SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2003

Duboki modeli za uklanjanje šuma u slikama iscrtanim slučajnim uzorkovanjem

Nikola Bunjevac

Zagreb, srpanj 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 15. ožujka 2019.

DIPLOMSKI ZADATAK br. 2003

Pristupnik: **Nikola Bunjevac (0036485677)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Duboki modeli za uklanjanje šuma u slikama iscrtanim slučajnim uzorkovanjem**

Opis zadatka:

Mnoge moderne biblioteke za vizualizaciju 3D scena generiraju slike primjenom Monte Carlo metoda. Takav pristup podrazumijeva računanje osvjetljenja svakog djelića scene usrednjavanjem doprinosa nasumičnog uzorka svih ostalih točaka scene. Zbog toga je ta metoda računski jako zahtjevna. Problemi se mogu umanjiti donošenjem odluke na temelju relativno malih uzoraka. Željenu kvalitetu iscrtavanja jeftinije je dobiti otklanjanjem šuma iz tako dobivene slike nego uzorkovanjem dovoljno velikog broja točaka scene.
U okviru rada, potrebno je proučiti i ukratko opisati postojeće pristupe za otklanjanje šuma nastalog uslijed nedovoljno velikog uzorka za računanje osvjetljenja. Uhodati postupke nadziranog učenja prikladnih modela. Validirati hiperparametre te prikazati i ocijeniti ostvarene rezultate. Predložiti pravce budućeg razvoja.
Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne sljedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 28. lipnja 2019.

Mentor:
Prof. dr. sc. Siniša Šegvić

Predsjednik odbora za diplomski rad profila:
Doc. dr. sc. Marko Čupić

Djelovođa:
Izv. prof. dr. sc. Tomislav Hrkać

Zahvaljujem prof. dr. sc. Siniši Šegviću na mentorstvu tijekom studija te svojoj obitelji i prijateljima na podršci tijekom cijelog školovanja.

SADRŽAJ

1. Uvod

Računalna grafika je područje računarske znanosti koje se bavi izradom i prikazom slika i video zapisa. Raznim algoritmima i metodama moguće je iz opisa scene pohranjenog u računalu dobiti vrlo vjeran i fizikalno točan prikaz te scene. Takve metode oslanjaju se na simulaciju interakcije svjetlosti s objektima i površinama. Da bi simulacija bila izvediva posežemo za slučajnim uzorkovanjem čime izbjegavamo potrebu za potpunom simulacijom stvarnosti. Matematički alati iz teorije vjerojatnosti nam garantiraju da će i takva nasumična, tj. nepotpuna simulacija dati rezultat koji je u prosjeku točan.

Jedan od algoritama koji daje takav vjeran rezultat je algoritam praćenja zrake (engl. *ray tracing*) ili algoritam praćenja puta (engl. *path tracing*). On se temelji na ideji da za svaki piksel slike pratimo zraku do prvog presjecišta u sceni te uzorkujemo izvore svjetlosti iz te točke u prostoru. Osim izvora svjetlosti, uzorkujemo i daljnje refleksije te zrake.

Iako je moguće i jasno kako dobiti zadovoljavajući prikaz scene, takav postupak može biti izrazito dugotrajan. Jedno moguće rješenje je iskoristiti mogućnost paralelizma koji se prirodno uklapa u algoritam praćenja putanje. Naime, svaka zraka je neovisna o drugim zrakama i moguće je računati doprinos više zraka istovremeno. U ovom radu ćemo opisati drugi pristup, a to je korištenje dubokog učenja. Razlog zašto je to prikladan pristup ovom problemu jest što možemo generirati parove slika s prisutnim šumom i referentnih slika bez, ili s prihvatljivom količinom šuma. Tada je potrebno pronaći preslikavanje iz slike s prisutnim šumom u sliku s uklonjenim šumom. Za to ćemo koristiti duboke modele.

U drugom poglavlju uvodimo osnovne pojmove strojnog, odnosno dubokog učenja. Gradivne jedinice bit će nam konvolucijski slojevi čija su svojstva i prednosti objašnjeni. Zatim su opisane neke značajnije aktivacijske funkcije koje unose nelinearnost u model, a opisani su i korišteni algoritmi učenja dubokog modela. Kao nadogradnja, korišteni su i rezidualni duboki modeli koji rješavaju probleme specifične za modele s mnogo slojeva te osiguravaju brže i stabilnije učenje.

Treće poglavlje posvećeno je detaljnijem opisu problema koji rješavamo. Prvo je opisan algoritam praćenja zrake koji omogućava iscrtavanje fotorealističnih prikaza. Kako bi rezultat iscrtavanja bio kvalitetan, koristimo slučajno uzorkovanje čime se postiže efikasnost. Nedovoljan broj uzoraka dovodi do karakterističnog šuma koji želimo ukloniti opisanim me-

todama.

U četvrtom poglavlju opisan je skup podataka za učenje. Kako ćemo koristiti nadzirano učenje, potrebni su nam parovi podataka sačinjeni od ulaza i željenog izlaza. U našem slučaju to će biti slika s prisutnim šumom te slika bez šuma. Kako bi se postigao bolji rezultat, potrebno je obaviti i određeno pretprocesiranje podataka koje je također opisano u spomenutom poglavlju.

Peto poglavlje opisuje korištene modele. To su tri modela kod kojih svaki pokušava ispraviti nedostatke prethodnog. Prvi i najjednostavniji model kao izlaz daje sliku bez šuma. Drugi model umjesto slike daje jezgre čijom primjenom ćemo ukloniti šum. Ova dva modela se sastoje od podmreža s odvojenim parametrima koje treći model pokušava eliminirati koristeći dijeljene parametre podmreža, odnosno efektivno jednu podmrežu.

U šestom poglavlju prezentirani su postignuti rezultati eksperimenata. Oni uključuju grafove učenja svakog modela, i to na skupu za učenje i validacijskom skupu. Prikazani su i određeni problemi koje modeli imaju te su prikazani primjeri uklanjanja šuma.

2. Duboko učenje

Svaki algoritam strojnog učenja možemo opisati pomoću tri komponente:

1. Model, $h(\mathbf{x}|\boldsymbol{\theta})$, gdje je $\mathbf{x}$ ulazni podatak, a $\boldsymbol{\theta}$ parametri modela.

2. Funkcija gubitka, koja govori koliko je kvalitetna predikcija za podatak $\mathbf{x}$ uz parametre modela $\boldsymbol{\theta}$.

3. Optimizacijski postupak, koji koristi funkciju gubitka kako bi poboljšao kvalitetu predviđanja algoritma (npr. traženje optimalnih parametara modela za koje funkcija gubitka daje najmanju pogrešku pomoću gradijentnog spusta).

U ostatku poglavlja ćemo pojasniti pojedine komponente.

Ideje na kojima se zasniva duboko učenje nisu nove, ali došle su do izražaja tek razvojem tehnologije, posebno grafičkih kartica (engl. *graphical processing unit, GPU*). To je omogućilo izvođenje algoritama koji do sada nisu bili praktični za izvođenje. Osnovna ideja ovog područja je i u samom nazivu–učenje. Da bismo mogli učiti, potrebno je imati podatke. Oni su ključni u cijelom postupku, i o njima ovisi kvaliteta rezultata. Te podatke nazvat ćemo skup podataka za učenje (engl. *dataset*). Podatke ćemo još nazivati i značajkama (engl. *features*).

Postupke učenja ugrubo možemo podijeliti na

- nadzirane,
- nenadzirane i
- podržano učenje.

Postupci nadziranog učenja su oni koji koriste parove ulaza i očekivanih izlaza (skup podataka je označen) te im je cilj za dani ulaz dobiti izlaz što bliži referentnom. Još se koristi i naziv učenje uz pomoć učitelja jer možemo ovo shvatiti kao da imamo nekoga tko će nam pokazati kakav rezultat trebamo dobiti, odnosno znamo čemu trebamo težiti.

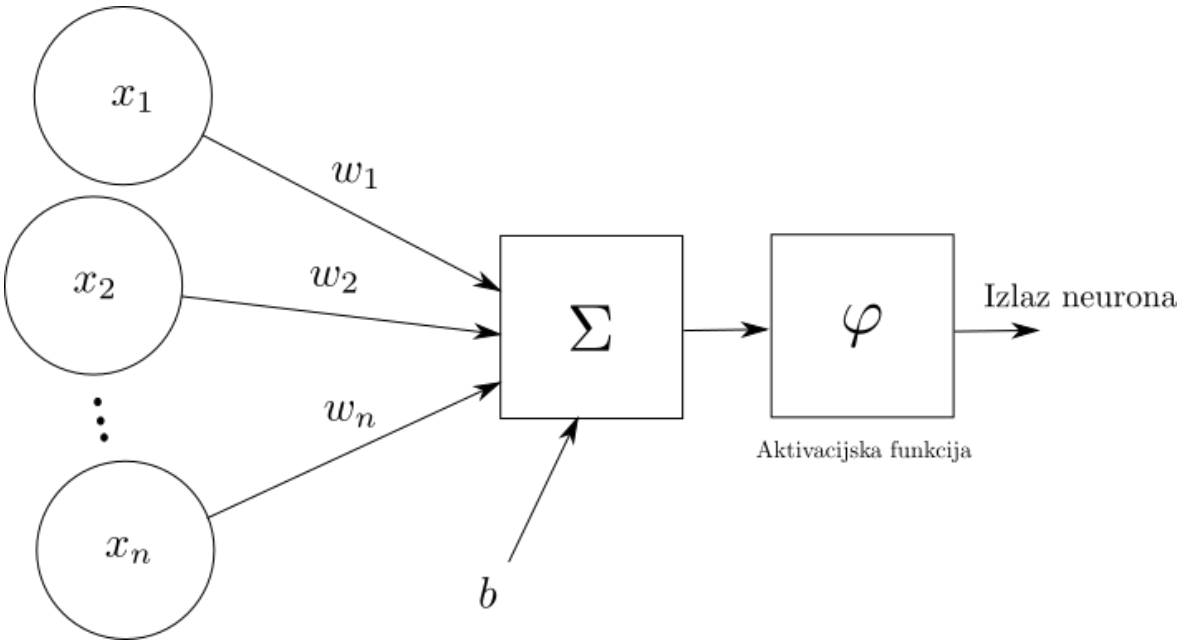
Nenadzirano učenje se još naziva i učenje bez učitelja. Ovdje nemamo parove ulaza i očekivanog izlaza već samo ulazne značajke. Tada želimo saznati neke korisne informacije o tim podacima, primjerice kako su grupirani, koja im je vjerojatnosna distribucija i slično.

Spomenimo još i sve popularnije podržano učenje (engl. *reinforcement learning*) kod kojega nemamo fiksan skup podataka, već model učimo u interakciji s okolinom, na primjer u igri. Model tada na temelju povratne veze uči i donosi odluke.

Objasnimo sada još i što točno učimo kod modela dubokog učenja. Oni se sastoje od slojeva koji se sastoje od neurona. Neuron možemo shvatiti kao funkciju koja linearno transformira ulaze i zatim dobiveni rezultat provuče kroz aktivacijsku funkciju. Matematički to možemo zapisati kao

$$f(\mathbf{x}) = \varphi(\mathbf{x} \cdot \mathbf{w} + b), \tag{2.1}$$

gdje je $\mathbf{x}$ vektor ulaznih značajki, $\mathbf{w}$ vektor težina, φ aktivacijska funkcija, a b pomak (engl. *bias*).



Slika 2.1: Prikaz matematičkog modela neurona.

2.1. Duboki konvolucijski modeli

Konvolucija je matematička operacija koju možemo opisati izrazom

$$s(t) = \int_{-\infty}^{\infty} x(\tau) w(t - \tau) d\tau$$

u kontinuiranoj domeni. Funkciju $x(t)$ nazivamo ulaznom funkcijom, a funkciju $w(t)$ jezgrom ili filterom. Izlaz $s(t)$ nazivamo mapom značajki. Međutim, kod konvolucijskih mreža često radimo sa slikama što znači da želimo diskretnu domenu. Diskretnu konvoluciju računamo kao

$$s(t) = \sum_{a=-\infty}^{\infty} x(a) w(t - a),$$

pri čemu se funkcije nazivaju analogno kontinuiranoj verziji.

Konvolucija se često označava zvjezdicom na sljedeći način

$$s(t) = (x * w)(t).$$

Iako je operacija konvolucije u dubokim modelima bliska prethodnom opisu, ona je u pravilu jednostavnija. To je zato što radimo s konačnim podacima, primjerice slikama. Jezgre su veličine $k \times k$, pri čemu je k neparan, zbog simetričnosti. Još je potrebno znati korak kojim će se jezgra "kretati" kroz sliku. Ovdje možemo uočiti potencijalan problem: što učiniti s vrijednostima koje se nalaze na rubovima slike? U praksi postoje 2 rješenja. Jedno je da se oko slike rub širine $(k-1)/2$ popuni (engl. *padding*) konstantnim vrijednostima, npr. nulom. Time osiguravamo da neće doći do problema gdje jezgra izlazi izvan okvira podataka. Drugi način je jednostavno ne raditi konvoluciju na područjima koja ne mogu u potpunosti biti obuhvaćena cijelom jezgrom (engl. *valid convolution*). Time gubimo rub širine $(k-1)/2$ na izlazu iz konvolucijskog sloja.

Duboki konvolucijski modeli su modeli koji u barem jednom sloju sadrže konvoluciju. Ona se ponekad kombinira s operacijom sažimanja (engl. *pooling*) koja se obično dodaje nakon više slojeva te aktivacijskom funkcijom (najčešće ReLU). Iako slojeve sažimanja ne koristimo u modelima, navedeno je radi kompletnosti.

Operacija sažimanja (engl. *pooling*) je relativno jednostavna. Postoji više varijanti sažimanja, a česte su one koje koriste maksimum ili srednja vrijednost. Ideja sažimanja jest smanjiti broj značajki koji ide u sljedeći sloj. Operacija se vrši analogno konvoluciji tako da se jezgra kreće mapom značajki i vrši odgovarajuću operaciju nad jezgrom, a rezultat se prenosi u sljedeći sloj. U pravilu se koristi korak koji će osigurati da ne dođe do preklapanja što povećava učinkovitost ove operacije (primjerice jezgra 2×2 uz korak 2).

Tri su značajne prednosti konvolucijskih modela spram ostalih modela dubokog učenja [7]. To su

1. rijetke interakcije,
2. dijeljenje parametara i
3. ekvivarijantne reprezentacije.

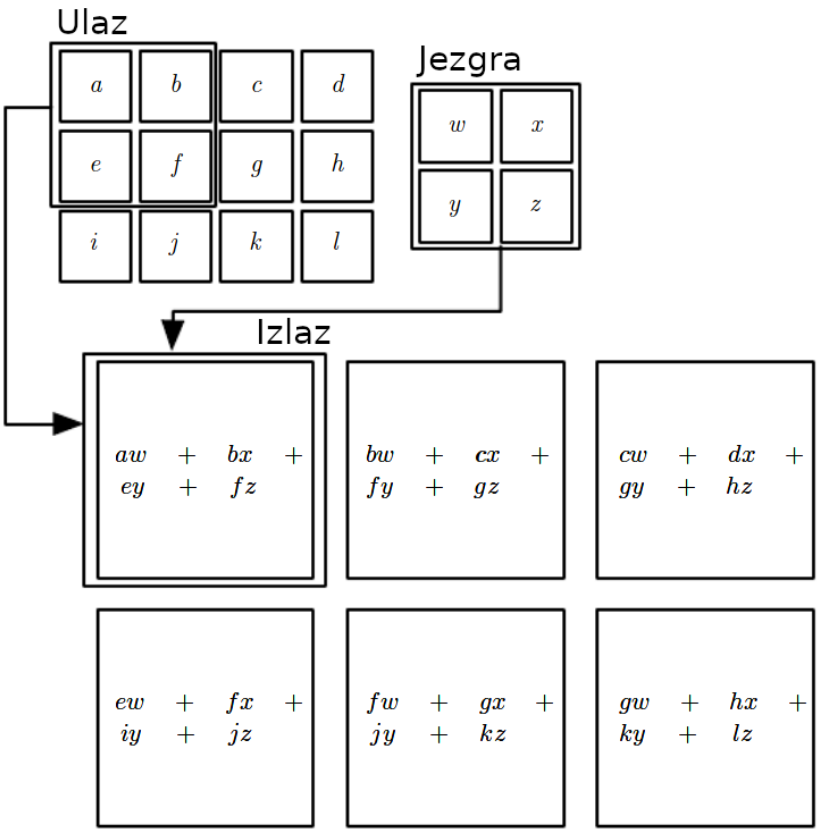
Svojstvo rijetkih interakcija posljedica je toga što je konvolucijska jezgra uobičajeno mnogo manja od ulaza. Potpuno povezani slojevi modeliraju se kao matrični umnožak matrice parametara s matricom ulaznih podataka. To znači da moramo modelirati vezu između svakog pojedinog ulaza i parametra što dovodi do bespotrebnih veza. Konvoluciju provodimo nad lokalnim skupovima vrijednosti ulaznih podataka. Svaki izlaz iz konvolucijskog sloja nije povezan sa svim ulaznim vrijednostima, već samo s onima koji su obuhvaćeni jezgrom konvolucije, tj. susjedstvom. Time se postiže mnogo efikasnija izvedba jer zahtijeva

višestruko manje računanja. Vrijednosti obuhvaćene konvolucijskom jezgrom nazivaju se receptivno polje. Konvolucijski slojevi koji se nalaze dublje unutar neuronske mreže imaju veće receptivno polje.

Drugo korisno svojstvo je dijeljenje parametara. Konvolucijski sloj može imati više jezgri, primjerice 32 ili 64, no svaka jezgra ima vlastite parametre koje primjenjuje na sve ulazne podatke. Ti parametri su nakon učenja nepromjenjivi. Isti parametri primjenjuju se na sve lokacije nad kojima se provodi operacija konvolucije (za istu jezgru). Time se postiže memorijska efikasnost jer ne moramo pamtiti težine za svaku vezu neurona sa svakom ulaznom vrijednosti.

Naposljetku, pojasnimo svojstvo ekvivarijantne reprezentacije. Ono je posljedica dijeljenja parametara. Kada kažemo da je neka funkcija ekvivarijantna, to znači da kako se mijenja njen ulaz, tako se mijenja i njen izlaz. Konvolucija je ekvivarijantna na translaciju što znači da ako imamo pomak u ulaznim podacima, takav pomak ćemo imati i na izlazu. Pretpostavimo da želimo detektirati rub na slici. Ako sve piksele slike pomaknemo udesno, opet ćemo moći detektirati rub jer će on sadržati svojstvo po kojemu ga možemo detektirati, samo će se nalaziti na drugom mjestu na slici. Ovo svojstvo je poželjno jer ne želimo naučiti detektirati značajke samo na određenom dijelu slike, već obično želimo da detekcija bude robusna s obzirom na translaciju.

Jedna od pretpostavki dubokih modela jest hijerarhijska struktura podataka nad kojima se uči. Time svaki sljedeći sloj postaje sve specijaliziraniji u svojoj zadaći. Na primjer, ako želimo prepoznati nalazi li se mačka na slici, model bi prvo mogao pronaći uši, zatim brkove, zatim šape, itd. To ima smisla jer što se sloj dublje nalazi, to on ima veće receptivno polje. Stoga, početni slojevi traže detaljnije značajke, a daljnji slojevi mogu generalizirati i tražiti općenitije ili veće značajke.



Slika 2.2: Prikaz konvolucije ulaza s jezgrom veličine 2×2 bez nadopunjavanja (engl. *padding*). Slika preuzeta iz [7].

2.2. Aktivacijske funkcije

Kako izlaz modela ne bi bila samo linearna kombinacija ulaza, na izlaze iz pojedinih slojeva možemo dodati funkcije kroz koje ćemo "provući" pripadajući izlaz. Njih nazivamo aktivacijskim funkcijama. Osim unošenja nelinearnosti, postoje i drugi razlozi zašto ih koristimo.

Najčešće korištena aktivacijska funkcije je ReLU(engl. ***Re**ctified **L**inear **U**nit*) koja je definirana izrazom

$$\mathrm{ReLU}(x) = \max(0, x). \tag{2.2}$$

Možemo primijetiti kako u 0 nema derivacije, međutim to u praksi ne predstavlja problem. Jedan od glavnih razloga korištenja aktivacijske funkcije ReLUje njena jednostavnost. Mnogo je efikasnije izračunati funkciju maksimum nego primjerice e^x ili pak dijeljenje. Na prvi pogled se može činiti pretjerano paziti na takve detalje, ali treba imati na umu kako duboki modeli mogu imati veliki broj slojeva pri čemu se može izvoditi iznimno mnogo operacija i svaka optimizacija je dobrodošla, pa tako i u aktivacijskoj funkciji.

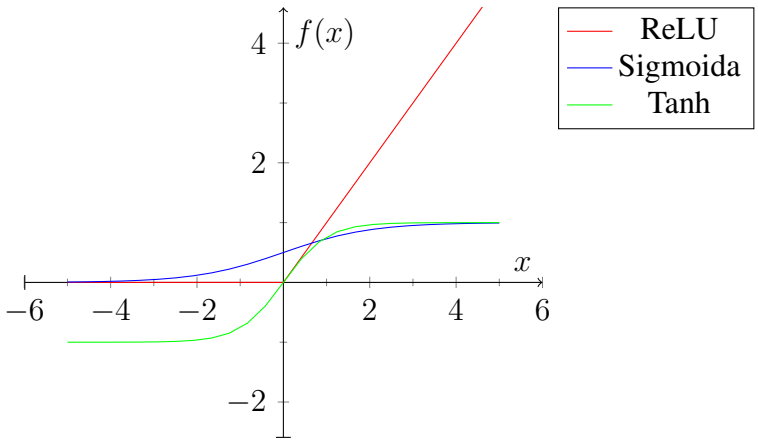
Još neke značajnije aktivacijske funkcije su sigmoida i tangens hiperbolni, također pri-

kazani na slici 2.3. Sigmoida je definirana kao

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.3}$$

dok je tangens hiperbolni definiran kao

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.4}$$



Slika 2.3: Grafovi češće korištenih aktivacijskih funkcija.

Spomenimo još i funkciju $\mathrm{softmax}$ koja je definirana kao

$$\mathrm{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}, \tag{2.5}$$

gdje je $\mathbf{x}$ vektor veličine n. Njenom primjenom normaliziramo ulazne vrijednosti tako da su sve u rasponu $[0, 1]$ te vrijedi $\sum_{i=1}^{n} \mathrm{softmax}(\mathbf{x})_i = 1$.

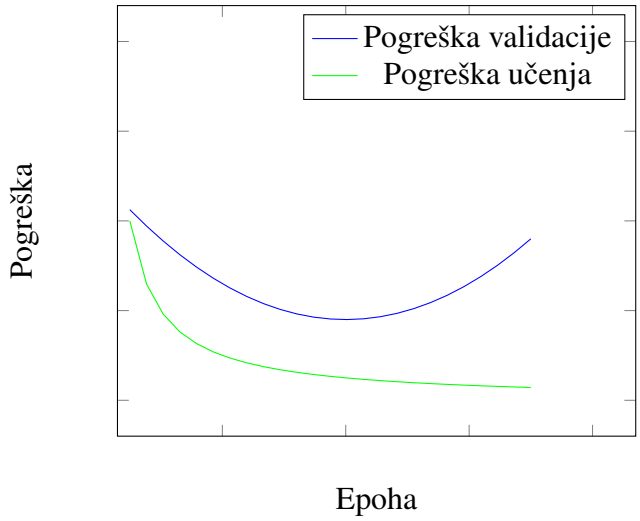
2.3. Optimizacijski postupak

Kako bi duboki model mogao učiti nad podacima, potrebno je imati mjeru koja nam govori koliko je model naučio podatke. Tu mjeru nazivamo funkcijom pogreške ili gubitka (engl. *loss function*). Na temelju nje možemo "gurati" model u smjeru poboljšanja tako da minimiziramo pogrešku na skupu podataka za učenje. Glavna ideja takve metode optimizacije je u svakoj iteraciji algoritma izračunati pogrešku, a zatim odrediti "krivce" za takvu pogrešku. Kada znamo koji dijelovi modela su koliko odgovorni, ovisno o doprinosu ćemo ih korigirati kako bi se bolje prilagodili podacima.

Prilikom učenja modela strojnog učenja provjeravamo napredak tako da, uz skup za učenje, imamo i skup za validaciju. Model učimo na skupu za učenje, a nakon svake epohe (prolaska kroz cijeli skup za učenje), provjeravamo pogrešku na skupu za validaciju. Važno

je da model “ne vidi” validacijski skup, odnosno ne uči na njemu, kako bismo dobili što realniju sliku o trenutnom napretku, tj. generalizaciji.

Standardno kretanje pogreške možemo vidjeti na slici 2.4. Vidimo kako pogreške konstantno padaju do određenog trenutka kada pogreška validacije počne rasti. Stanje lijevo od te točke nazivano podnaučenost (engl. *underfitting*), a stanje desno od te točke nazivamo prenaučenost (engl. *overfitting*). Podnaučenost je jasna jer se model još nije dovoljno prilagodio podacima–nije dovoljno naučen. Kod prenaučenosti, model se jako dobro prilagodio podacima (“naštrebao” ih je), dok je kod skupa za validaciju postao lošiji. Kažemo još da takav model loše generalizira, odnosno daje loše rezultate na neviđenim podacima. To je nepoželjno svojstvo jer je svrha strojnog učenja na ograničenoj količini podataka postići veliku sposobnost generalizacije.



Slika 2.4: Standardni graf strojnog učenja.

Tu prilagodbu izvodimo na temelju gradijenta funkcije gubitka. Stoga se osnovni algoritam optimizacije naziva gradijentni spust. U najjednostavnijoj varijanti izračunavamo gradijent na temelju cijelog skupa podataka za učenje što može dovesti do lošijih rezultata, na primjer prevelikih pomaka parametara ili sporog učenja. Druga moguća varijanta je osvježavati parametre nakon svakog primjera za učenje, no najčešće se koristi kompromis–učenje na malim grupama ulaznih primjera (engl. *minibatch*). Tako radimo male pomake više puta tijekom jedne epohe učenja umjesto samo jednom na kraju epohe (potencijalno preveliki pomak) ili nakon svakog primjera (potencijalno premali korak). Taj algoritam nazivamo stohastički gradijentni spust i njegov opis se nalazi u pseudokodu 1.

Pseudokod 1 Stohastički gradijentni spust

Ulaz: Korak učenja ϵ

Ulaz: Početni parametri $\boldsymbol{\theta}$

dok uvjet završetka nije zadovoljen **radi**

  Uzorkuj mini grupu veličine m iz skupa za učenje $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ s pripadajućim referentnim vrijednostima $\mathbf{y}^{(i)}$

  Izračunaj gradijent: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$

  Osvježi parametre: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon \mathbf{g}$

kraj dok

Iako je stohastički gradijentni spust vrlo dobar algoritam, s vremenom su se pojavila i poboljšanja. Jedan od osnovnih problema gradijentnog spusta je odrediti korak učenja. O njemu ovisi koliko brzo će model učiti te hoće li uopće učiti ili će možda divergirati. Još jedan algoritam koji se često koristi je Adam [9], a naziv mu potječe od engleskog izraza *adaptive moment estimation* (prilagodljiva procjena momenata). Njegova ideja je ne imati fiksni korak učenja već on ovisi o prošlim pomacima algoritma. Na temelju te prošlosti, Adam prilagođava brzinu učenja čime se postiže robusnost na odabir hiperparametara.

Pseudokod 2 Algoritam Adam

Ulaz: Korak učenja ϵ

Ulaz: Faktori odumiranja za procjene momenata, ρ_1 i ρ_2 iz $[0, 1)$ (preporučene vrijednosti 0.9 i 0.999)

Ulaz: Mala konstanta δ za numeričku stabilizaciju (preporučena vrijednost 10^{-8})

Ulaz: Početni parametri $\boldsymbol{\theta}$

Inicijaliziraj varijable prvog i drugog momenta: $\mathbf{s} = 0, \mathbf{r} = 0$

Inicijaliziraj vremenski korak: $t = 0$

dok uvjet završetka nije zadovoljen **radi**

  Uzorkuj mini grupu veličine m iz skupa za učenje $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ s pripadajućim referentnim vrijednostima $\mathbf{y}^{(i)}$

  Izračunaj gradijent: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$

  $t \leftarrow t + 1$

  Osvježi pristranu procjenu prvog momenta. $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\mathbf{g}$

  Osvježi pristranu procjenu drugog momenta: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\mathbf{g} \odot \mathbf{g}$

  Popravi pristranost procjene prvog momenta: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}$

  Popravi pristranost procjene drugog momenta: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t}$

  Izračunaj promjenu: $\delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$

  Osvježi parametre: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}$

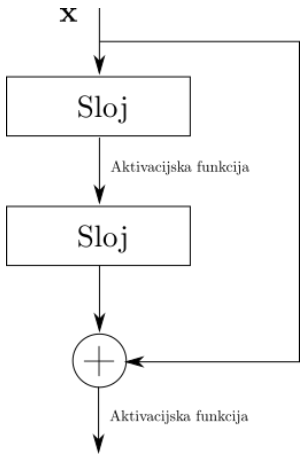
kraj dok

2.4. Duboki rezidualni modeli

Duboki modeli s mnogo slojeva mogu imati probleme od kojih ne pate modeli s malim brojem slojeva. To su između ostalog nestajući gradijenti (engl. *vanishing gradients*). Dublji slojevi modela mogu ostati "zakinuti" zbog svoje udaljenosti i gradijenti mogu postati približno nula. Analogan problem su i eksplodirajući gradijenti (engl. *exploding gradients*). Ti problemi su uglavnom riješeni normaliziranom inicijalizacijom parametara modela te normalizacijskim slojevima. Rješavanjem tih problema do izražaja je došao novi problem, a to je degradacija. Povećanjem broja slojeva točnost modela postaje bolja, a zatim naglo degradira. Taj problem nije uzrokovan prenaučenošću, i dodavanjem dodatnih slojeva povećavamo pogrešku učenja modela.

Članak [8] predlaže uvođenje preskočnih veza između slojeva koje će spriječiti navedeni problem degradacije. Time se omogućuje sljedećim slojevima da se oslanjaju na aktivacije prethodnih slojeva što također dovodi do stabilnijeg i bržega učenja. Mrežu koja sadrži rezidualne blokove još nazivamo i ResNet.

Osnovnu strukturu rezidualnog modela možemo vidjeti na slici 2.5.



Slika 2.5: Primjer rezidualnog bloka [8].

3. Problem

Sada ćemo opisati kako se odvija postupak iscrtavanja (engl. *rendering*) te koji njegov nedostatak želimo riješiti ili barem ublažiti.

Cilj iscrtavanja je iz opisa scene u računalu dobiti prikaz te scene koji možemo vidjeti na zaslonu računala, ispisati ga ili na neki drugi način vizualizirati. Ograničit ćemo se na prikaz na zaslonu računala. To znači da će se slika sastojati od slikovnih elemenata koje nazivamo pikselima (engl. *picture element, pixel*). Svaki piksel možemo opisati pomoću trojke crvene, zelene i plave boje (engl. *red, green, blue; RGB*).

Opis scene obično je zapisan u tekstualnom formatu, a sadrži specifikacije oblika, modela, pripadajućih materijala i dodatnih svojstava poput transformacija. Još su nam potrebni neki podaci kako bi opis bio potpun, a to su

- kamera - određuje poziciju s koje će se prikaz scene generirati te opisuje senzor svjetlosti (kako će se svjetlost zabilježiti) i hoće li biti kakvih deformacija nad ispaljenim zrakama (aproksimacija leće),
- izvori svjetlosti - određuju kako i koji dijelovi scene će biti osvijetljeni; mogu biti raznih oblika, na primjer točkasti, površinski, globalni (sunčeva svjetlost). Bez izvora svjetlosti ne možemo generirati prikaz scene, odnosno izlaz bi bio crna boja u svakom pikselu.

Kompleksnije scene mogu sadržavati još dodatnih svojstava poput opisa prostornih svojstava koja utječu na širenje svjetlosti (razni mediji). Svaki objekt u sceni mora imati opis interakcije sebe sa svjetlošću. To znači je li objekt proziran ili ne, kako će se svjetlost reflektirati, itd. Taj opis obično u sebi sadržava materijal koji je pridružen objektu. Korisno je napomenuti i kako su objekti u sceni obično sazdani od poligona, konkretnije trokuta, što je uobičajeno u računalnoj grafici.

3.1. Algoritam praćenja zrake

Nakon što imamo opis scene, potrebno ga je učitati, *parsirati* i izraditi opis koji možemo efikasno koristiti prilikom iscrtavanja. Tom prilikom se također izgrađuju i strukture za ubrzavanje rada poput hijerarhije obujmica (engl. *bounding volume hierarchies, BVH*) koja služi

za brzo određivanje relevantnih dijelova scene kod traženja presjecišta. Kada smo učitali sve potrebne informacije i obavili sve predradnje, može početi postupak iscrtavanja.

Iako postoji više algoritama iscrtavanja, mi ćemo se usredotočiti na algoritam praćenja zrake (engl. *ray tracing*). Razlog je što on omogućava efektivno simuliranje fizikalnih svojstava širenja i interakcije svjetlosti s površinama što dovodi do vrlo kvalitetnih (fotorealističnih) rezultata. Također je robustan jer se ne oslanjamo suviše na razne optimizacije, aproksimacije i rubne slučajeve, već na sveobuhvatnost algoritma.

Ključno pitanje algoritma iscrtavanja je odrediti boju svakog pojedinog piksela slike koju generiramo. Praćenje zrake započinje od (virtualne) kamere. Za svaki piksel slike koju želimo dobiti iz kamere, "ispucat" ćemo zraku i pratiti njenu interakciju sa scenom. To znači da moramo izračunati količinu svjetlosti koja dopire do kamere duž te konkretne zrake. To ćemo postići tako da nađemo prvo presjecište zrake u sceni te uzorkujemo izvore svjetlosti. Nakon toga možemo rekurzivno ponoviti postupak te ispaliti još zraka uz pretpostavku da je trenutno sjecište sada izvor nove zrake. Sve daljnje doprinose ćemo pribrojiti početnoj zraci te tako dobiti i efekt refleksija.

Kako bismo dobili što kvalitetniji rezultat u prihvatljivom vremenu, potrebno je usredotočiti se na korak uzorkovanja izvora svjetlosti. Jedna mogućnost je uzorkovati sve izvore svjetlosti, no to može dovesti do problema. U slučaju da imamo tamo točkaste izvore svjetlosti, neće biti većih problema, no, u praksi često nalazimo puno kompleksnije izvore svjetlosti koji mogu imati površinu te ih nije moguće jednostavno uzorkovati.

U praksi je to dovelo do korištenja slučajnog uzorkovanja.

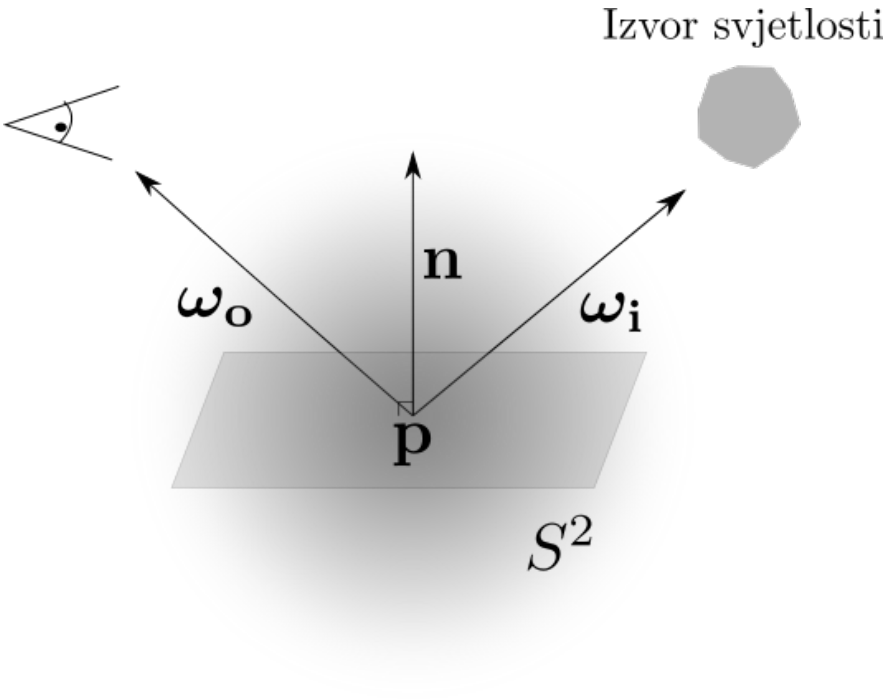
3.2. Slučajno uzorkovanje

Postupak iscrtavanja možemo formalno zapisati kao rješavanje jednadžbe

$$L_o(\mathbf{p}, \boldsymbol{\omega_o}) = L_e(\mathbf{p}, \boldsymbol{\omega_o}) + \int_{S^2} f(\mathbf{p}, \boldsymbol{\omega_o}, \boldsymbol{\omega_i}) L_i(\mathbf{p}, \boldsymbol{\omega_i}) |\cos\theta_i| \mathrm{d}\boldsymbol{\omega_i}, \tag{3.1}$$

gdje $L_o(\mathbf{p}, \boldsymbol{\omega_o})$ predstavlja količinu svjetlosti u točki prostora $\mathbf{p}$ duž vektora $\boldsymbol{\omega_o}$. Ona se sastoji od svjetlosti koju objekt u toj točki emitira (ako je i sam izvor) $L_e(\mathbf{p}, \boldsymbol{\omega_o})$ te svjetlosti koja dopire do točke $\mathbf{p}$ iz svih smjerova oko te točke skaliranih funkcijom $f(\mathbf{p}, \boldsymbol{\omega_o}, \boldsymbol{\omega_i})$ te kosinusnim članom $|\cos\theta_i|$ gdje θ_i predstavlja kut između normale u točki $\mathbf{p}$ i vektora $\boldsymbol{\omega_i}$. Funkcija f opisuje svojstva materijala u točki $\mathbf{p}$, odnosno matematički modelira koliko svjetla će se reflektirati od površine, a koliko će proći kroz nju. Nazivamo ju funkcijom dvosmjerne distribucije raspršenja (engl. *bidirectional scattering distribution function, BSDF*). Ona se zapravo sastoji od dva dijela, a to su funkcija dvosmjerne distribucije reflektancije (engl. *bidirectional reflectance distribution function, BRDF*) i funkcija dvosmjerne distribucije transmitancije (engl. *bidirectional transmittance distribution function, BTDF*). Razlika

je što BRDF vrijedi za slučaj kada su ω_o i ω_i u istoj polusferi oko točke $\mathbf{p}$, dok BTDF vrijedi u slučaju kada su ω_o i ω_i u suprotnim polusferama. Objedinjavanjem tih dvaju funkcija dobivamo funkciju koja vrijedi za cijelu sferu S^2. Jednadžbu 3.1 nazivamo jednadžbom širenja svjetlosti (engl. *light transport equation*) ili jednadžbom iscrtavanja (engl. *rendering equation*) [10]. Slika 3.1 prikazuje glavne pojmove jednadžbe 3.1.



Slika 3.1: Prikaz pojmova jednadžbe iscrtavanja.

Praćenje zrake sada možemo shvatiti kao rješavanje jednadžbe 3.1 u svakoj točki presjecišta zrake iz kamere s objektom u sceni. No, problem je što jednadžba nema analitičko rješenje, osim za najjednostavnije slučajeve. Stoga pribjegavamo postupcima numeričke integracije, konkretno Monte Carlo integracije.

3.3. Monte Carlo integracija

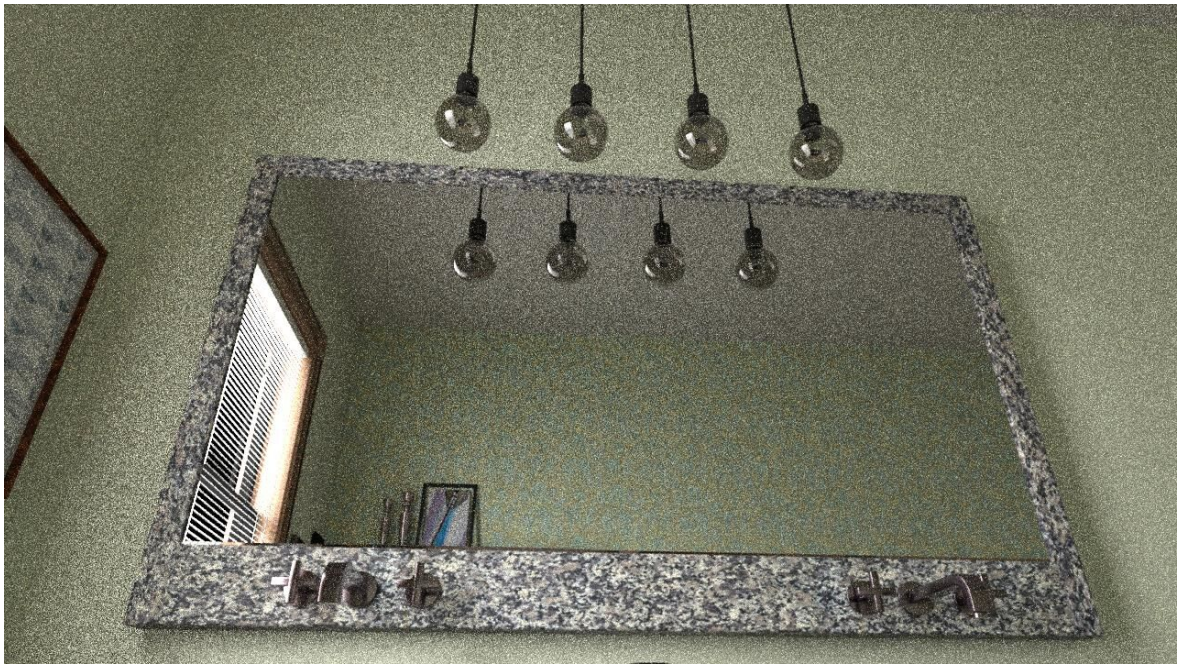
Metode slučajnog uzorkovanja pokazale su se kao vrlo značajan alat u raznim postupcima pa su tako našle svoj put i do računalne grafike. Trenutno najbolje (najbliže fizici) rezultate daju metode iscrtavanja koje koriste Monte Carlo integraciju za rješavanje jednadžbe 3.1. Razlog tome je što ona daje odlične rezultate za integrale visoke dimenzionalnosti koji mogu sadržavati i diskontinuitete, što nije slučaj kod ostalih metoda numeričke integracije.

Još jedna prednost ove metode integracije je što nije potrebno znati funkciju koju integriramo. Drugim riječima, možemo ju tretirati kao crnu kutiju koja će nam za dani ulaz vratiti pripadajuću vrijednost funkcije. To uvelike olakšava implementaciju, odnosno čini

ovu metodu modularnom.

Nedostatak Monte Carlo integracije je korak konvergencije koji iznosi $\mathcal{O}(n^{-1/2})$ (n je broj uzoraka) što znači da kako bismo pogrešku smanjili upola, potrebno nam je četiri puta više uzoraka, ili u našem slučaju "ispucanih" zraka.

Poduzorkovanje se manifestira kao šum koji je vidljiv na slici 3.2



Slika 3.2: Primjer šuma uz nedovoljan broj uzoraka. Slika je preuzeta iz skupa za učenje.

Pokažimo sada Monte Carlo procjenitelj. Pretpostavimo da želimo izračunati integral $\int_a^b f(x)\mathrm{d}x$. Uz uniformno distribuiranu slučajnu varijablu $X_i \in [a, b]$ očekivana vrijednost Monte Carlo procjenitelja

$$F_N = \frac{b-a}{N} \sum_{i=1}^{N} f(X_i), \tag{3.2}$$

$E[F_N]$ je jednaka vrijednosti integrala. Očekivanje kontinuirane varijable je definirano kao

$$E[f(x)] = \int_D f(x)p(x)\mathrm{d}x, \tag{3.3}$$

gdje je D domena, a $p(x)$ razdioba vrijednosti funkcije $f(x)$. Slijedi

$$\begin{aligned} E[F_N] &= E\left[\frac{b-a}{N}\sum_{i=1}^{N} f(X_i)\right] \\ &= \frac{b-a}{N}\sum_{i=1}^{N} E[f(X_i)] \\ &= \frac{b-a}{N}\sum_{i=1}^{N}\int_a^b f(x)p(x)\mathrm{d}x \\ &= \frac{b-a}{N}\sum_{i=1}^{N}\int_a^b \frac{f(x)}{b-a}\mathrm{d}x \\ &= \frac{1}{N}\sum_{i=1}^{N}\int_a^b f(x)\mathrm{d}x \\ &= \int_a^b f(x)\mathrm{d}x. \end{aligned} \tag{3.4}$$

Dakle, kako bismo dobili rezultat integrala pomoću Monte Carlo metode integracije, potrebno je uprosječiti rezultate pojedinih uzoraka. Uz veliki broj uzoraka dobit ćemo rezultat koji je vrlo blizu točnome.

Spomenimo još kako i osim porodice Monte Carlo metoda postoje i Las Vegas metode. Razlika je u tome što Las Vegas metode koriste nasumičnost, ali uvijek daju isti rezultat (ili jave da ga nisu pronašle), dok Monte Carlo metode daju u prosjeku točan rezultat uz nasumičnu pogrešku [10].

3.4. Uklanjanje šuma korištenjem dubokog učenja

U odjeljku 3.3 vidjeli smo kako nam najveći problem čini broj uzoraka potrebnih kako bismo dobili rezultat uz što manje prisutnog šuma. Htjeli bismo sa što manje uzoraka po svakom pikselu dobiti što kvalitetniju sliku. Upravo taj korak ćemo pokušati ublažiti koristeći duboko učenje.

Uklanjanje šuma na slikama iscrtanim slučajnim uzorkovanjem možemo formalno opisati kao preslikavanje g ulazne n-torke $\mathbf{x} = \{\mathbf{c}, \mathbf{f}\}$ u procjenu $\mathbf{d}$ referentne boje $\mathbf{r}$. Za svaki piksel p, vrijednosti ulazne n-torke dobivene su iz iscrtavatelja kao prosječna boja u RGB zapisu $\mathbf{c}_p$ i dodatne značajke $\mathbf{f}_p$, na primjer varijanca boje, normale, udaljenost od kamere i slično.

To preslikavanje možemo ostvariti koristeći duboki konvolucijski model koji će predstavljati funkciju g uz svoje parametre $\boldsymbol{\theta}$. Najbolji rezultat preslikavanja postiže se za idealne parametre $\hat{\boldsymbol{\theta}}$ koje ćemo procijeniti koristeći nadzirano učenje nad skupom podataka veličine N koji sadrži parove ulaznih n-torki (slike s prisutnim šumom) te pripadnih referentnih slika

(slike koje imaju prihvatljivu razinu šuma; da bismo dobili sliku bez šuma, morali bismo računati iznimno mnogo vremena pa se zadovoljavamo i zanemarivom količinom šuma). Skup podataka označimo kao $\mathcal{D}_N = \{(\mathbf{x}^{(1)}, \mathbf{r}^{(1)}), \ldots, (\mathbf{x}^{(N)}, \mathbf{r}^{(N)})\}$.

Sada želimo minimizirati gubitak između izlaza modela $\mathbf{d} = g(\mathbf{x}; \boldsymbol{\theta})$ i referentne slike koristeći funkciju gubitka ℓ:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \ell(g(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{r}^{(i)}). \tag{3.5}$$

Iako sliku bez šuma možemo dobiti jednostavno duljim vremenom iscrtavanja, tj. s više uzoraka po svakom pikselu, to može dugo trajati. Stoga bismo htjeli skratiti vrijeme čekanja i naučiti duboki model kako na temelju malog broja uzoraka procijeniti idealnu sliku bez šuma.

Korisnost i potrebu za ovakvim postupkom ubrzavanja nam dočarava i poznati Blinnov zakon.

3.5. Blinnov zakon

Jim Blinn definirao je (empirijski) zakon koji vrlo slikovito opisuje probleme s kojima se susreću korisnici računalne grafike, a on u prijevodu glasi ovako [10]: *"kako tehnologija napreduje, vrijeme iscrtavanja ostaje konstantno."*

U njemu je sadržano više značenja:

- u iscrtavanju smo uvijek ograničeni vremenom, bilo da se radi o vrlo kratkom vremenu, na primjer jedna slika u videoigri, bilo da imamo više vremena na raspolaganju, primjerice za animirani film,
- iako smo ograničeni vremenom, računala s vremenom postaju brža i efikasnija, ali se ta poboljšanja brzo iskorištavaju i vrijeme koje bismo uštedjeli ćemo svejedno utrošiti na dodatnu kvalitetu slike.

4. Skup podataka

Sada ćemo opisati skup podataka za učenje (engl. *dataset*). On je javno dostupan na službenoj stranici članka [1][1]. Kako se radi o slikama koje osim RGB boje imaju i niz dodatnih značajki, veličina skupa je očekivano zavidna, gotovo 300 GiB.

4.1. Iscrtavanje scena

Skup se sastoji od 1482 slika koje prikazuju jednu od 8 javno dostupnih scena. Kako je 8 scena relativno mala količina podataka za potrebe metode, umjetno su generirane dodatne scene tako da su se parametri scene (opisani niže) nasumično mijenjali iz unaprijed zadanog skupa vrijednosti. Tako dobivene scene iscrtane su (algoritmom opisanim u odjeljku 3.1) modificiranim iscrtavateljem *Tungsten* [2] (modificirana verzija je dostupna na stranici članka). Razlog modifikacije je što željene dodatne značajke koje koristimo pri učenju uobičajeno nisu dio standardnog izlaza pa ih je bilo potrebno dodati.

Kao što smo napomenuli, scene su dobivene permutacijom izvornih 8 scena [2]. Permutirane značajke su

- mapa okoline (rotiranje) i položaj sunca,
- hrapavost materijala (engl. *roughness*),
- teksture unutar pripadnog razreda (npr. pod, tapeti, kamen, drvo, ...),
- jačina svjetlosti,
- kamera.

Značajke kamere su permutirane tako da su određene točke interesa u sceni u koje kamera gleda, a zatim se kamera pomicala unutar određenog prostora. Širina pogleda (engl. *field of view*) je također mijenjana, a korištene su i dvije vrste kamere—jednostavna kamera (engl. *pinhole camera*) te kamera s tankom lećom. Permutacije scene koje nakon iscrtavanja nisu imale značaja, primjerice, bile su potpuno crne, su ručno izbačene. Iscrtane slike

[1] `http://drz.disneyresearch.com/~jnovak/publications/KPCN/`
[2] `https://github.com/tunabrain/tungsten`

pohranjene su u EXR formatu koji podržava visoki raspon vrijednosti (engl. *high dynamic range, HDR*) i više kanala u slici što je praktično za pohranu dodatnih značajki izlaza.

Razlog permutiranja scena jest postizanje što veće raznolikosti kako bi se izbjegla prenaučenost. Druga mogućnost je bila izraditi veliki broj različitih scena, međutim, to nije izvedivo iz više razloga–cijene, vremena, ljudi, itd.

Kako bismo imali parove za učenje, slike su iscrtane u više razina uzorkovanja, u potencijama broja 2, od 128 do 1024 uzorka za šumovite slike, te 8192 uzorka za referentne slike. Najviše uzorkovane slike ipak nisu savršene, ali izlaz je dovoljno konvergirao kako bismo ih mogli proglasiti referentnima. Slike s prisutnim šumom su izlaz jednog pokretanja iscrtavatelja, dok je referenca iscrtana posebno kako bi se dobio neovisan rezultat (različiti *seed* generatora nasumičnih brojeva).

4.2. Značajke izlaznih slika

Generirane slike su veličine 1280×720 piksela i pohranjene su u EXR formatu. Već smo spomenuli kako iscrtane slike osim glavnog RGB kanala (ali ne ograničenog na interval [0, 1], već visokog raspona (HDR)) sadrže i dodatne značajke koje ćemo koristiti u učenju modela. Značajke svake EXR datoteke su (pojašnjenja u nastavku)

- RGB boja (glavni kanal, 3 komponente),
- difuzni kanal (3 komponente),
- zrcalni kanal (3 komponente),
- albedo (3 komponente),
- dubina (1 komponenta),
- normale (3 komponente),
- vidljivost izvora svjetlosti (1 komponenta).

RGB boja se može podijeliti na difuzni i zrcalni kanal (zbroj). Difuzni kanal predstavlja doprinos boje iz točke prvog sjecišta zrake sa scenom, dok zrcalni kanal predstavlja doprinose zraka reflektiranih od točke prvog sjecišta. Nadalje, difuzni kanal se može podijeliti na albedo i efektivno osvjetljenje (umnožak). Albedo predstavlja lokalnu jačinu difuzne refleksije za svaku boju [11]. Dubina predstavlja udaljenost od kamere do prvog presjecišta zrake sa scenom. Normale su vektori okomiti na površinu u točki najbližeg sjecišta zrake iz kamere. Na prvom presjecištu također uzorkujemo izvore svjetlosti. Ako je uzorkovani izvor izravno vidljiv, dodajemo vrijednost 1, a u suprotnom 0. Značajka vidljivosti izvora svjetlosti je prosjek svih takvih uzorkovanja [11]. Svaki navedeni kanal ima još pridruženu

i svoju varijancu (1 komponenta). Ukratko, svaka slika sadrži ukupno 14 kanala (7 navedenih plus kanali s varijancama). Izlaz iscrtavatelja zapravo sadrži i više kanala, ali oni nisu navedeni jer ih nismo koristili.

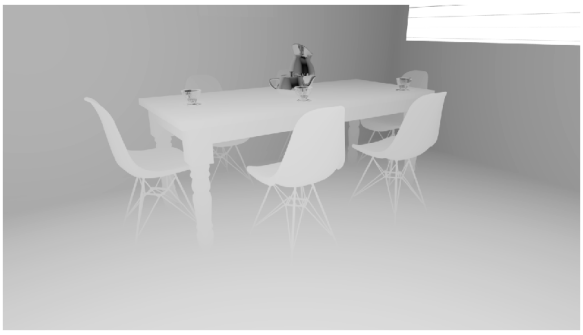
Slika 4.1: Glavni kanal (RGB) uz prisutan šum.

Slika 4.2: Glavni kanal (RGB) bez šuma.

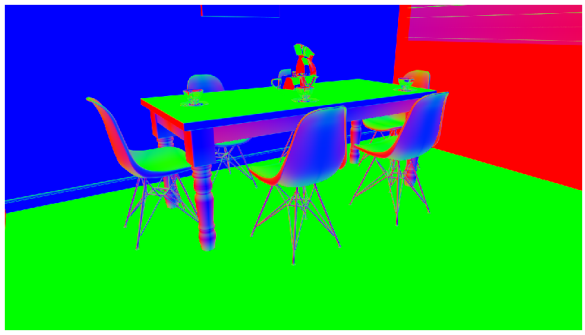
(a) Difuzna komponenta boje.

(b) Zrcalna komponenta boje.

(c) Albedo.



(d) Dubina.



(e) Normale.

(f) Varijanca normala.

Slika 4.3: Dodatne značajke.

4.3. Pretprocesiranje

Prije nego krenemo na generiranje parova ulaza i referentnog izlaza, potrebno je obaviti određene transformacije nad dobivenim slikama.

Difuznu komponentu transformiramo kao

$$\tilde{\mathbf{c}}_{\text{diffuse}} = \mathbf{c}_{\text{diffuse}} \oslash (\mathbf{f}_{\text{albedo}} + \epsilon),$$

pri čemu je $\oslash$ Hadamardov operator dijeljenja (svaki element lijevog tenzora dijelimo odgovarajućim elementom desnog tenzora iste veličine), a $\epsilon = 0.00316$ (konstanta iz članka [1]).

Zrcalnu komponentu ćemo transformirati izrazom

$$\tilde{\mathbf{c}}_{\text{specular}} = \log(1 + \mathbf{c}_{\text{specular}}).$$

Nakon transformiranja ovih dviju komponenti potrebno je transformirati i njihove varijance sljedećim izrazima:

$$(\tilde{\sigma}_{\text{diffuse}})^2 \approx \sigma^2_{\text{diffuse}} \oslash (\mathbf{f}_{\text{albedo}} + \epsilon)^2,$$

$$(\tilde{\sigma}_{\text{specular}})^2 \approx \sigma^2_{\text{specular}} \oslash (\tilde{\mathbf{c}}_{\text{specular}})^2.$$

Razlog ovih transformacija leži u vrijednostima odgovarajućih komponenti. Difuzna komponenta inače ima "pitome" vrijednosti, ali se u praksi izdvaja albedo kako bi se radilo s efektivnom osvijetljenošću (engl. *effective irradiance*). Ona je glatkija od čiste difuzne komponente i omogućava nam korištenje jezgri većih dimenzija.

Zrcalna komponenta nam stvara malo više problema. Naime, ona može imati veliki raspon vrijednosti (i nekoliko redova veličine) i stoga je teško raditi s čistim izlazom te komponente. Zato uvodimo logaritamsku transformaciju kako bismo dobili manji raspon vrijednosti i osigurali stabilniju optimizaciju.

Kako bismo iz transformiranih značajki dobili izvorne, potrebno je primijeniti inverzne transformacije. To je prikazano relacijom

$$\hat{\mathbf{c}} = (\mathbf{f}_{\text{albedo}} + \epsilon) \odot \hat{\mathbf{c}}_{\text{diffuse}} + \exp(\hat{\mathbf{c}}_{\text{specular}}) - 1,$$

pri čemu su $\hat{\mathbf{c}}_{\text{diffuse}}$ i $\hat{\mathbf{c}}_{\text{specular}}$ procjene vrijednosti odgovarajućih komponenti.

Dodatno, još ćemo izračunati gradijente odgovarajućih kanala u x i y smjerovima, što će činiti dio ulaza u mrežu. Gradijente računamo tako da od određenog piksela oduzmemo vrijednosti lijevog ili gornjeg susjeda po svim komponentama. Rubove ćemo popuniti nulama. Funkcije koje računaju gradijente označavamo s G_x i G_y.

4.4. Generiranje ulaza neuronske mreže

Nakon što smo obavili pretprocesiranje, potrebno je generirati sličice (engl. *patches*) veličine 65×65 piksela koje ćemo koristiti kao ulaz u neuronske mreže.

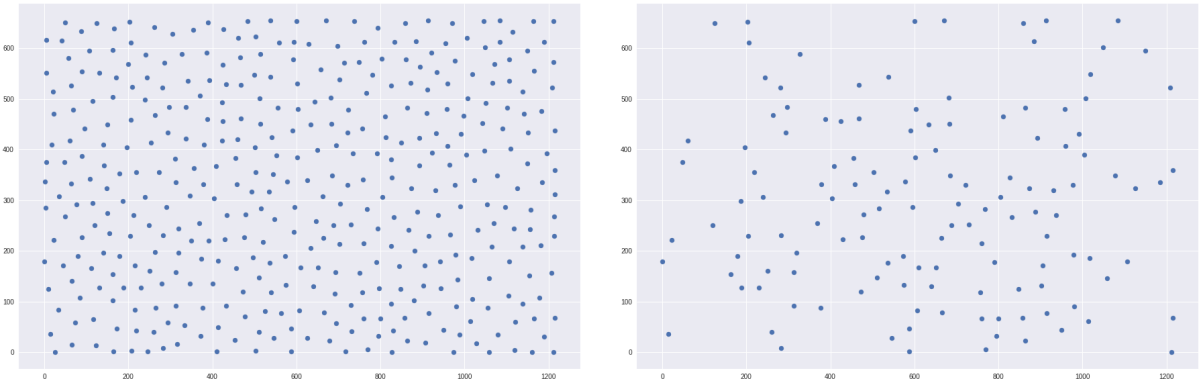
Jedna mogućnost je pozicije sličica odrediti uniformno nasumično, ali to će nam dati puno "laganih" primjera s kojih nije teško ukloniti šum (npr. glatka površina). Stoga ćemo koristiti metodu uzorkovanja pomoću mape važnosti (engl. *importance map*) koja određuje vjerojatnost odabira sličice s pojedinog područja slike. Ideja je dati veću važnost područjima koja smatramo težima za postupak uklanjanja šuma kao što su rubovi, nagli prijelazi, velike varijance normala i boje, itd. Mapu važnosti računamo pomoću RGB kanala te kanala s

normalama, i to na temelju njihovih varijanci. Konačni rezultat dobivamo zbrojem njihovih varijanci (uzimajući u obzir maksimum od 3 komponente pojedinog kanala) normaliziranim na raspon između 0 i 1.

Slika 4.4: Mapa važnosti za sliku 4.1.

Na slici 4.4 možemo vidjeti primjer mape važnosti. Tamnija područja označavaju veću važnost, odnosno mjesta gdje očekujemo da će biti teže otkloniti šum i želimo više primjera s takvih područja. Drugim riječima, primjeri s većom važnosti će imati veću šansu da budu odabrani, ali će biti odabrani i primjeri s manjom važnosti.

Na slikama 4.5a i 4.5b možemo vidjeti utjecaj mape važnosti. Slika 4.5a prikazuje početni uniformni odabir pozicija sličica na velikoj slici. Nakon toga primjenjujemo mapu



(a) Uniformno odabrane pozicije.

(b) Pozicije nakon odbacivanja.

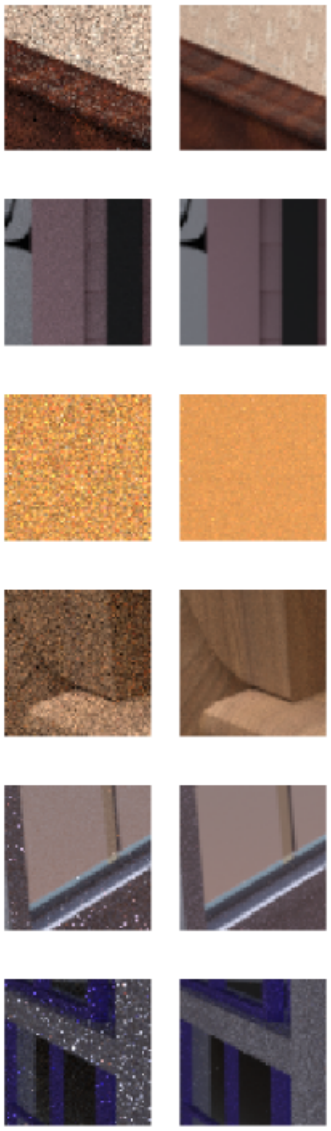
Slika 4.5: Odabir pozicija prema mapi važnosti.

važnosti i odbacujemo (engl. *pruning*) primjere za koje smatramo da neće suviše doprinijeti učenju. Slika 4.5b prikazuje pozicije nakon odbacivanja s obzirom na važnost. Vidimo kako je na rubovima slike odabrano manje primjer nego iz središta. Ako to usporedimo sa slikom 4.4, tamo je uistinu manja važnost na rubovima, a većina tamnijih područja je koncentrirana oko središta slike. Također smo zadovoljili i želju da odaberemo i primjere s malom važnosti.

Nakon što smo obavili pretprocesiranje, možemo "sastaviti" ulaz u model koji ćemo prikazati kao:

$$\mathbf{x} = \{\tilde{\mathbf{c}}, G_x(\{\tilde{\mathbf{c}}, \mathbf{f}\}), G_y(\{\tilde{\mathbf{c}}, \mathbf{f}\}), \tilde{\sigma}^2, \tilde{\sigma}^2_{\mathbf{f}}\},$$

pri čemu su G_x i G_y gradijenti odgovarajućih značajki u x i y smjeru, $\tilde{\mathbf{c}}$ i $\tilde{\sigma}^2$ mogu pripadati difuznoj ili zrcalnoj komponenti, a $\mathbf{f}$ sadrži dodatne značajke (normale, dubina, itd.).



Slika 4.6: Primjeri parova odabranih sličica iz skupa za učenje (lijevo ulaz, desno referenca).

5. Implementacija

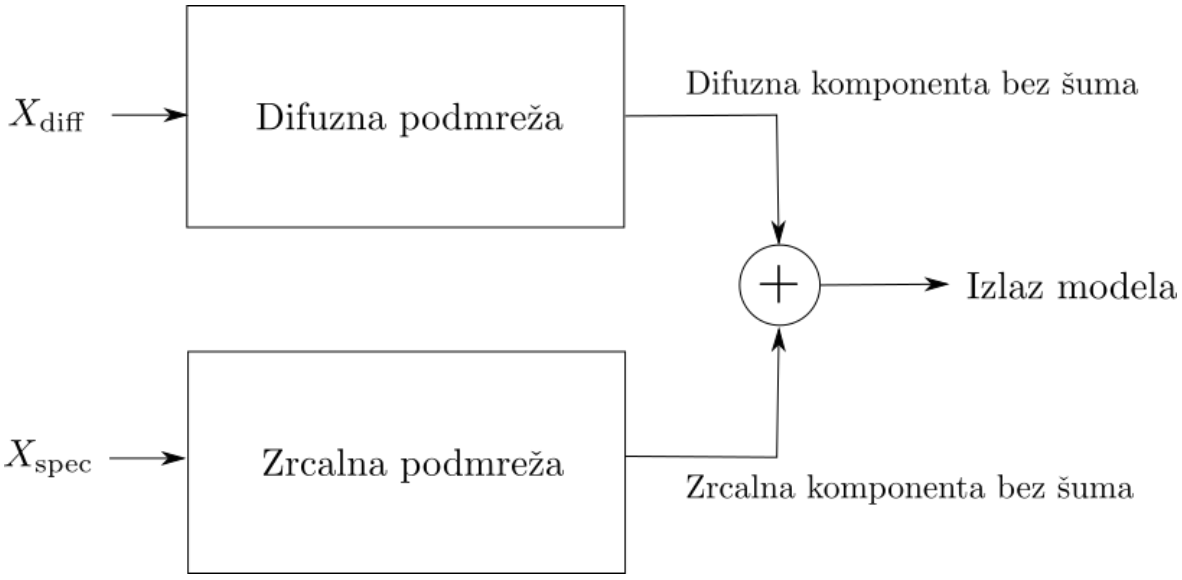
Sada ćemo opisati korištene modele. Radi bolje usporedivosti, svi modeli su imali iste ulazne podatke. Oni se sastoje od 28 kanala širine i visine 65 piksela. Kanali dolaze od pretprocesiranih značajki opisanih u poglavlju 4, a sadrže i njihove varijance te gradijente. Radi boljih rezultata, ulazi su podijeljeni na difuzne (X_{diff}) i zrcalne (X_{spec}), a jedina razlika je u kanalu boje. Dva osnovna modela ćemo zvati model s izravnim predviđanjem te model s predviđanjem pomoću jezgre. Oba modela su konvolucijska, a sastoje se od dvije podmreže (zadužene za difuznu i zrcalnu komponentu) s dijeljenim parametrima. Nakon njih ćemo opisati model koji je također konvolucijski, ali se sastoji od rezidualnih blokova. On također koristi predviđanje pomoću jezgre. Isprobat ćemo ga u varijanti s dijeljenim te zasebnim parametrima podmreža.

5.1. Model s izravnim predviđanjem

Najjednostavniji model koji kao izlaz daje tri komponente boje pojedinih piksela slike (crvena, zelena i plava boja), ali s uklonjenim šumom. Sastoji se od dvije konvolucijske podmreže–difuzne i zrcalne. Razlog takve podjele su različita svojstva tih dvaju komponenti. Zrcalna komponenta ima veći raspon vrijednosti od difuzne pa se pokazalo korisno razdvojiti postupak uklanjanja šuma za svaku komponentu.

Ovaj model zvat ćemo DPCN (kratica od engleskog naziva *direct prediction convolutional network*. Iako postiže dobre rezultate, učenje ovog modela je sporije od ostalih te ima problema s pomakom u boji–izlaz ponekad uz uklonjeni šum ima drugačiju boju, na primjer svjetliju ili tamniju. Razlog sporog učenja je veći prostor pretraživanja. On gotovo da i nema ograničenja u izlazu pa mu je potrebno duže vremena kako bismo dobili zadovoljavajuće rezultate.

Dijagram ovog modela možemo vidjeti na slici 5.1.



Slika 5.1: Dijagram modela s izravnim predviđanjem.

Dimenzije tenzora ovog modela su sljedeće (B je veličina mini grupe, H visina, a W širina slike uz konvenciju $BCHW$):

- ulaz: $B \times 28 \times H \times W$,
- skriveni slojevi: $B \times 100 \times H \times W$,
- izlaz: $B \times 3 \times H \times W$.

Broj parametara iznosi

$$28 \times 100 \times 5 \times 5 + 100 + (L-2) \times (100 \times 100 \times 5 \times 5 + 100) + 100 \times 3 \times 5 \times 5 + 3,$$

što za $L = 9$ i $k = 21$ iznosi 1828303 (puta dva zbog dvije podmreže).

5.2. Model s predviđanjem pomoću jezgre

Drugi korišteni model je također konvolucijski, ali s predviđanjem pomoću jezgre (engl. *kernel prediction convolutional network, KPCN*) (slika 5.2). On kao izlaz podmreža daje jezgru (engl. *kernel*) za svaki piksel čijom primjenom nad susjedstvom uklanjamo šum sa središnjeg piksela. Vrijednost svake RGB komponente svakog piksela susjedstva potrebno je pomnožiti s odgovarajućom vrijednosti jezgre. Sve tri komponente množimo istom vrijednosti. Time sprječavamo pomak u boji koji se javlja kod prethodnog modela. Jezgru dobivamo tako da zadnji sloj kao izlaz da onoliko mapi značajki koliko težina jezgra sadrži.

Dimenzije tenzora su sljedeće (B je veličina mini grupe, H visina, a W širina slike uz konvenciju $BCHW$):

- ulaz: $B \times 28 \times H \times W$,

– skriveni slojevi: $B \times 100 \times H \times W$,

– izlaz: $B \times k^2 \times H \times W$.

Korištena je dimenzija jezgre $k = 21$. Tako ćemo na izlazu imati $21^2 = 441$ mapu značajki. Dakle, svaki piksel ima jezgru koju će primijeniti na sebe i svoje susjede, a ista jezgra se primjenjuje na sve tri RGB komponente. Broj parametara iznosi

$$28 \times 100 \times 5 \times 5 + 100 + (L-2) \times (100 \times 100 \times 5 \times 5 + 100) + 100 \times k^2 \times 5 \times 5 + k^2,$$

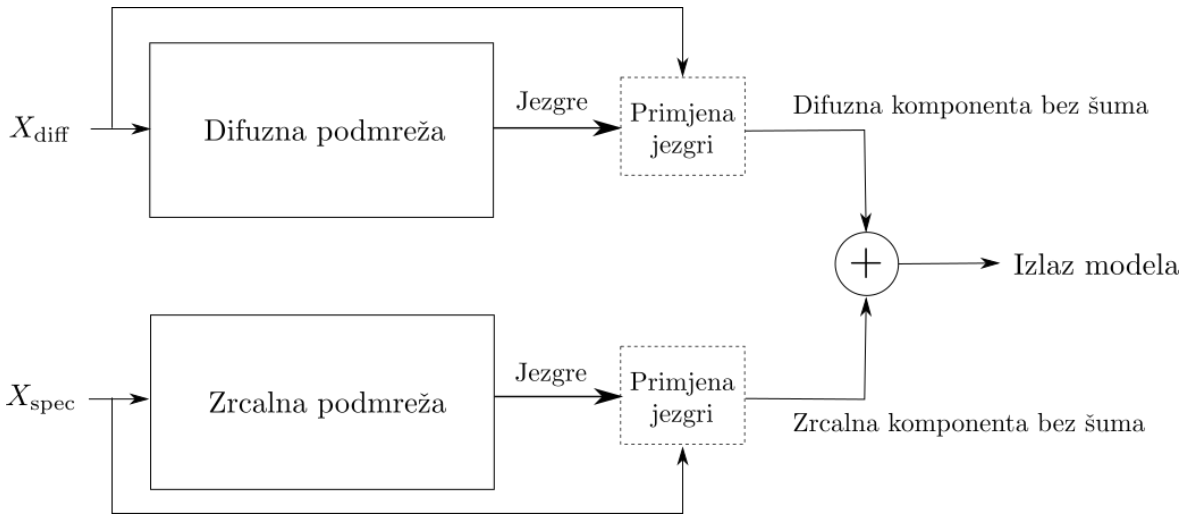
što za $L = 9$ i $k = 21$ iznosi 2923741 (puta dva zbog dvije podmreže). To je otprilike 1.6 puta više parametara od prethodne varijante.

Dodatno, izlaz se prvo treba "provući" kroz aktivacijsku funkciju $\mathrm{softmax}$ (jednadžba 2.5) kako bismo dobili jezgre čije su težine u rasponu $[0, 1]$ i čija je suma jednaka 1. Razlozi su višestruki. Kako ova varijanta vuče ideje iz "klasičnih" *denoisera*, tako pokušava zadržati dobre aspekte, a popraviti loše (to je uglavnom odabir jezgre). Ono što postižemo funkcijom $\mathrm{softmax}$ je sljedeće [1]:

– osigurava da procjena leži u konveksnoj ljusci susjedstva ulazne slike i time znatno smanjuje prostor pretraživanja u odnosu na izravnu predikciju,

– osigurava bolje ponašanje gradijenata s obzirom na težine jezgre, tj. treba samo naučiti relativnu važnost pojedinog piksela u susjedstvu, a ne mora znati i skalu,

– omogućava primjenu na slojeve slike, tako da koristimo jednake težine na svakoj komponenti i time osiguravamo da ne dođe do "pomaka u boji" i zadržavamo prosječni intenzitet boje [11].



Slika 5.2: Dijagram modela s predviđanjem pomoću jezgre.

Usporedbu ova dva modela možemo vidjeti u tablici 5.1.

Tablica 5.1: Usporedba modela DPCN i KPCN.

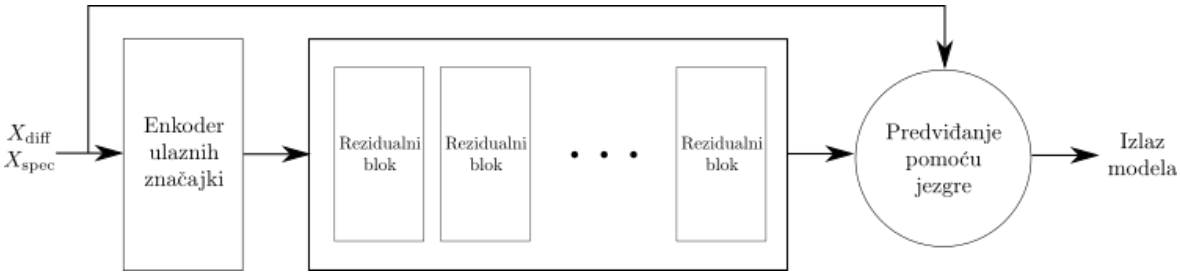
	DPCN	KPCN
Broj parametara	2×1828303	2×2923741
Broj slojeva	9	9
Veličina konvolucijske jezgre	5×5	5×5
Veličina izlaza	$B \times 3 \times H \times W$	$B \times k^2 \times H \times W$
Aktivacijska funkcija na izlazu	-	$\mathrm{softmax}$

5.3. Rezidualni model

Kao treći model korišten je rezidualni duboki model. Zadržat ćemo predviđanje pomoću jezgre jer se pokazalo da daje dobre rezultate, a pokušat ćemo koristiti dijeljene parametre, odnosno s jednom "podmrežom" ukloniti šum na obje komponente boje. Pretpostavljamo da nam je zato potreban veći kapacitet modela što znači da trebamo više slojeva. Kako su se rezidualni modeli pokazali kao efikasno rješenje problema koji nastaju kod dubokih modela s mnogo slojeva, iskoristit ćemo tu ideju. Dijagram možemo vidjeti na slici 5.3.

Možemo ga podijeliti na tri glavna dijela, a to su

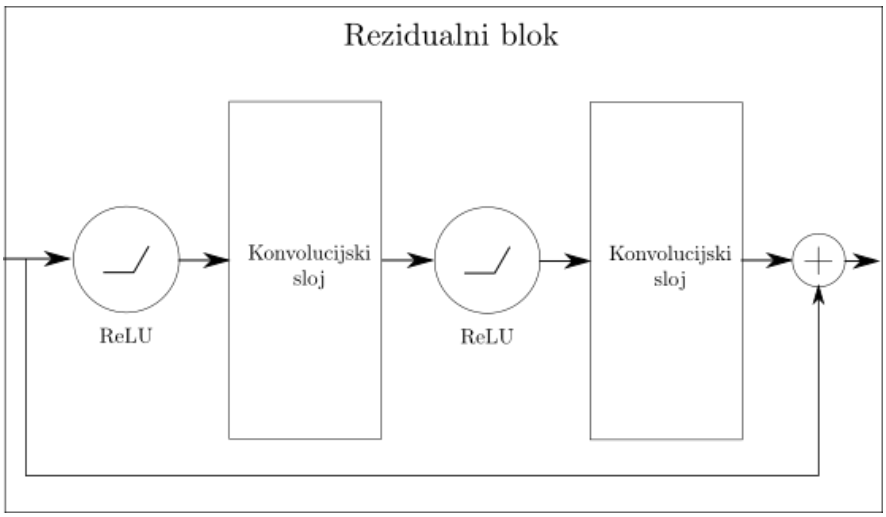
1. enkoder ulaznih značajki,
2. niz rezidualnih blokova,
3. modul za predviđanje pomoću jezgre.



Slika 5.3: Dijagram rezidualnog modela s predviđanjem pomoću jezgre.

Enkoder ulaznih značajki nam omogućava modularnost kako bismo mogli lakše mijenjati broj mapi značajki u ostatku modela. Članak [12] navodi i kako može poslužiti za bolju prilagodbu različitim iscrtavateljima zbog manjih razlika u njihovim izlazima. Time se može učiti samo enkoder, a ostatak mreže može ostati konstantan i tako se prilagoditi novim podacima bez dugotrajnog učenja ispočetka. Sastoji se od dva konvolucijska sloja s jezgrama dimenzija 3×3 između kojih se nalazi aktivacijska funkcija ReLU.

Zatim slijedi niz rezidualnih blokova čiji dijagram možemo vidjeti na slici 5.4. Oni su glavni dio modela, a sastoje se od dva konvolucijska sloja s jezgrama dimenzija 3×3. Prije ulaza u svaki sloj nalazi se aktivacijska funkcija ReLU.



Slika 5.4: Dijagram rezidualnog bloka.

Naposljetku, imamo modul za predviđanje pomoću jezgre koji se, slično kao i ulazni enkoder sastoji od dva konvolucijska sloja povezana funkcijom ReLU, ali kod njega su konvolucijske jezgre dimenzije 1×1. Izlaz modula je k^2 mapi značajki gdje je k veličina jezgre kojom uklanjamo šum. U našem slučaju ona iznosi $k = 21$ što znači da ćemo imati $21^2 = 441$ mapu značajki na izlazu.

Važno je napomenuti kako sada više nemamo podmreže s odvojenim parametrima, već ćemo difuznu i zrcalnu komponentu tretirati istim modelom s istim težinama. To također dovodi do manjeg ukupnog broja parametara. Korišteni model ima 972217 parametara što je znatno manje od druga dva modela. Za usporedbu ćemo isprobati i model s posebnim parametrima za svaku komponentu boje.

6. Rezultati

U ovom poglavlju ćemo prezentirati i usporediti rezultate korištenih modela. Svi modeli su trenirani algoritmom Adam uz korak učenja 10^{-4}. Veličina mini grupe bila je 64. Skup podataka sastoji se od 16867 primjera za učenje uz podjelu 80–20 na skup za učenje i skup za validaciju. Kao gubitak korišten je L1. Primjeri za učenje su kvadratne sličice dimenzije 65 piksela. Svi eksperimenti provedeni su na Google Compute Engine instanci koja kao GPU koristi Nvidia K80 s 12 GB memorije.

Modeli s izravnim i predviđanjem pomoću jezgre sastoje se od 9 konvolucijskih slojeva, svaki s po 100 jezgri dimenzije 5×5. Rezidualni model se sastoji od 12 rezidualnih blokova unutar kojega se nalaze dva konvolucijska sloja. Svi skriveni konvolucijski slojevi modela sadrže 64 konvolucijske jezgre. Sve konvolucijske jezgre su dimenzije 3×3, osim u modulu za predviđanje jezgrom gdje su veličine 1×1 (opisano u odjeljku 5.3).

Na slikama 6.1, 6.2 i 6.3 redom su prikazane pogreške na skupovima za učenje i validaciju za model s izravnim predviđanjem, model s predviđanjem pomoću jezgre te rezidualna inačica s predviđanjem pomoću jezgre. U gornjim redovima prikazane su akumulirane pogreške tijekom jedne epohe, dok su u donjim redovima prikazane pogreške pojedine mini grupe. Pogreška na skupu za validaciju je manja od pogreške na skupu za učenje zbog manjeg broja primjera, a grafovi prikazuju akumuliranu pogrešku cijelog skupa za učenje, odnosno validaciju.

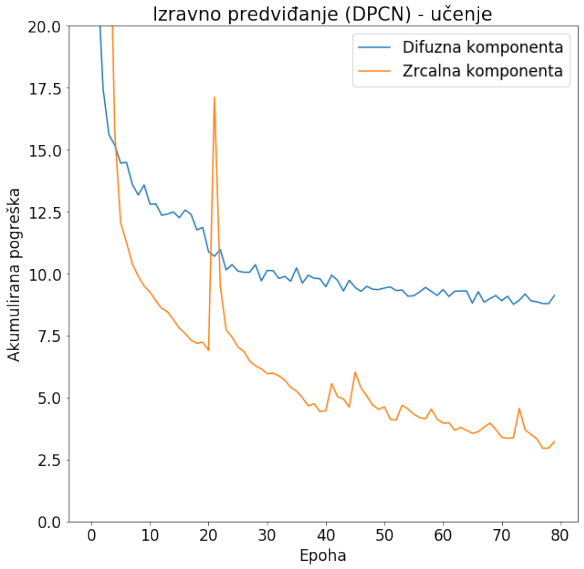
Prikazane su pogreške difuzne i zrcalne komponente te pogreške na završnoj boji. Pogreška boje izravno ovisi o pogreškama dvaju pripadnih komponenti. Pogreške pojedine komponente prikazane su kako bi se vidjela pogreška odgovorne podmreže.

Model s izravnim predviđanjem treniran je 80 epoha, odnosno duplo više od ostalih modela što je posljedica njegovog sporijeg učenja zbog većeg prostora pretraživanja. Uz dovoljno dugo učenje, trebao bi moći dostići druga dva modela.

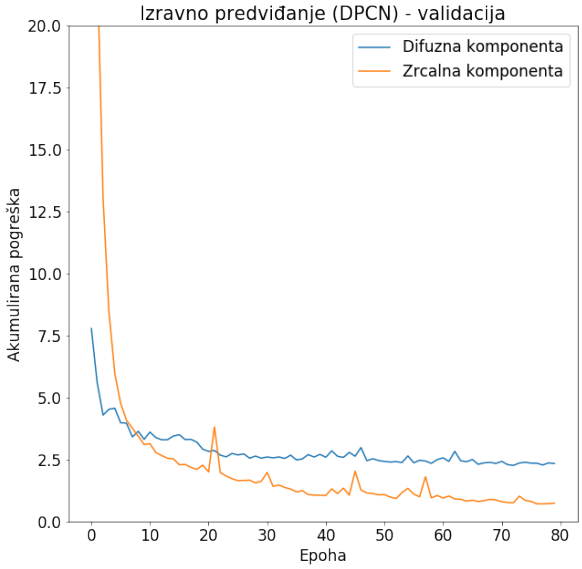
Rezidualni model postigao je slabije rezultate za komponentu boje od modela s predviđanjem pomoću jezgre, ali možemo reći kako smo uspjeli u namjeri da dvije podmreže sa zasebnim parametrima zamijenimo jednim modelom, odnosno dijeljenim parametrima. Time smo smanjili broj parametara modela. Pristup predviđanja jezgrom se pokazao boljim u odnosu na izravno predviđanje. S obzirom na rezultate, najbolje se pokazao model KPCN

(odjeljak 5.2). Rezidualni model (odjeljak 5.3) je postigao malo lošije rezultate, ali s obzirom na to da koristi rezidualne blokove i modularan je, ima veći potencijal za daljnji razvoj.

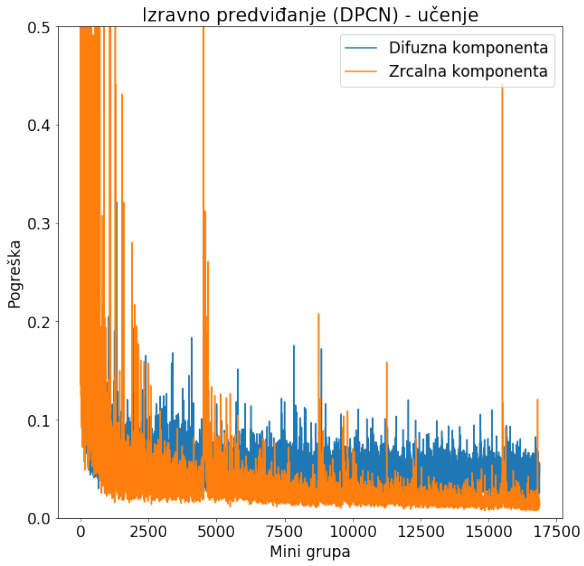
Na slikama je prikazan rezidualnih model s dijeljenim parametrima ako nije drugačije naznačeno.



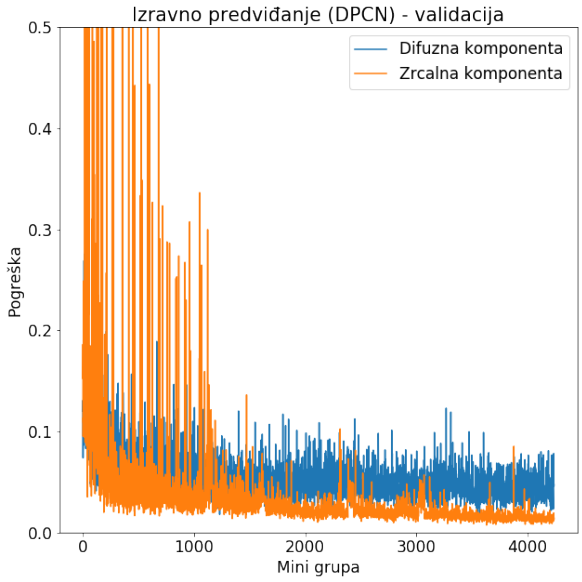
(a) Pogreška na skupu za učenje.



(b) Pogreška na skupu za validaciju.

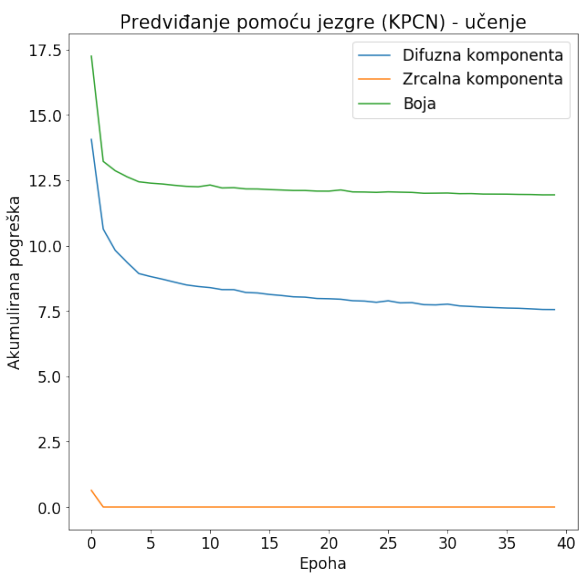


(c) Pogreška učenja po mini grupi.

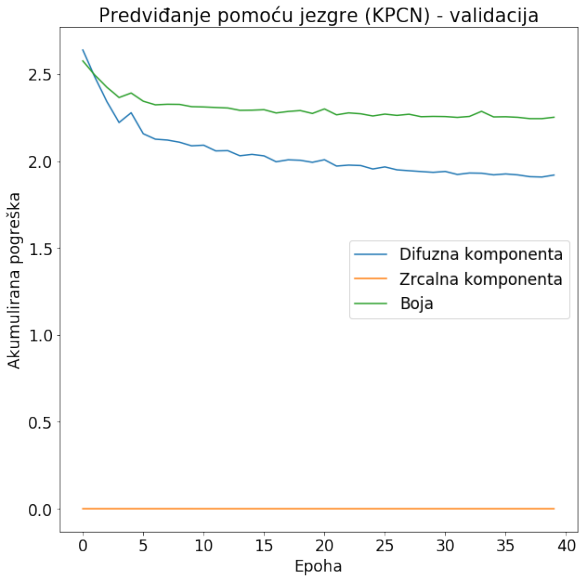


(d) Pogreška validacije po mini grupi.

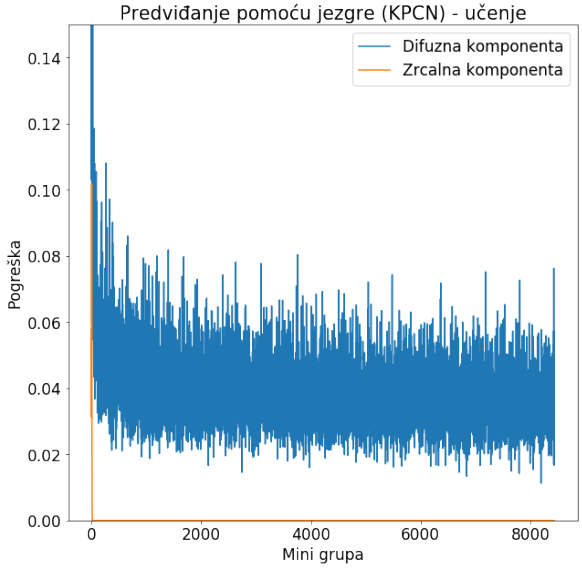
Slika 6.1: Prikaz pogreške učenja modela s izravnim predviđanjem. Korišten je gubitak L1. Gubitak boje nije prikazan zbog prevelike razlike u skali.



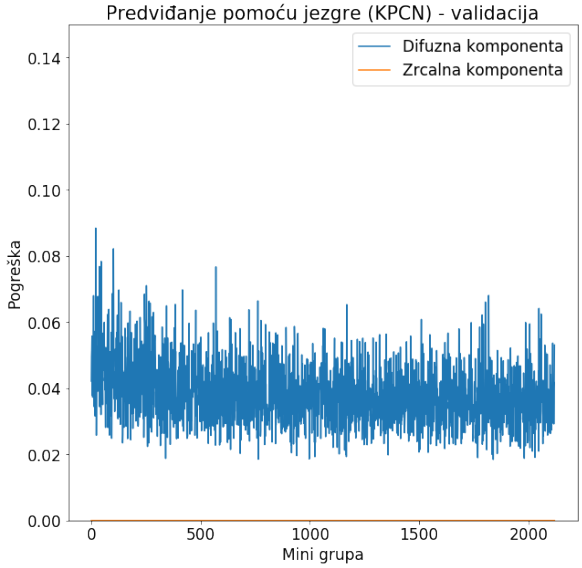
(a) Pogreška na skupu za učenje.



(b) Pogreška na skupu za validaciju.

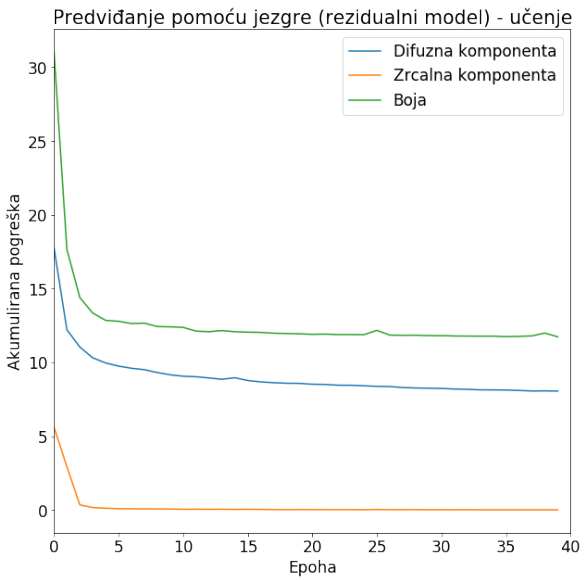


(c) Pogreška učenja po mini grupi.

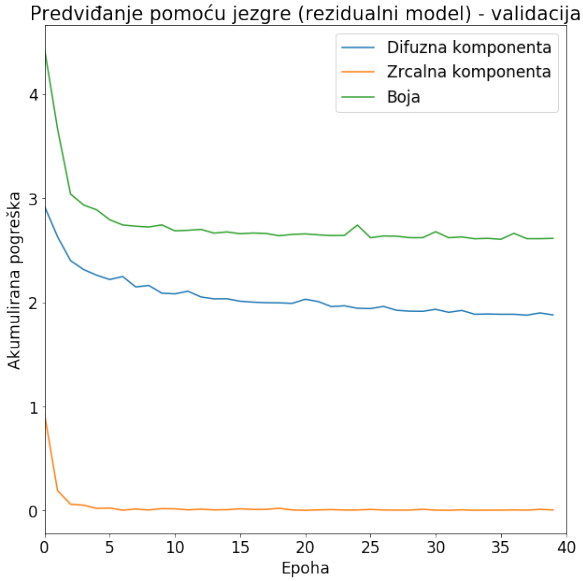


(d) Pogreška validacije po mini grupi.

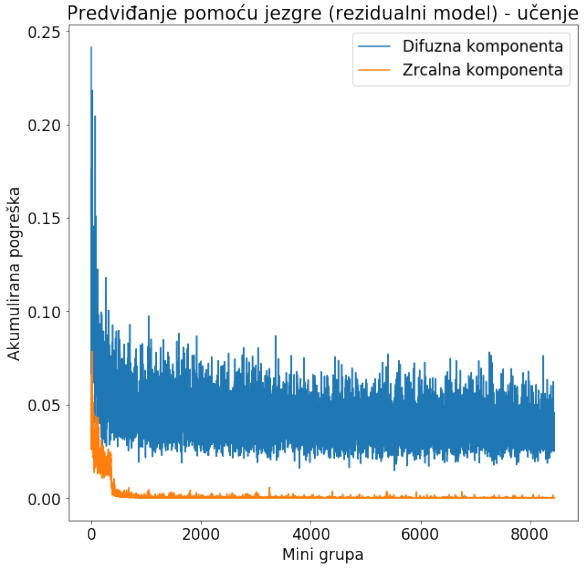
Slika 6.2: Prikaz pogreške učenja modela s predviđanjem pomoću jezgre. Korišten je gubitak L1.



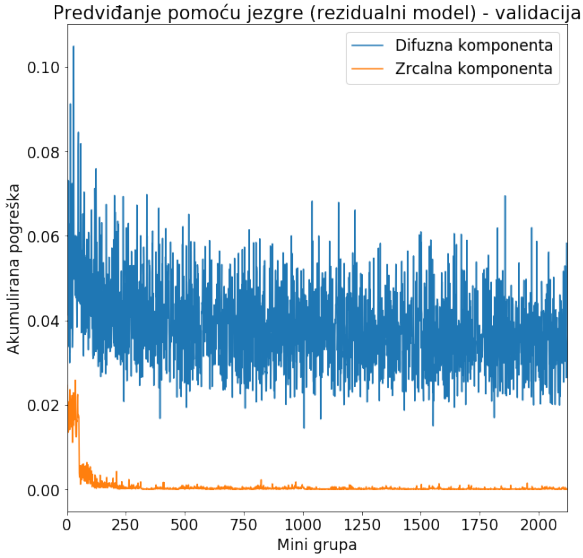
(a) Pogreška na skupu za učenje.



(b) Pogreška na skupu za validaciju.



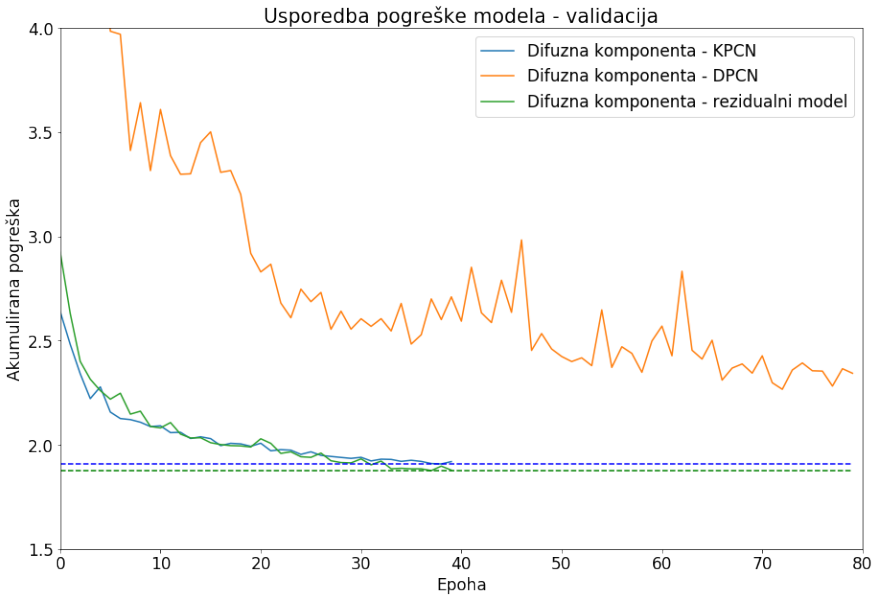
(c) Pogreška učenja po mini grupi.



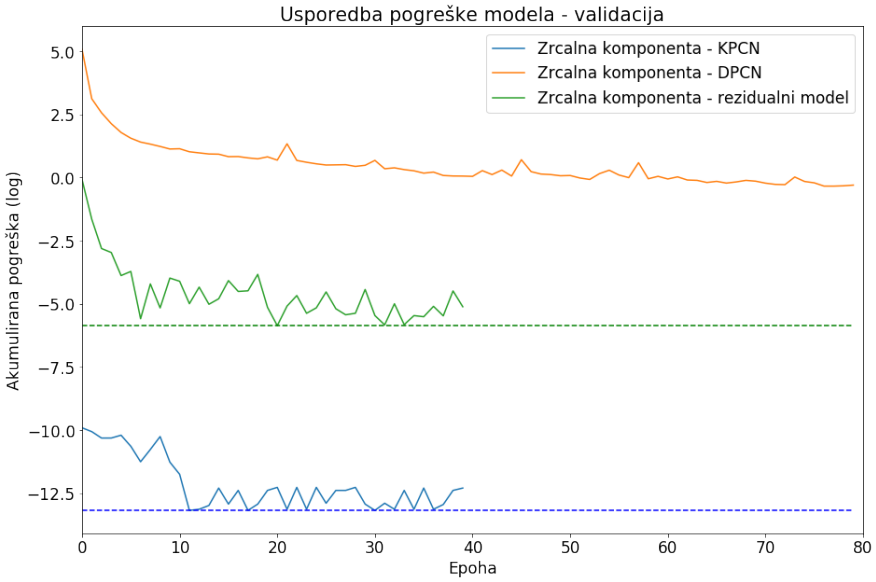
(d) Pogreška validacije po mini grupi.

Slika 6.3: Prikaz pogreške učenja rezidualnog modela s predviđanjem pomoću jezgre. Korišten je gubitak L1.

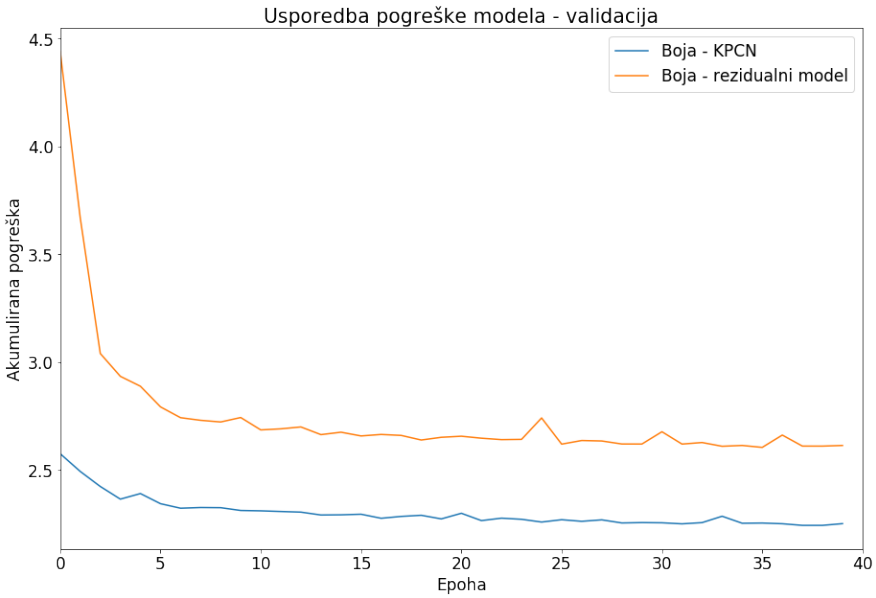
Slike 6.4, 6.5 i 6.6 prikazuju usporedbe pojedine komponente na svakom modelu. Na slici 6.4 možemo primijetiti kako je difuzna komponenta podjednako dobro obrađena i kod KPCN-a, i kod rezidualnog modela, dok DPCN zaostaje, ali uz duže treniranje bi mogao postići podjednaku vrijednost gubitka. Na slikama 6.5 i 6.6 je KPCN uvjerljivo najbolji, a kod pogreške na završnoj boji gubitak DPCN-a nije prikazan zbog prevelike razlike u odnosu na druga dva modela.



Slika 6.4: Usporedba pogreške na difuznoj komponenti boje.

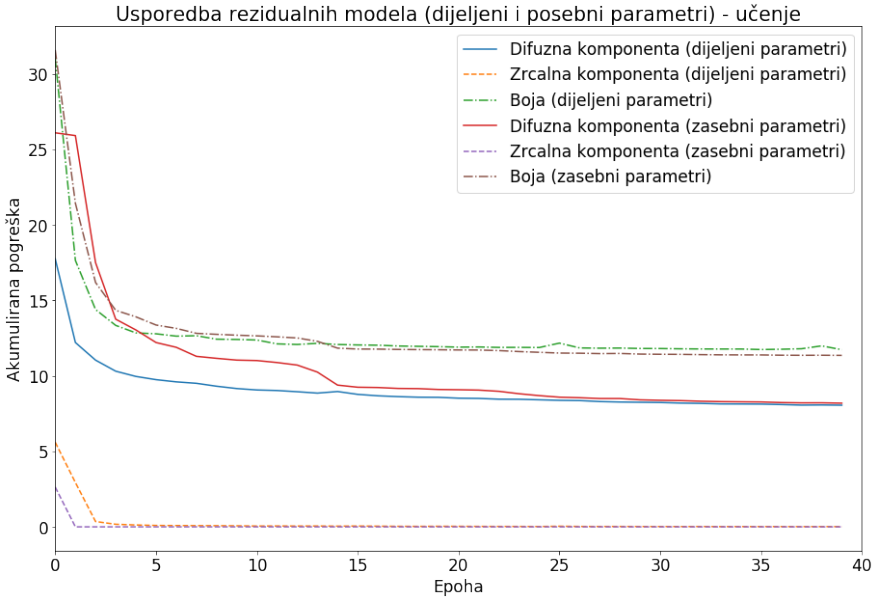


Slika 6.5: Usporedba pogreške na zrcalnoj komponenti boje. Pogreška je u logaritamskoj skali zbog velike razlike kod DPCN-a.

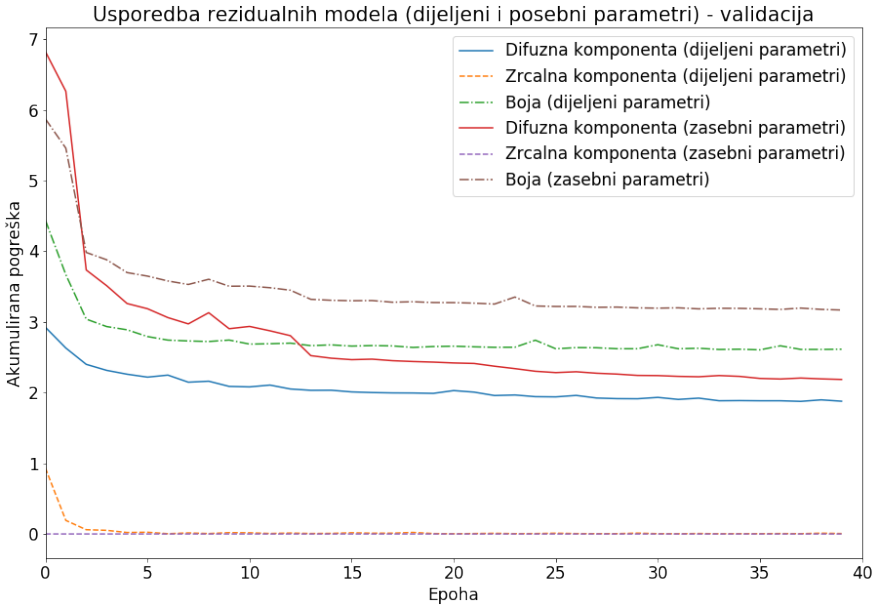


Slika 6.6: Usporedba pogreške završne boje. Pogreška modela DPCN je izostavljena zbog prevelike razlike u skali.

Za usporedbu su isprobani rezidualni model s dijeljenim te zasebnim parametrima. Rezultati na skupu za učenje prikazani su na slici 6.7, a na skupu za validaciju na slici 6.8. Vidimo kako su obje varijante na skupu za učenje postigle gotovo jednake rezultate uz opasku da je model sa zasebnim parametrima malo sporije učio u početku. No, na validacijskom skupu model s posebnim parametrima zaostaje za modelom s dijeljenim parametrima. Mogući uzrok je da korištenje istih parametara za obje komponente boje ima regularizacijski učinak.

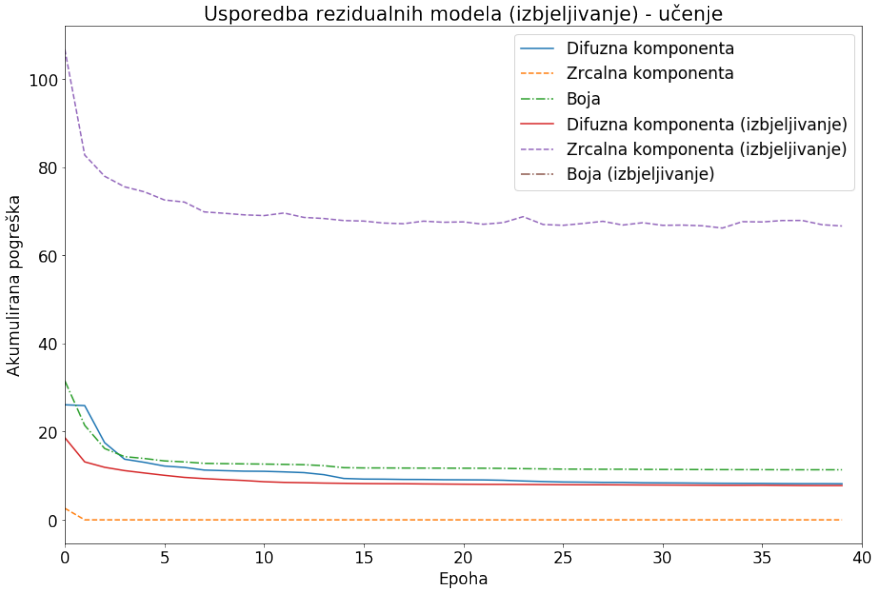


Slika 6.7: Usporedba pogreške rezidualnih modela s dijeljenim i zasebnim parametrima na skupu za učenje.

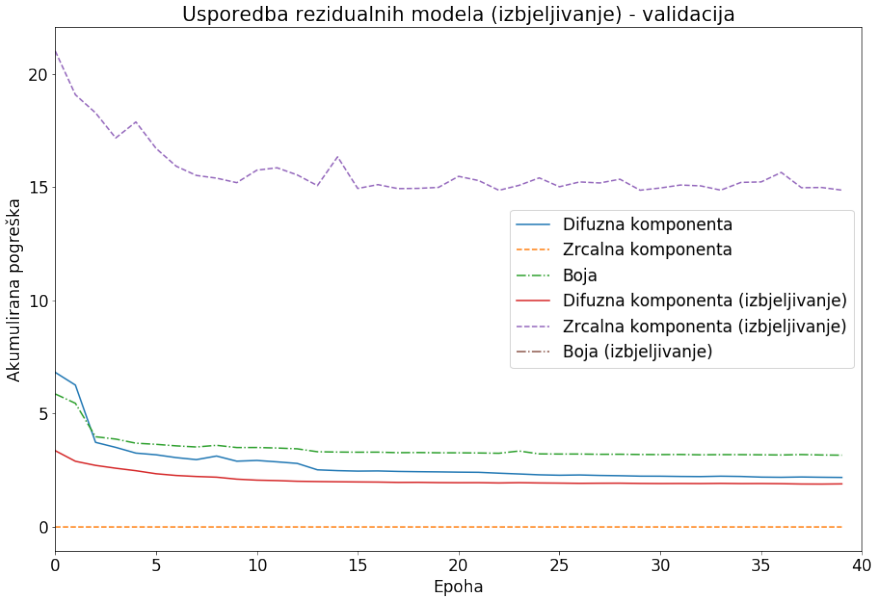


Slika 6.8: Usporedba pogreške rezidualnih modela s dijeljenim i zasebnim parametrima na validacijskom skupu.

Isprobano je i izbjeljivanje (engl. *whitening*) ulaznih značajki na rezidualnom modelu s odvojenim parametrima. Izbjeljivanje provodimo tako da od svakog ulaznog kanala oduzmemo njegovu srednju vrijednost te rezultat podijelimo standardnom devijacijom pripadnog kanala. Rezultati su prikazani na slikama 6.9–6.10. Iako smo očekivali poboljšanje rezultata, eksperiment nije uspješan. Gubitak na difuznoj komponenti podjednak je kao i bez izbjeljivanja, dok je gubitak na zrcalnoj komponenti značajno veći nego prije.



Slika 6.9: Usporedba utjecaja izbjeljivanja ulaznih kanala (skup za učenje).



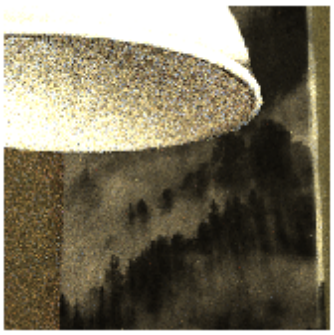
Slika 6.10: Usporedba utjecaja izbjeljivanja ulaznih kanala (skup za validaciju).

U tablici 6.1 prikazano je trajanje učenja te trajanje uklanjanja šuma na jednoj slici dimenzija 1280×720 za pojedini model. Možemo primijetiti kako korištenje jezgri za uklanjanje šuma dovodi do značajnog povećanja utrošenog vremena, ali razlog tome može biti neefikasna implementacija primjene jezgri što ostavlja još prostora za napredak ove metode.

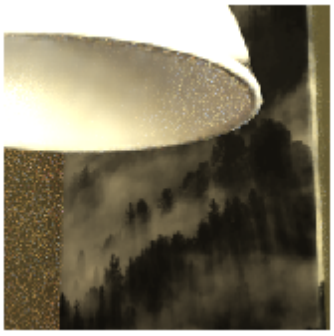
Tablica 6.1: Usporedba trajanja učenja i uklanjanja šuma.

	DPCN	KPCN	Rezidualni model
Trajanje učenja (40 epoha) [h]	4.0	6.8	7.0
Trajanje uklanjanja šuma (1280×720 piksela) [s]	10	140	140

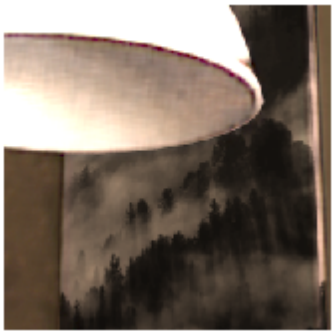
Pogledajmo još i problem s kojim se susreće samo model s izravnim predviđanjem, a to je pomak u boji. Predviđanje jezgrom je taj problem eliminiralo jer se sve tri RGB komponente množe istim faktorom i time se sprječava svjetlija ili tamnija boja od očekivane. DPCN nema ograničenja u izlazu i potrebno mu je dulje vrijeme da nauči uz uklanjanje šuma, uskladiti i intenzitet boje.



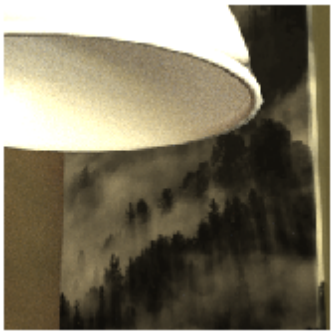
(a) Slika s prisutnim šumom.



(b) Izlaz iz KPCN-a.



(c) Izlaz iz DPCN-a.



(d) Referentna slika.

Slika 6.11: Prikaz problema pomaka u boji kod DPCN-a. Preuzeto iz [5].

Na slikama 6.12–6.13 nalaze se primjeri uklanjanja šuma modela s predviđanjem pomoću jezgre te rezidualnog modela.

(a) Slika s prisutnim šumom.

(b) Izlaz iz KPCN-a.

(c) Referentna slika.

Slika 6.12: Primjer uklanjanja šuma modelom s predviđanjem pomoću jezgre.

(a) Slika s prisutnim šumom.

(b) Izlaz iz rezidualnog modela.

(c) Referentna slika.

Slika 6.13: Primjer uklanjanja šuma rezidualnim modelom s predviđanjem pomoću jezgre.

7. Zaključak

Napredak računalne grafike omogućio je razvoj vrlo vjernih prikaza i vizualizacija, od video igara, preko filmova pa sve do znanosti. Istovremeno, napretkom strojnog, točnije dubokog učenja otvorila su se nova vrata za dodatna poboljšanja. Kako je vrijeme jedno od ključnih resursa, nikada nije na odmet ubrzati postojeće postupke.

Uklanjanje šuma pomoću dubokog učenja opisano u ovom radu upravo je imalo za zadatak skratiti vrijeme potrebno kako bi se od opisa scene dobio iscrtani prikaz te scene, ali u kraćem vremenu od korištenja isključivo iscrtavatelja, i to uz pomoć dubokog učenja. Iako je problem iscrtavanja kvalitetne slike pomoću nasumičnog uzorkovanja gotovo riješen– možemo dobiti vrlo visoku kvalitetu uz veliki broj uzoraka–željeli bismo izbjeći pretjerano dugo čekanje. To vrijeme se onda može utrošiti na dodatna poboljšanja, primjerice više detalja u sceni, ili kompleksnije animacije u animiranom filmu.

U radu su opisana tri modela koji rješavaju navedeni problem. To su model s izravnim predviđanjem (DPCN), model s predviđanjem pomoću jezgre (KPCN) te rezidualni model s predviđanjem pomoću jezgre. Sva tri modela su se pokazala kao dobrim mogućim rješenjima, ali kada su u pitanju rezultati, KPCN se pokazao najboljim. Rezidualni model malo zaostaje, ali s obzirom na modularnost ima dovoljno prostora za napredak. Model s izravnim predviđanjem je dobar početni model za rješavanje ovog problema, ali s obzirom na sporiju konvergenciju, bolje je koristiti druga dva modela.

Iako smo postigli zadovoljavajuće rezultate, naravno, ostaje mjesta za napredak. Opisani modeli nisu uzeli u obzir temporalnu stabilnost. Ako bismo postupak primijenili na više uzastopnih slika iz animacije, prilikom prikazivanja takve animacije primijetili bismo šum. Taj šum ne bismo primijetili kada bismo promatrali svaku sliku iz animacije zasebno. Temporalna stabilnost može se osigurati dodatnim poboljšanjima rezidualnog modela opisanog u članku [12]. Osim toga, bilo bi zanimljivo istražiti može li se izbaciti pojedine ulazne značajke uz postizanje sličnog ili boljeg rezultat. Time bi se skratilo vrijeme učenja i memorijski zahtjevi metode. Također bi bilo korisno istražiti može li se na neki drugi način pretprocesirati ulazne značajke i time utjecati na rezultat. Postoje i metode koje ne koriste duboko učenje za uklanjanje šuma, npr. [3] te bi možda bilo korisno kombinirati njih i metode dubokog učenja za još bolje rezultate.

LITERATURA

[1] Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, i Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)*, 36(4):97:1–97:14, 2017. doi: 10.1145/3072959.3073708.

[2] Benedikt Bitterli. Rendering resources, 2016. URL `https://benedikt-bitterli.me/resources/`.

[3] Benedikt Bitterli, Fabrice Rousselle, Bochang Moon, José Antonio Iglesias Guitián, David Adler, Kenny Mitchell, Wojciech Jarosz, i Jan Novák. Nonlinearly weighted first-order regression for denoising monte carlo renderings. *Comput. Graph. Forum*, 35 (4):107–117, 2016. doi: 10.1111/cgf.12954. URL `https://doi.org/10.1111/cgf.12954`.

[4] Nikola Bunjevac. Diplomski seminar, 2018.

[5] Nikola Bunjevac. Dokumentacija diplomskog projekta, 2019.

[6] Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, i Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph.*, 36(4):98:1–98:12, Srpanj 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073601. URL `http://doi.acm.org/10.1145/3072959.3073601`.

[7] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[9] Diederik P. Kingma i Jimmy Ba. Adam: A method for stochastic optimization. U *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA,*

USA, May 7-9, 2015, Conference Track Proceedings, 2015. URL http://arxiv.org/abs/1412.6980.

[10] Matt Pharr, Wenzel Jakob, i Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2016. ISBN 0128006455, 9780128006450. URL http://www.pbr-book.org/.

[11] Thijs Vogels. Kernel-predicting Convolutional Neural Networks for Denoising Monte Carlo Renderings. Magistarski rad, ETH Zürich, Switzerland, 2016.

[12] Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, i Jan Novák. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018)*, 37(4):124:1–124:15, 2018. doi: 10.1145/3197517.3201388.

Duboki modeli za uklanjanje šuma u slikama iscrtanim slučajnim uzorkovanjem

Sažetak

Iscrtavanje slučajnim uzorkovanjem postiglo je zavidne rezultate od svojih začetaka. Napretkom tehnologije omogućena su dodatna poboljšanja kvalitete generiranih slika. Ipak, značajan dio vremena troši se na veliki broj uzoraka koji je nužan za željenu kvalitetu rezultata. Nedovoljan broj uzoraka manifestira se u obliku šuma. Ovaj rad razmatra alternativni način uklanjanja šuma primjenom dubokog učenja. Spomenuti pristup pogodan je zbog toga što možemo generirati parove slika s i bez prisutnog šuma. Korišteni su duboki konvolucijski modeli, a kao dodatno poboljšanje korišten je pristup uklanjanja šuma pomoću jezgre koji se pokazao efikasnijim od naivnog pristupa. Prezentirani su postignuti rezultati te predloženi pravci budućeg razvoja.

Ključne riječi: Duboko učenje, računalna grafika, algoritam praćenja zrake, slučajno uzorkovanje.

Deep models for denoising images obtained by Monte Carlo sampling

Abstract

Monte Carlo rendering has made remarkable results since its inception. Technological progress has enabled additional improvements to the quality of the generated images. Nevertheless, a significant part of computing time is spent on large number of samples that are necessary for the desired quality of results. Insufficient number of samples is manifested in the form of a noise. This thesis deals with an alternative way of denoising by applying deep learning. This approach is appropriate because we can generate pairs of pictures with and without noise. Deep convolutional models were used, and in addition a kernel based approach was used for noise removal that proved to be more efficient than a naive approach. Results and the proposed directions for future development are presented.

Keywords: Deep learning, computer graphics, ray tracing, Monte Carlo method.