju, koji je računalo namijenjeno učenju ili hobistima koji vole raditi zanimljive projekte. Postoji jako puno već gotovih biblioteka za programski jezik što također pridonosi njegovoj popularnosti. Programski jezik je interpretiran, a ne preveden što uvelike usporava njegovo izvođenje, no korisnik ne mora razmišljati o memoriji ili specifičnostima svakog sustava na kojemu se program izvodi. Osim što je njegova sintaksa relativno jednostavna, instalirati potrebno programsko okruženje je također jednostavno. Na već spomenutom Raspberry Pi-ju to okruženje često dolazi već instalirano, ovisno o odabranom operacijskom sustavu. Moguće je poigrati se sa sintaksom i na nekim internetskim stranicama.

3.4 Keras

Keras je aplikacijsko korisničko sučelje programskoga jezika Python koje omogućava korisnicima jednostavnu implementaciju algoritama dubokog učenja. Autori Keras-a smatraju kako su principi dubokoga učenja jednostavni, te kako bi njihova implementacija također trebala biti jednostavna. Naučiti koristiti ovo aplikacijsko korisničko sučelje je vrlo jednostavno, pogotovo za nekakve jednostavnije projekte, sučelje podržava i najsloženije ideje što naravno utječe na jednostavnost korištenja. Mogu se koristiti osnovni predefinirani moduli ili se mogu stvoriti novi po potrebi, Keras je vrlo prilagodljiv. Omogućava inženjerima i znanstvenicima da brže provedu svoje ideje u opipljive rezultate. Podržava spremanje već treniranih mreža za

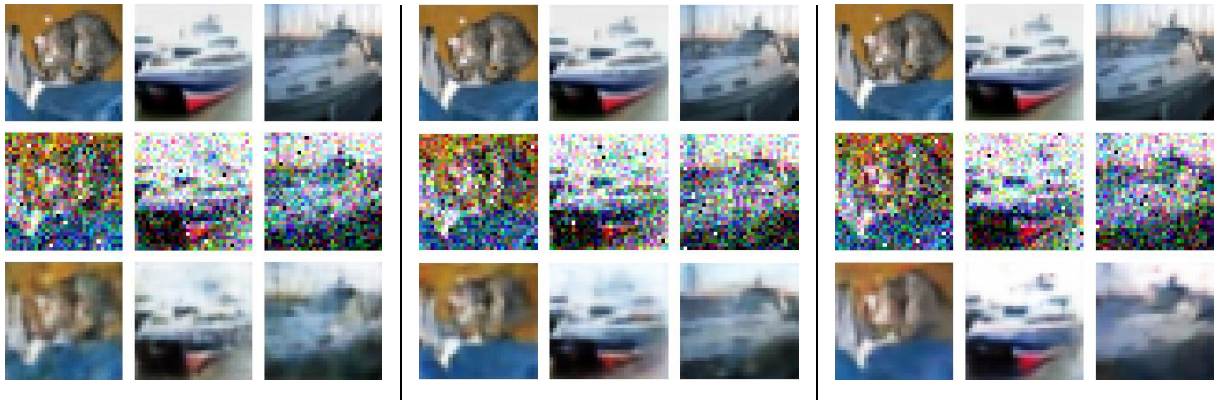
korištenje na drugim računalima ili platformama, te treniranje na procesorima, grafičkim karticama i drugom sklopovlju. [11]

4. UKLANJANJE ŠUMA IZ SLIKA

4.1 Odabrani model

Arhitektura modela je bazirana na U-Net arhitekturi objašnjenoj u potpoglavlju 3.2.3 gdje je na slici 3.14 prikazana arhitektura originalnoga rada. Mogu se uočiti 5 razina spuštajućeg dijela mreže sa 4 udruživanja značajki počevši od 64 karte značajki. Udvostručujući na svakoj razini, taj lijevi dio slova U završava sa 1024 karte značajki, te takva mreža ukupno ima oko 31 milijun promjenjivih parametara što dovodi u pitanje ograničenje sklopovlja i vrijeme izvođenja. Ukoliko se U-Net arhitektura prilagodi tako da umjesto 64 početne karte značajki sadrži 32, broj parametara mreže se drastično mijenja i iznosi oko 7 milijuna i takva se mreža puno brže trenira uz korištenje manje računalnih resursa. Takva prilagođena arhitektura je iskorištena za realizaciju autokodera i uklanjanje šuma sa slijepim popunjavanjem.

Ako se dodatno smanji broj karata značajki na primjerice 16, jasno je da bi broj prilagodljivih parametara mreže bio ponovno umanjen i to na oko 2 milijuna parametara. Međutim, takva mreža puno sporije smanjuje vrijednosti funkcije grešaka; potrebno je više epoha[2] za treniranje i lošije obavlja svoj posao za jednake vrijednosti parametara aditivnog bijelog Gaussovog šuma i šuma soli i papra. Testirano je više različitih početnih karata značajki. Empirijski podaci ukazuju da smanjenjem veličine mreže vrijednosti funkcije grešaka počinju na većem broju, te pošto mreža može upamtiti manji broj značajki kvaliteta rekonstruirane slike je manja, dodatno, ukoliko se mreža trenira sa više šuma, vrijednosti funkcije grešaka također kreću od veće vrijednosti.



Slika 4.1: Prikaz uklanjanja šuma sa različitim početnim brojem karata značajki: 16, 32, 64

[2] Epoha pri treniranju označava jedan prolaz kroz mrežu svih predanih podataka (slika) za treniranje

Predstavljeno je uklanjanje šuma sa različitim početnim brojem karata značajki na slici 4.1 gdje su u prvom redu prikazane slike bez šuma, u idućem redu slike sa šumom, a u posljednjem redu slike nakon otklanjanja šuma. Na prvom dijelu na slici 4.1 uočeno je da je uklanjanje šuma dosta lošije nego na preostala dva dijela slike na kojima je šum otklonjen podjednako. Sve 3 mreže su trenirane na jednakim iznosima šuma i sa jednakim brojem etapa. Navedena slika potvrđuje izbor od 32 karte značajki kao parametar mreže zbog dovoljno dobrog uklanjanja šuma i manjeg zahtjeva računalnih resursa.

Zajedno baze slika CIFAR-10 i MNIST sadrže 110,000 slika za trening, računati grešku nakon svake slike koja se propagira kroz mrežu bi dugo trajalo, stoga se greška računa nakon pojedine serije od više slika. Povećanjem broja slika u seriji ubrzava se vrijeme izvođenja, no zahtjev nad računalnim sklopovljem je također veći; treniranje koristi više memorije. Odabran je broj od 150 slika u seriji pri treniranju mreže, te takva aproksimacija stvarnih grešaka ne utječe uvelike na ishod treniranja.

Mreža se može trenirati na samo jednoj od dvije navedene baze slika, a postignuti rezultati su dakako zanimljivi. Ako pri treniranju manje slika prolazi kroz mrežu, tada se brže trenira. Uz jednake količine šuma i broj epoha treniranja kreirana su dva modela, pojedini za svaku od vrsta šuma.

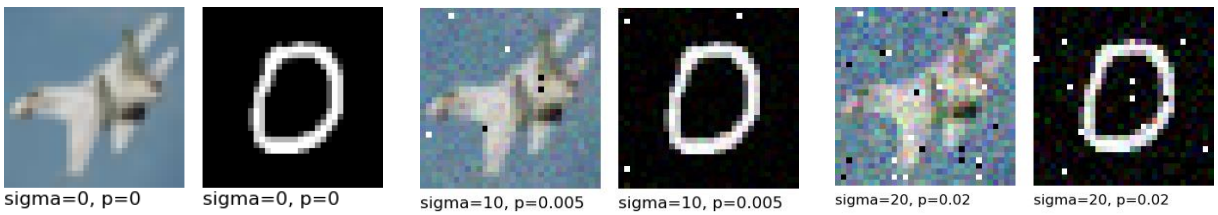
Slika 4.2 : Uklanjanje šuma modelom treniranim na CIFAR-10 bazi slika

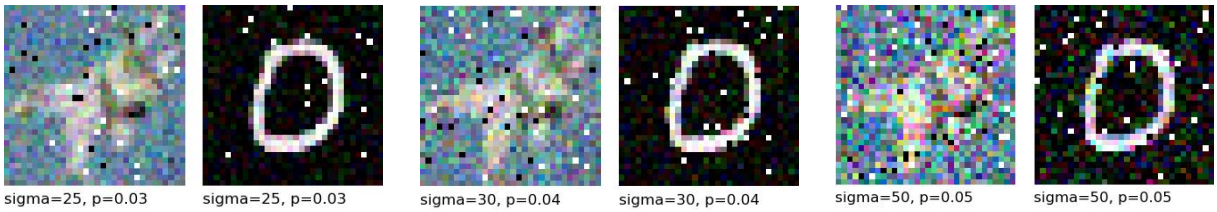
Model treniran na CIFAR-10 bazi slika dobro uklanja zadani šum sa slika iz obje baze, s tim da slike sa uklonjenim šumom iz MNIST baze nisu više potpuno crno-bijele (slika 4.2). Uklanjanje šuma bazirano na ovakvom modelu je zadovoljavajuće. Ukoliko se trenira isključivo na slikama iz MNIST baze, tada je situacija potpuno drugačija (slika 4.3). Slike sa uklonjenim šumom iz MNIST baze su potpuno crno-bijele, kao i originalna slika, no slike iz CIFAR-10 baze su također bezbojne za razliku od originalnih slika. Ovako trenirana mreža ne daje zadovoljavajuće rezultate pošto se mreža nije susrela sa slikama u boji.

Model za ovaj rad se trenira na obje baze slika zbog toga što se tako postižu najbolji rezultati za slike iz obje baze.

Slika 4.3: Uklanjanje šuma modelom treniranim na MNIST bazi slika

Preostalo je odabrati početnu količinu obje vrste šuma. Parametri koji definiraju količinu šuma su standardna devijacija (Gaussove distribucije) σ za aditivni bijeli Gaussov šum, te vjerojatnost (piksela da ima minimalnu ili maksimalnu vrijednost) p za šum soli i papra.





Slika 4.4: Razni intenziteti šuma σ = sigma, p = p

Na slici 4.4 su prikazani različiti intenziteti šuma na slikama iz baza slika CIFAR-10 i MNIST. Slike iz CIFAR-10 baze su puno manje otporne na šum zbog svoje veće kompleksnosti naspram slika iz druge baze koje imaju manje rubova za prepoznati, te su crno-bijele. Uočeno je kako se slika aviona već u trećem porastu parametara šuma teško raspoznaje, dok slika broja 0 se može raspoznati na svim intenzitetima šuma. Za treniranje mreže su odabrani $\sigma = 30$, $p = 0.04$.

4.2 Programsko rješenje

4.2.1 Rješenje autokodera

Glavna zadaća ovoga rada je izrada modela za uklanjanje šuma iz slika. Potrebno je u programskom jeziku Python koristeći aplikacijsko korisničko sučelje Keras dizajnirati zadovoljavajući autokoder. Potrebni moduli za autokoder su prikazani na slici 4.5.

```
import keras.layers as layers
import keras.models as models
import keras.callbacks as callbacks
```

Slika 4.5: Potrebni moduli za kreiranje autokodera

Iz Keras-a su korišteni navedeni moduli nakon ključne riječi *import* sa svojim skraćenim nazivima koji su navedeni nakon riječi *as*.

Potrebno je definirati funkcije koje opisuju pojedine operacije navedene u originalnom radu U-Net arhitekture sa slike 3.14. Prva od takvih funkcija opisuje dvostruku konvoluciju koja se primjenjuje na svim razinama arhitekture (plave strelice), iduća funkcija opisuje sažimanje značajki (crvene strelice) koje je dominantno korišteno na lijevoj strani slova U, te posljednja pomoćna funkcija inverzne konvolucije koja je dominantno korištena na desnoj strani arhitekture (zelene strelice).

```
def convolution(k, n_filters, kernel_size=(3,3), strides=1):
    def apply_convolution(k):
        h = layers.Conv2D(filters=n_filters,
            kernel_size=kernel_size, strides=strides, padding='same')(k)
        h = layers.BatchNormalization()(h)
        h = layers.ReLU()(h)
        return h

    h = apply_convolution(k)
    h = apply_convolution(h)
    return h
```

Slika 4.6: Pomoćna funkcija dvostruke konvolucije

Dvostruka konvolucija je opisana slikom 4.6; funkcija *convolution* koja prima 4 argumenta. Argument *k* označava prethodni sloj (držimo se oznaka iz 3. gdje je k označavao prethodni neuron, h trenutni, a l idući neuron), *n_filters* broj filtera koji otkrivaju određene značajke slike tvoreći tako karte značajki za svaki filter, *kernel_size* veličinu prozora (jezgra) koji pomičemo po slici za korak od *strides*. Ukoliko je korak pri konvoluciji veći od 1, tada konvolucija smanjuje svoj ulaz poput sažimanja značajki što se ovdje neće koristiti jer je korišten poseban sloj za sažimanje značajki niti se značajke sažimaju nakon svake konvolucije. Unutar *convolution* definirana je pomoćna funkcija *apply_convolution* koja prima prethodni sloj *k* radi lakše dvostruke primijene konvolucije bez ponavljanja u programskom kodu. *apply_convolution* na prethodni sloj veže sloj *Conv2D* koji predstavlja konvoluciju u mreži, te na navedeni veže *BatchNormalization* sloj i na kraju sloj *ReLU* koji predstavlja aktivacijsku funkciju linearnog ispravljača. Takav poredak je proizašao iz rada [12] u kojemu je provedeno istraživanje o normalizaciji serije radi ubrzanja izvedbe gdje su autori stvorili svoj model koristeći normalizaciju serije prije aktivacijske funkcije. Normalizacija pospješuje konvergenciju mreže optimalnim parametrima tako što sprječava promjenu distribucije na ulazima u slojeve pri pojedinim iteracijama tijekom treniranja mreže. Normalizaciju serije opisuje sloj *BatchNormalization*. *Conv2D* je definiran sa još jednim parametrom, *padding*, koji simbolizira dodavanje pomoćnih 0 na ulazu u sloj ukoliko veličina slike nije djeljiva sa veličinom primijenjene jezgre. Postoje dvije moguće *padding* vrijednosti: *'valid'* i *'same'*. Kod prve se na sliku ne dodaju 0 u slučaju nedjeljivosti, nego se jezgra ne primjenjuje na područja u kojima ne preostaje dovoljno slike što za posljedicu ima smanjenje prve dvije dimenzije izlaznog tenzora, dok kod druge vrste se jezgra primjenjuje na čitavu sliku jer se dodaju 0 u slučaju nedjeljivosti kako bi se proširila ulazna slika što ne smanjuje veličinu izlaznog tenzora naspram

veličine ulaznoga. Ovdje je odabrana *padding* vrijednost *'same'*. Spomenuti tenzor ne predstavlja ništa drugo već višedimenzionalnu matricu. Slike su predstavljene sa 3-dimenzionalnim tenzorom gdje prve dvije dimenzije opisuju visinu i širinu slike, dok posljednja dimenzija predstavlja pojedini kanal slike u boji (kanali R, G i B za crvenu zelenu i plavu boju).

```
def pooling(k, pool_size=(2,2), strides=2):
    h = layers.MaxPooling2D(pool_size=pool_size, strides=strides)(k)
    return h
```

Slika 4.7: Pomoćna funkcija sažimanja značajki

Sažimanje značajki (slika 4.7) je oblikovano točno po originalnom radu U-Net. Koristi se *max pooling* sažimanje značajki zastupljeno sa *MaxPooling2D* slojem gdje *pool_size* predstavlja veličinu prozora kojeg pomičemo po slici. Ostali parametri se jednako zovu kao i u prethodnoj funkciji, te označavaju jednake stvari.

```
def deconvolution(k, n_filters, kernel_size=(2,2), strides=2):
    h = layers.Conv2DTranspose(filters=n_filters, kernel_size=kernel_size,
      strides=strides, padding='same')(k)
    h = layers.BatchNormalization()(h)
    h = layers.ReLU()(h)
    return h
```

Slika 4.8: Pomoćna funkcija inverzne konvolucije

Inverzna konvolucija je predstavljena funkcijom *deconvolution* (slika 4.8). Slična je konvoluciji, no umjesto *Conv2D* korišten je *Conv2DTranspose* koji opisuje operaciju inverzne konvolucije. Veličina korištene jezgre je 2x2 po uzoru na originalni rad, korakom od 2 na određeni način se poništava sažimanje značajki i mreža nastavlja desnom stranom slova U.

```
def autoencoder(first_n_filters, n_layers, shape=(32,32,3)):
    input_layer = layers.Input(shape=shape, name='input')
    n_filters = first_n_filters
    h = input_layer
    down = []

    for n in range(n_layers):
        h = convolution(h, n_filters)
        down.append(h)
        h = pooling(h)
        n_filters = n_filters * 2

    h = convolution(h, n_filters)
    down.reverse()

    for d in down:
        n_filters = n_filters // 2
        h = deconvolution(h, n_filters)
        h = layers.concatenate([h,d])
        h = convolution(h, n_filters)

    output_layer = layers.Conv2D(filters=3, kernel_size=(3, 3),
      activation='sigmoid', padding='same')(h)

    return models.Model(inputs=input_layer, outputs=output_layer,
      name="denoising_autoencoder")
```

Slika 4.9: Autokoder

Naposljetku je opisan autokoder koji objedinjuje prethodne pomoćne funkcije (slika 4.9). Funkcija slaže sloj na sloj tvoreći zadanu arhitekturu, te prima 3 parametra. *first_n_filters* predstavlja početni broj traženih karata značajki pri prvoj konvoluciji, *n_layers* broj razina na lijevoj strani slova U arhitekture, *shape* simbolizira oblik podataka koje predajemo mreži. To je 3-dimenzionalni tenzor veličine (32,32,3) što prikazuje sliku veličine 32x32 piksela sa 3 sloja, svaki za pojedinu boju (R, G, B slojevi). Prvenstveno je definiran ulazni sloj koji očekuje navedeni tenzor. Za svaku pojedinu razinu do razine *n_layers* primjenjena je dvostruka konvolucija (plava strelica na slici 3.14), sažimanje značajki, te je udvostručen broj filtera (*n_filters*) koji je predan odgovarajućim pomoćnim funkcijama. Nakon toga je primijenjena konvolucija koja predstavlja dno slova U nakon čega se prelazi na desnu stranu. Za svaku prethodnu dvostruku konvoluciju, osim zadnje za dno, dijeljen je broj filtera sa 2, primjenjena je inverzna konvolucija i pridodan je odgovarajući sloj sa lijeve strane slova U (siva strelica na slici 3.14), nakon čega je izvršena

dvostruka konvolucija. Mreža je gotovo dovršena, no izlaz iz mreže nije tenzor (32,32,3) što se postiže posljednjom primjenom konvolucije (svijetlo plava strelica na slici 3.14) dobivajući izlazni sloj. Testirano je više verzija izlaznog sloja, no najbolji rezultati se postižu sa prikazanim parametrima. Ukoliko ulaze u mrežu normaliziramo (ulazi poprimaju vrijednosti iz [0,1]) tada sigmoidna aktivacijska funkcija daje dobre rezultate jer također kao svoje izlaze može imati samo vrijednosti između 0 i 1.

4.2.2 Rješenje dodatnih funkcija

Potrebni moduli za dodatne funkcije prikazani su na slici 4.10.

```
import numpy as np
import cv2
```

Slika 4.10: Potrebni moduli za dodatne funkcije

Mreža kao ulaz očekuje tenzor veličine (32,32,3) što odgovara slikama iz CIFAR-10 baza slika. Kako bi mreža mogla uklanjati šum iz MNIST baze slika, te slike se moraju prilagoditi. Cijela baza CIFAR-10 slika ima oblik (n,32,32,3) gdje n predstavlja broj slika u bazi. Baza je 4-dimenzionalni tenzor, dok MNIST baza ima oblik (n,28,28) koja je 3-dimenzionalni tenzor zbog toga što su slike crno-bijele, nemaju slojeve za određene boje.

```
def reshape_MNIST(mnist):
    l,_,_ = mnist.shape
    bordered = np.ndarray((l,32,32), 'float32')
    for i in range(l):
        bordered[i] = cv2.copyMakeBorder(src=mnist[i], top=2, bottom=2, left=2,
            right=2, borderType=cv2.BORDER_CONSTANT, value=0)
    reshaped = bordered.reshape((bordered.shape + (1,)))
    reshaped = reshaped.repeat(3,-1)
    return reshaped
```

Slika 4.11: Oblikovanje MNIST slika

Funkcija *reshape_MNIST* (slika 4.11) svakoj slici iz baze slika MNIST dodaje crni obrub veličine 2 piksela sa svake strane čime sa (28,28) se dolazi do slike (32,32) što i dalje nije jednako kao kod druge baze slika. Svakoj slici se dodaje jedna dimenzija koja omogućava da slike poprime

oblik (32,32,1). Ponavljajući prvi sloj ukupno 3 puta se dolazi do oblika (32,32,3) čime oblikovanje završava.

```
def add_noise(dataset, noise_level_gauss = 50, noise_level_sp = 0.05,
              sp_distribution = 0.5):
    #gaussian noise
    noisy = dataset + np.random.normal(0, noise_level_gauss, dataset.shape) / 255

    #salt and pepper noise
    width, height, _ = dataset[0].shape
    size = width * height
    n_s = int(np.ceil(noise_level_sp * size * sp_distribution))
    n_p = int(np.ceil(noise_level_sp * size * (1-sp_distribution)))

    for picture in noisy:
        coordinates_s = [np.random.randint(0, i-1, n_s) for i in (width, height)]
        coordinates_p = [np.random.randint(0, i-1, n_p) for i in (width, height)]

        picture[tuple(coordinates_s)] = [1., 1., 1.]
        picture[tuple(coordinates_p)] = [0., 0., 0.]

    np.clip(noisy, 0., 1., noisy)
    return noisy
```

Slika 4.12: Dodavanje šuma slikama

Šum slikama je dodan funkcijom *add_noise* (slika 4.12) koja kao argumente prima: bazu slika kojoj se dodaje šum, σ za Gaussov šum, p za šum soli i papra, odnos između soli i papra (slika ima više svijetlih impulsa nego tamnih ili obratno). Svakoj slici u predanoj bazi slika se dodaje Gaussov šum dobiven pomoću *numpy.random* modula i funkcije *normal*. Nakon Gaussovog šuma se dodaje šum soli i papra računat tako da se pri nasumičnim koordinatama slike u svakom sloju slike postavi minimalna ili maksimalna vrijednost piksela. Naposljetku, vrijednosti piksela se ograniče između 0 i 1 ukoliko neke vrijednosti se više ne nalaze u tom rasponu nakon dodavanja šuma.

4.2.3 Rješenje treniranja mreže

Potrebno je prethodno navedene funkcije sastaviti u glavni program u kojemu je mreža trenirana i pripremiti podatke za obradu. Prikazati će se samo važniji dijelovi programa zbog svoje zanimljivosti.

```
import keras.datasets as datasets
import cv2
import keras.callbacks as callbacks
from autoencoder import autoencoder
from functions import *

(train_clean_mnist, _), (test_clean_mnist, _) = datasets.mnist.load_data()
(train_clean_cifar, _), (test_clean_cifar, _) = datasets.cifar10.load_data()
```

Slika 4.13: Učitavanje slika iz obje baze i potrebni moduli

Učitavanje slika se vrši vrlo jednostavno pozivom funkcije *load_data* na odabranim bazama slika (slika 4.13).

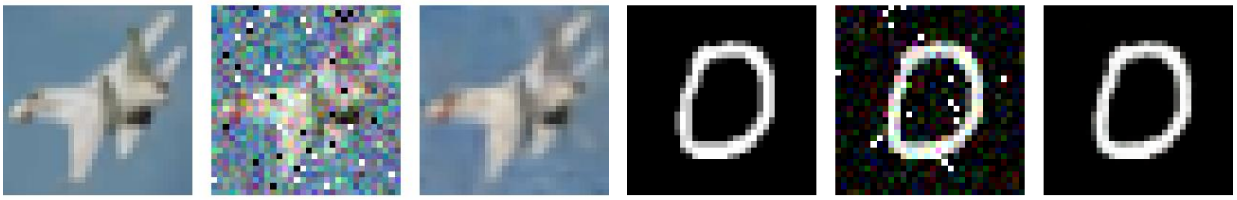
```
model = autoencoder(32, 4)
model.compile(loss='mse', optimizer='adam')
early_stopping = callbacks.EarlyStopping(monitor='val_loss', patience=4)
checkpoint = callbacks.ModelCheckpoint(filepath='model.hdf5',
      monitor='val_loss', verbose=1, save_best_only=True, mode='auto')
history = model.fit(train_noisy, train_clean, validation_data=(test_noisy,
      test_clean), epochs=20, batch_size=150, shuffle=True,
      callbacks=[early_stopping, checkpoint])
```

Slika 4.14: Definiranje i treniranje modela

Kako bi se definirao model, poziva se funkcija autoencoder iz potpoglavlja 4.2.1 sa parametrima 32 i 4 koji su objašnjeni u navedenom potpoglavlju. Prilikom podešavanja modela definirana je funkcija grešaka parametrom *loss* i način optimiziranja parametrom *optimizer*. Kao funkciju grešaka odabirana je srednja kvadratna pogreška (jednadžba 3-2). Optimizaciju se vrši pomoću adam optimizatora koji implementira objašnjene spuštanje gradijentom i propagaciju unatrag te daje jako dobre rezultate pri treniranju dubokih neuronskih mreža. Jedan je od češće korištenih optimizatora. Ukoliko se vrijednosti funkcije greški uzastopno ne smanjuju na testnom uzorku prilikom treniranja, tada se treniranje prekida jer to znači da mreža ima problem sa prilagodbom podacima (engl. *overfitting*). Najbolji model je spremljen u zasebnu datoteku radi naknadnog korištenja; nije potrebno više puta trenirati mrežu. Pozivom naredbe *fit* nad modelom počinje treniranje mreže nad predanim podacima, određeni broj epoha i veličine serije. Podaci su nasumično posloženi prije treniranja. Navedeno se odnosi na sliku 4.14.

4.3 Dobiveni rezultati

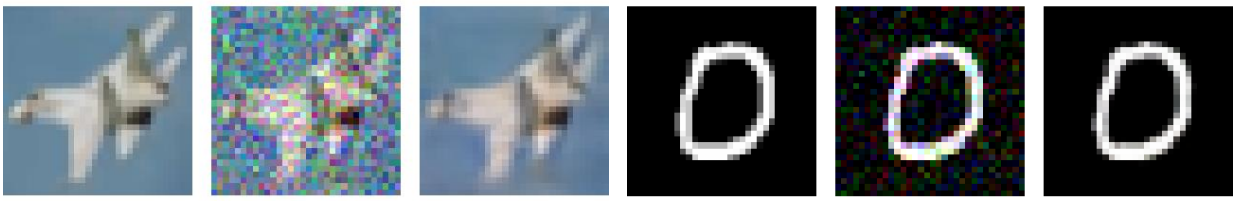
Mreža je trenirana 20 epoha s veličinom serije od 150. Prvo će se pokazati otklanjanje šuma na slikama sa $\sigma = 30$ i $p = 0.04$ baš kao što je odabrano za treniranje mreže.



Slika 4.15: Uklanjanje šuma sa $\sigma = 30$ i $p = 0.04$

Lijevo na slici 4.15 je prikazano otklanjanje šuma iz CIFAR-10 baze slika (prve tri sličice), a desno uklanjanje šuma iz MNIST baze. Sličice predstavljaju redom: originalnu sliku, sliku sa šumom, te sliku sa otklonjenim šumom. Vidi se kako originalna slika i slika sa otklonjenim šumom nisu potpuno jednake. To odstupanje se prikazuje PSNR-om koji iznosi oko 16 dB za slike sa šumom naspram originalnih slika. PSNR za sliku sa otklonjenim šumom iz CIFAR-10 baze iznosi 29.12 dB, a iz MNIST baze 32.22 dB. Poboljšanje se nakon uklanjanja šuma jasno vidi i predstavljeno je i numerički. Zbog svoje jednostavnosti, slike iz MNIST baze su otpornije na šum i sa njih se lakše otklanja što je vidljivo iz navedenih vrijednosti PSNR.

Uklanjanje samo jedne od dvije navedene vrste šuma je također uspješno. Na slici 4.16 je prikazano otklanjanje samo aditivnog bijelog Gaussovog šuma, a na slici 4.17 otklanjanje samo šuma soli i papra.



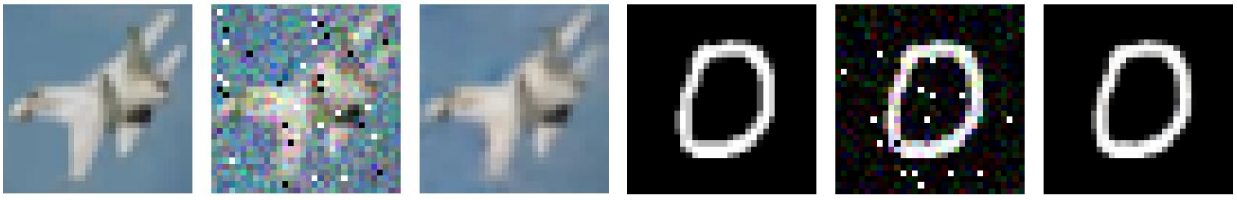
Slika 4.16: Uklanjanje šuma sa $\sigma = 30$ i $p = 0.0$

PSNR između originalne i slike sa isključivo Gaussovim šumom iznosi oko 18 dB za slike iz CIFAR-10 baze, a oko 22 dB za slike iz MNIST baze. PSNR za sliku sa otklonjenim Gaussovim šumom je poboljšan naspram PSNR za sliku sa otklonjenim oba šuma.

Slika 4.17: Uklanjanje šuma sa $\sigma = 0$ i $p = 0.04$

Ukoliko je korišten samo šum soli i papra, tada se PSNR od oko 20 dB i 17 dB povećava na 31.83 dB i 36.11 dB za slike iz CIFAR-10 i MNIST baze što je još veći porast naspram prijašnjih primjera. Mreža dobro otklanja oba šuma zajedno i pojedinačno bez obzira što slikama sa primjenim pojedinačnim šumom nije bila izložena pri treniranju.

Postavlja se pitanje kakve rezultate mreža postiže sa drugačijim parametrima šuma kao primjeri sa slike 4.4. Odabran je šum sa sličica lijevo i desno (manji i veći) od šuma odabranog za treniranje (sličice sa navedene slike 4.4). To su $\sigma = 25$, $p = 0.03$ i $\sigma = 50$, $p = 0.05$.



Slika 4.18: Uklanjanje šuma sa $\sigma = 25$ i $p = 0.03$

Ukoliko je šum smanjen naspram šuma pri treniranju, PSNR se povećava (slika 4.18), u suprotnom PSNR se smanjuje (slika 4.19). Razlika se vidi i u izgledu samih slika. Duboke mreže ne uče doslovce gdje se šum nalazi na slici (to bi bilo i teško jer je šum uvijek nasumičan), nego uče koji dio slike smatramo originalnim dijelom slike, a koji dodani šum i kako ga ukloniti. Mreža pri treniranju je vidjela samo određene intenzitete šuma, dok se sa drugim nije susretala, a daje dobre rezultate i pri promijenjenim parametrima. Ukoliko dodanog šuma ima puno više nego na kojemu je trenirana, tada mreža djelomično uklanja šum, no i dalje je relativno uspješna. Uočeno

je kako na CIFAR-10 bazi slika mreža lošije ukloni šum, a na MNIST bazi ga gotovo u potpunosti ukloni zbog već navedene jednostavnosti tih slika. U svakom slučaju, PSNR se poboljšava naspram slike sa šumom.

Slika 4.19: Uklanjanje šuma sa $\sigma = 50$ i $p = 0.05$

Na tablici 4.1 su zajedno prikazani dobiveni rezultati za obje baze slika. Zelenom bojom je naznačen eksperiment na slikama koje imaju σ i p jednak onom korištenom prilikom treniranja. Rezultati ostalih eksperimenata se daju usporediti s tim polaznim eksperimentom. Plavo je označen eksperiment sa smanjenom količinom šuma, a crveno eksperiment sa povećanom količinom šuma. Dosadašnji zaključci se jasno vide na jednom mjestu. Neobojano su prikazani eksperimenti bez pojedine vrste šuma. Prvi PSNR označava razliku između originalne slike i slike sa otklonjenim šumom, drugi predstavlja razliku između originalne i slike sa šumom. Uspoređujući ta dva PSNR, vidi se kvantitativno poboljšanje kvalitete slike u svakom od eksperimenata.

Tablica 4.1: Dobiveni rezultati za određene σ i p

Baza slika	σ	p	**PSNR$_1$ (dB)**	**PSNR$_2$ (dB)**
CIFAR-10	30	0.04	29.12	16.06
	30	0.00	29.43	18.81
	0	0.04	31.83	19.67
	25	0.03	30.70	17.54
	50	0.05	21.95	13.25
MNIST	30	0.04	32.22	15.93
	30	0.00	32.52	21.62
	0	0.04	36.11	17.31
	25	0.03	33.58	16.42
	50	0.05	27.55	13.49

4.4 Primjena modela

Ljepota dubokog učenja je primjena modela na slične probleme, no s kojima se mreža još nije susrela. Demonstrirano je uklanjanje šuma iz slike veličine 3088x3088 koja je snimljena mobilnim uređajem. Pošto mreža kao ulaz očekuje tenzor (32,32,3), a ne (3088,3088,3), slika je prilagođena i to na 2 načina. Prvi način smanjuje sliku na sliku veličine 32x32 čime se gube mnogi detalji slike, no i dalje je dobivena nova slika koju mreža nije vidjela. Drugi način sliku veće rezolucije cjepka na dijelove od 32x32, pa ih na kraju spoja u cjelovitu sliku. Primijenjeni šum bit će odabrani šum pri treniranju.

Slika 4.20: Uklanjanje šuma iz nove slike smanjene rezolucije

Snimljenoj slici je smanjena rezolucija, dodan šum, te je taj šum otklonjen (slika 4.20). Od oko 16 dB za PSNR se dolazi do poboljšanja koje iznosi 23.26 dB. Mreža je dobro uklonila šum na do sada neviđenoj slici ruže.

Sliku je rascjepkana na manje dijelove, svakom je primjenjen šum, pa otklonjen. Mreža prilično dobro odrađuje svoj posao i na ovakvom primjeru. Detalji su prikazani na slici (4.21) gdje se vide i nedostaci ovakvog pristupa, no rezultati su dakako bolji od očekivanih. Daju se razaznati male sličice od kojih je sastavljena slika veće rezolucije na slici sa otklonjenim šumom, no smatra se kako su to dobri rezultati s obzirom da mreža nije trenirana da uklanja šum sa takvih slika.

Slika 4.21: Detalji slike veće rezolucije

Od PSNR od oko 16 dB se dolazi do 31.47 dB za cijelu sliku koja je prikazana na slici 4.22. Bez detalja se razlika teško vidi.

Slika 4.22: Originalna slika (lijevo) i slika sa uklonjenim šumom (desno)

5. ZAKLJUČAK

Aditivni bijeli Gaussov šum i šum soli i papra uspješno su uklonjeni sa slika iz baza slika CIFAR-10 i MNIST. Mreža će otkloniti većinu šuma sa slike, bez obzira na njegovu količinu. Kvaliteta slike se u svim primjerima višestruko povećava. Duboke neuronske mreže se dobro primjenjuju na ovaj problem unatoč svojoj kompleksnosti. Detaljno razumjeti i dobro primijeniti duboke mreže nije lak zadatak, no ovdje je pokazano kako se to itekako isplati. Što dublje zalazimo u pitanja „zašto“ i „kako“, nailazimo na desetljeća ljudskog istraživanja i razvoja tehnologije koji su nas doveli do danas. Pomoću Keras aplikacijskog korisničkog sučelja implementirati model neuronske mreže je daleko pojednostavljeno i omogućava bilo kome tko je zainteresiran da modelira i trenira barem nekakvu jednostavnu mrežu i možda potakne tu osobu da uroni dublje u svijet neuronskih mreža. Za određivanje pojedinih parametara mreže ni kompleksna matematika ne daje nam odgovore, nego čovjek uz razne pokušaje dolazi do optimalnog rješenja i uz iskustvo ti pokušaji postaju smisleniji. Do nekih parametara smo došli empirijski i time dobili iskustvo koje nosimo pri daljnjem radu sa neuronskim mrežama.

Naravno, ukloniti navedene vrste šuma je moguće sa još boljim rezultatima tako što promijenimo arhitekturu, a promjene zahtijevaju znanja koja se dobivaju radom na tom području. Neke od različitih arhitektura koje pokazuju jako dobre rezultate su pokazane u ovom radu.

Literatura

[1] S. Anwar, N. Barnes: *Real Image Denoising with Feature Attention*, 2020.

[2] K. Wei, Y. Fu, J. Yang, H. Huang: *A Physics-based Noise Formation Model for Extreme Low-light Raw Denoising*, Beijing Institute of Technology, Microsoft Research, 2020.

[3] A. Krull, T. Buchholz, F. Jug: *Noise2Void - Learning Denoising from Single Noisy Images*, Njemačka, 2019.

[4] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, T. Aila: *Noise2Noise: Learning Image Restoration without Clean Data*, 2018.

[5] P. Liu, H. Zhang, K. Zhang, L. Lin, W. Zuo: *Multi-level Wavelet-CNN for Image Restoration*, 2018.

[6] D. Ulyanov, A. Vedaldi, V. Lempitsky: *Deep Image Prior*, 2020.

[7] M. A. Nielsen: *Neural Networks and Deep Learning*, Determination Press, 2015.

[8] D. Kriesel: *A Brief Introduction to Neural Networks*, 2007.
(http://www.dkriesel.com/en/science/neural_networks)

[9] I. Goodfellow, Y. Bengio, A. Courville: *Deep Learning*, MIT Press, 2016.
(http://www.deeplearningbook.org)

[10] O. Ronneberger, P. Fischer, T. Brox: *U-Net: Convolutional Networks for Biomedical Image Segmentation*, University of Freiburg, Njemačka, 2015.

[11] https://keras.io/about/ [9.7.2020.]

[12] S. Ioffe, C. Szegedy: *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015.

Sažetak

Ukloniti šum iz slike pomoću dubokih neuronskih mreža se lakše realizira korištenjem aplikacijskog korisničkog sučelja Keras u programskom jeziku Python. Korištena je U-Net arhitektura bazirana na originalnom radu uz promjenu početnih karata značajki. Prikazane su neuronske mreže i duboko učenje kao potrebno znanje za rješavanje navedenog problema kao i važniji dijelovi programskog koda. Osim za uklanjanje šuma iz slika iz baza slika CIFAR-10 i MNIST, trenirani model se može koristiti i za uklanjanje šuma iz sličnih slika koje moraju odgovarati navedenom očekivanom ulazu u mrežu što pokazuje neovisnost mreže o danim slikama za trening. Izložene su slike sa različitim intenzitetima šuma sa kojih je otklonjen sa prikazanim uspjehom u obliku PSNR vrijednosti iskazanih u decibelima.

Ključne riječi: autokoder, duboko učenje, neuronske mreže, uklanjanje šuma

Summary

Image denoising with deep neural networks

Image denoising using deep neural networks is accomplished more easily using Keras application programming interface and Python programming language. The architecture used is based on original U-Net architecture with a changed starting number of feature maps. Neural networks and deep learning are presented as prerequisites for solving the given problem along with noteworthy code examples. Trained model can be used not only for denoising images from CIFAR-10 and MNIST dataset but also for denoising similar images that are compatible with the network's input layer which portrays the network's independence of given training images. Denoised images with various noise intensity levels are shown with their success levels described with PSNR values in decibels.

Keywords: autoencoder, deep learning, image denoising, neural netoworks

Životopis

Dijana Ivezić rođena je 5. prosinca 1998. u Slavonskom Brodu, Hrvatska. Završila je Tehničku školu Slavonski Brod kao arhitektonski tehničar. Od djetinjstva razvija znanja i vještine vezane uz tehnologiju što je uvijek proizlazilo iz znatiželje. Ljubitelj je glazbe i plesa, kao i rolanja i sličnih sportskih aktivnosti. Upisuje 2017. Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer računarstvo. Osim računarstva, zanimaju je i često proučava teme iz područja fizike i matematike koje ne moraju biti neophodne u profesionalnom radu.