

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI PROJEKT

**Uklanjanje šuma na slikama
iscrtanim Monte Carlo metodama
primjenom dubokih konvolucijskih
modela**

Nikola Bunjevac

Mentor: *prof. dr. sc. Siniša Šegvić*

Zagreb, siječanj 2019.

SADRŽAJ

1. Uvod	1
1.1. Monte Carlo integracija	3
1.2. Uzorkovanje	3
2. Teorijska podloga	5
3. Skup podataka za učenje	7
3.1. IsCRTavanje scena	7
3.2. Značajke izlaznih slika	8
3.3. Preprocesiranje	10
3.4. Generiranje ulaza neuronske mreže	11
4. Model	14
4.1. Izravna predikcija	14
4.2. Predikcija pomoću jezgre	15
4.3. Učenje modela	16
5. Rezultati	17
5.1. Usporedba modela	17
5.2. Trajanje učenja modela	17
5.3. Primjeri uklanjanja šuma	19
6. Daljnji rad	27
7. Tehnički detalji	28
7.1. Isječci koda	28
8. Literatura	31

1. Uvod

Računalna grafika vrlo je široko i zanimljivo područje računarstva. Ona se bavi izradom vizualnih reprezentacija (slika, video...) iz ostalih načina zapisa u računalu. Primjerice, možemo imati opis scene u tekstualnoj datoteci koju će zatim iscrtavatelj (engl. *renderer*) pročitati i iscrtati prikaz te scene na ekranu.

Prije nego počne iscrtavanje, program prvo mora pročitati i *parsirati* datoteku u kojoj je pohranjen opis scene. Postoji mnogo načina reprezentacije scene i objekata koji se u njoj nalaze. Najčešće je korišten prikaz pomoću poligona, točnije trokuta. Osim pozicija vrhova, svaki objekt ima još neka svojstva kao što su materijali, transformacije, itd. Osim objekata, potrebni su nam još podaci o virtualnoj kameri i osvjetljenju kako bismo mogli iscrtati prikaz. Nakon učitavanja navedenih informacija, obično se izgrađuju strukture za ubrzavanje rada, poput hijerarhija obujmica (engl. *bounding volume hierarchies, BVH*) koje služe za brzo određivanje presjecišta u sceni. Nakon što su učitane sve potrebne informacije i obavljene sve predradnje, može početi postupak iscrtavanja.

Postoje razne tehnike za iscrtavanje scena, a među češće korištenima su rasterizacija i praćenje zrake (engl. *raytracing*). Uobičajeno, grafičke kartice (GPU) koriste rasterizaciju, dok se praćenje zrake obično izvodi na procesoru (CPU), iako se i taj trend postupno mijenja.

Prirodno bismo očekivali precizan prikaz scene na ekranu, no iz iskustva znamo da to nije uvijek tako. Ako se dovoljno približimo zaslonu računala, vidjet ćemo male točkice koje zovemo pikselima. Matematički opis scene koji koristi dvodimenzionalne točke povezane raznim funkcijama nazivamo vektorskog grafikom. Rasterizacija je postupak pretvorbe iz vektorske grafike u rasterski prikaz—niz piksela, točaka ili linija koji zajedno na zaslonu tvore sliku. Ovdje je važno primijetiti kako se rasterska slika sastoji od diskretnih elemenata, dok se vektorski zapis ne sastoji. To je razlog zašto ne možemo uvijek prikazati sve oštro na zaslonu. Primjerice, sliku u vektorskog zapisu možemo sumirati proizvoljno, bez gubitka kvalitete, dok će ista operacija na rasterskoj slici dovesti do toga da ćemo vidjeti pojedine piksele i izgubiti detalje slike.

Drugi važan postupak jest praćenje zrake. To je također postupak stvaranja slike koju možemo prikazati na zaslonu, ali na drugačiji način. U ovom slučaju ćemo za svaki piksel slike “ispucati” zraku iz kamere i pratiti njenu interakciju s površinama u trodimenzionalnoj

sceni. Time ćemo dobiti niz piksela, kao kod rasterizacije, koji ponovo tvore sliku koju možemo prikazati.

Važan pojam je i fotorealistično iscrtavanje koje za cilj ima generirati prikaz scene koji ne možemo razlikovati od fotografije uslikane fotoaparatom koja bi prikazivala istu takvu scenu u stvarnom svijetu. Naravno, teško je scenu iz stvarnog svijeta opisati u računalu, ali ostvaren je značajan napredak na tom području.

Postupak rasterizacije se obično koristi za interaktivnu računalnu grafiku koja se prikazuje u stvarnom vremenu, dok se praćenje zrake koristi za neinteraktivno iscrtavanje (engl. *offline rendering*). U ostatku dokumentacije ćemo se usredotočiti na postupak praćenja zrake i fotorealistično iscrtavanje.

Kako je postupak praćenja zrake vrlo računalno zahtjevan, postoje mnogi pokušaji ubrzavanja tog postupka. Neki se koriste kao komplementarni, dok se neki koriste zasebno.

Donedavno najbolji postupci nisu koristili metode strojnog, tj. dubokog učenja. Jedan od razloga je što takve metode nisu bile široko raširene, a osim toga, nisu bile računalno traktabilne. One također zahtijevaju veliku količinu podataka kako bi donijele željene rezultate što dodatno otežava problem. Ipak, u posljednjih nekoliko godina korištenje dubokog učenja postalo je sve pristupačnije pa je samim time stiglo i do računalne grafike. Možemo reći kako se takvi postupci nadograđuju na prethodne (“klasične”), samo što su fleksibilniji od njih jer se prilagođavaju podacima. Drugim riječima, učenje nad velikom količinom podataka omogućuje nam da ostvarimo generalizaciju nad raznim podacima i uvjetima.

Učenje nad označenim podacima (u našem slučaju parovima šumovitih i referentnih slika) naziva se nadzirano učenje (engl. *supervised learning*).

Još jedan važan pojam koji ćemo koristiti su duboki konvolucijski modeli. Oni se sastoje od niza slojeva koje nazivamo konvolucijskim slojevima. Svaki sloj sadrži jezgre koje se primjenjuju na ulaz u sloj i time dobivamo određeni izlaz koji nazivamo mapama značajki. Svaka jezgra ima težine, a dodatno može imati i pomak (engl. *bias*). Uobičajeno nakon svakog konvolucijskog sloja slijede slojevi sažimanja i aktivacijska funkcija.

Jedna od prednosti takvih modela je velika prilagodljivost, neovisnost o veličini ulaza (što je vrlo korisno kod slika), manji broj parametara od alternativa, itd.

Sljedeći pojam koji smo spomenuli su parametri. Oni su predmet učenja modela strojnog učenja, odnosno njih možemo mijenjati i prilagođavati kako obrađujemo podatke za učenje. Kod konvolucijskih slojeva možemo učiti težine i pomak svake jezgre.

1.1. Monte Carlo integracija

Korištenje Monte Carlo metoda¹ se u posljednje vrijeme pokazalo kao vrlo pogodnim rješenjem problema iscrtavanja. Takve metode se temelje na korištenju nasumično generiranih brojeva kako bi aproksimirale krajnji numerički rezultat.

Iscrtavanje se temelji na rješavanju jednadžbe iscrtavanja koja se još naziva i jednadžbe širenja svjetlosti (engl. *rendering equation, light transport equation*)

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_{S^3} f(\mathbf{p}, \omega_o, \omega_i) L_i(\mathbf{p}, \omega_i) |\cos \theta_i| d\omega_i. \quad (1.1)$$

Pomoću nje računamo izlazno osvjetljenje $L_o(\mathbf{p}, \omega_o)$ u točki prostora \mathbf{p} u smjeru ω_o . Ono se sastoji od dvije komponente: emitirane svjetlosti $L_e(\mathbf{p}, \omega_o)$ te utjecaja sporedne svjetlosti iz svih smjerova kugle S^3 oko točke \mathbf{p} skalirane funkcijom $f(\mathbf{p}, \omega_o, \omega_i)$ i kosinusom kuta između ω_o i ω_i [3].

Navedena jednadžba iznimno je zahtjevna za rješavanje te nema analitičko rješenje, osim za najjednostavnije slučajeve. Postoje mnoge metode za numeričku aproksimaciju integrala, ali one funkciraju dobro samo za integrale niske dimenzionalnosti. Ako imamo više dimenzija, rezultati više nisu zadovoljivi.

Stoga pribjegavamo metodi Monte Carlo aproksimacije integrala [3]. Pravilna primjena te metode daje odlične rezultate i to ju čini jednom od metoda koja daje rezultat najbliži fizikalno ispravnom rezultatu koji bismo dobili kao analitičko rješenje gornjeg integrala.

Korisnost Monte Carlo aproksimacije integrala leži u svojstvu da koristeći nasumičnost za evaluaciju integrala čine stopu konvergencije neovisnu o broju dimenzija. Još jedna prednost je što ova metoda zahtijeva samo mogućnost evaluacije funkcije na nasumičnim pozicijama, tj. funkciju možemo predstaviti crnom kutijom koja za dani ulaz daje pripadajući izlaz. Ne moramo znati kako crna kutija funkcirira iznutra.

Kako bismo dobili procjenu integrala pomoću ovog algoritma, potrebno je provesti dovoljan broj uzorkovanja i uprosječiti dobivene rezultate. Očekivanje ove procjene je jednaktočno vrijednosti integrala [3].

Spomenimo još i Las Vegas metode koje također koriste nasumične brojeve. Razlika između ovih dviju metoda je što Las Vegas metode mogu “različitim putevima” doći uvijek do istog rješenja, dok Monte Carlo metode ne moraju uvijek dati isto rješenje.

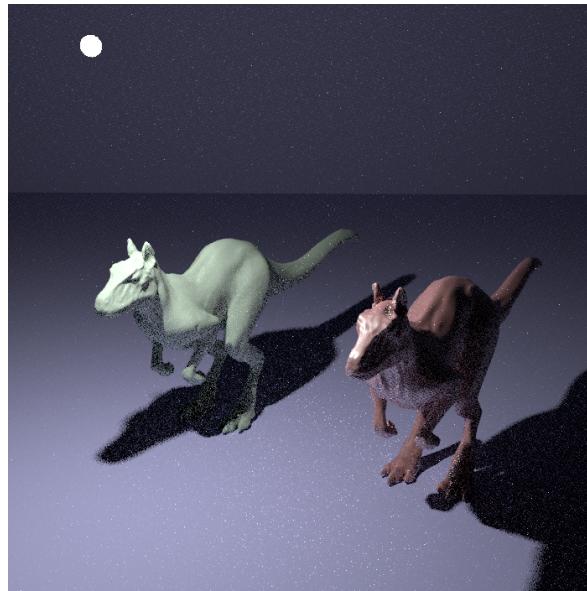
1.2. Uzorkovanje

Postupak praćenja zrake uvelike ovisi o količini uzorkovanja svakog piksela, tj. jednadžbe 1.1. Nakon što ispučamo zraku iz kamere, pratimo gdje će se ona odbiti, te pratimo izvore

¹https://en.wikipedia.org/wiki/Monte_Carlo_method

svjetlosti i utjecaj okoline. Ako uzorkujemo uniformno u svim smjerovima, rezultati neće biti zadovoljavajući. Stoga su razvijene metode koje favoriziraju “teške” situacije, odnosno bolje je da vrijeme utrošimo na uzorkovanje u smjerovima bliže izvorima svjetlosti, područjima koja se nalaze u sjeni, rubovima površina itd.

Naravno, što više uzoraka po pikselu imamo, to će rezultat biti bolji, ali će i vrijeme izračuna biti veće. Savršeno iscrtavanje dobivamo kada broj uzoraka teži u beskonačnost. Stoga, željeli bismo izbjegći veliku količinu uzorkovanja.



Slika 1.1: Primjer šuma pri malom broju uzoraka. Slika je dobivena iscrtavateljem *pbrt*²s 4 uzorka po pikselu.

Problem koji se javlja pri malom broju uzoraka je šum koji prikazuje 1.1. Možemo primijetiti kako se on pogotovo ističe na područjima u sjeni i rubovima. To nam daje naznaku da ćemo se morati posebno usredotočiti na takva područja.

Ono što bismo htjeli postići jest da sa što manje uzoraka imamo što bolji rezultat. Kako bismo to postigli, iscrtat ćemo sliku s malim brojem uzoraka te na takvoj slici pokušati otkloniti šum.

²<https://www.pbrt.org/>

2. Teorijska podloga

Prvo ćemo opisati teoretsku podlogu problema. Označimo piksel p kao \mathbf{x}_p . Tada svaki piksel možemo prikazati kao

$$\mathbf{x}_p = \{\mathbf{c}_p, \mathbf{f}_p\}, \quad \mathbf{x}_p \in \mathbb{R}^{3+D},$$

pri čemu je \mathbf{c}_p boja u RGB zapisu, a \mathbf{f}_p vektor D dodatnih značajki.

RGB zapis se sastoji od 3 boje - crvene, zelene i plave. Takav način zapisa je praktičan i univerzalno rasprostranjen. Važno je napomenuti kako su vrijednosti u RGB zapisu koje možemo prikazati na ekranu u rasponu $[0, 1]$, ali izlaz iz iscrtavatelja nije ograničen navedenim rasponom. Ostale značajke također ne moraju biti u tom rasponu, primjerice dubina.

Dodatne značajke koje želimo izvući iz iscrtavatelja osim boje su:

- normale,
- dubina,
- albedo.

Normale predstavljaju vektore okomite na površinu u točki najbližeg presjecišta zrake iz kamere i površine. Značajka dubine pohranjuje udaljenost od kamere do najbližeg presjekista. Albedo predstavlja boje teksture bez primjenjenog osvjetljenja. Drugim riječima, difuznu komponentu možemo prikazati kao umnožak albeda i efektivnog osvjetljenja (engl. *effective irradiance*) [4].

Referentnu boju piksela označimo s $\bar{\mathbf{c}}_p$. Savršenu referentnu boju ne možemo izračunati jer bismo trebali beskonačan broj uzoraka, ali u praksi je dovoljno da broj uzoraka bude velik (u skupu podataka on iznosi 8192). Tada želimo na temelju ulaznog piksela pronaći procjenu boje koju ćemo označiti s $\tilde{\mathbf{c}}_p$.

Kako bismo postigli što bolju procjenu, obično to ne radimo na temelju samo jednog piksela, već na temelju piksela i njegova susjedstva $N(p)$. Blok vektora (boje i značajke) pridruženih pikselima u susjedstvu od p označimo s \mathbf{X}_p .

Želimo pronaći funkciju $g(\mathbf{X}_p; \theta) = \hat{\mathbf{c}}_p$ koja će na temelju bloka piksela \mathbf{X}_p i parametara θ procijeniti vrijednost boje piksela s uklonjenim šumom. Ta funkcija, odnosno njezini parametri, su središnji pojam problema uklanjanja šuma.

Rješenje problema pronalaska takve funkcije možemo definirati kao

$$\hat{\boldsymbol{\theta}}_{\mathbf{p}} = \arg \min_{\boldsymbol{\theta}} l(\bar{\mathbf{c}}_{\mathbf{p}}, g(\mathbf{X}_{\mathbf{p}}; \boldsymbol{\theta})),$$

gdje je $l(\bar{\mathbf{c}}, \hat{\mathbf{c}})$ funkcija gubitka između referentne boje i procjene. No, tu se javlja problem jer nemamo $\bar{\mathbf{c}}_{\mathbf{p}}$, tj. referentnu boju piksela.

Većina metoda za uklanjanje (Monte Carlo) šuma, mijenjaju funkciju g transformacijom $\boldsymbol{\theta}^T \phi(\mathbf{x}_{\mathbf{q}})$, gdje je $\phi : \mathbb{R}^{3+D} \rightarrow \mathbb{R}^M$. Nakon toga metodom najmanjih kvadrata rješavaju težinsku regresiju

$$\sum_{\mathbf{q} \in N(\mathbf{p})} (\mathbf{c}_{\mathbf{q}} - \boldsymbol{\theta}^T \phi(\mathbf{x}_{\mathbf{q}}))^2 \omega(\mathbf{x}_{\mathbf{p}}, \mathbf{x}_{\mathbf{q}}),$$

gdje ω nazivamo jezgrom ili filtrom. Intuitivno, ona pomaže u razlučivanju između piksela koji su šum i treba im dati manju važnost i onih ispravnih koje želimo više uvažiti.

Problem koji se javlja kod takvih metoda je odabir filtra. On je fiksani i ne nudi fleksibilnost, a mora davati dobre rezultate za vrlo širok spektar ulaznih podataka. Jedno moguće poboljšanje je primijeniti regresiju većega stupnja čime ćemo dobiti i bolji rezultat, ali to nerijetko dovodi do prevelike prilagodbe podacima (engl. *overfitting*).

To nas dovodi do ideje da primijenimo metode dubokog učenja. Za to nam je potreban skup podataka za učenje koji će biti centralni pojam ove metode. Označimo ga s \mathcal{D} . Kako ćemo koristiti nadzirano učenje, \mathcal{D} će se sastojati od N parova primjera za učenje i pripadne referentne boje piksela. To možemo zapisati kao

$$\mathcal{D} = \{(\mathbf{X}_1, \bar{\mathbf{c}}_1), \dots, (\mathbf{X}_N, \bar{\mathbf{c}}_N)\}.$$

U našem slučaju ćemo imati sličice (engl. *patch*) veličine 65×65 piksela dobivene iz većih slika iscrtanih Monte Carlo metodama. Ulazna sličica će imati manji broj uzoraka, dok će referentna sličica iz skupa za učenje imati velik broj uzoraka. Cilj nam je ukloniti šum na ulaznoj sličici i dobiti rezultat što sličniji referentnoj.

3. Skup podataka za učenje

Sada ćemo se pozabaviti analizom skupa podataka (engl. *dataset*). On je javno dostupan na službenoj stranici članka¹. Kako se radi o slikama koje osim RGB boje imaju i niz dodatnih značajki, veličina skupa je očekivano zavidna, gotovo 300 GiB.

3.1. Isrtavanje scena

Skup se sastoji od 1482 nasumične permutacije 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 nasumično mijenjali iz unaprijed zadanoj skupu vrijednosti. Tako dobivene scene iscrtane su 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. Permutirane značajke su

- mapa okoline (rotiranje) i položaj sunca,
- hrapavost materijala (engl. *roughness*),
- teksture unutar pripadnog razreda (npr. pod, tapete, 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. Isrtane slike pohranjene su u EXR formatu koji podržava visoki raspon vrijednosti i više kanala u slici što je praktično za pohranu dodatnih značajki izlaza.

¹<http://drz.disneyresearch.com/~jnovak/publications/KPCN/>

²<https://github.com/tunabrain/tungsten>

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 proglašiti referentnim. Slike s prisutnim šumom su izlaz jednog pokretanja iscrtavatelja, dok je referenca iscrtana posebno kako bi se dobio neovisan rezultat (dručiji *seed* generatora nasumičnih brojeva).

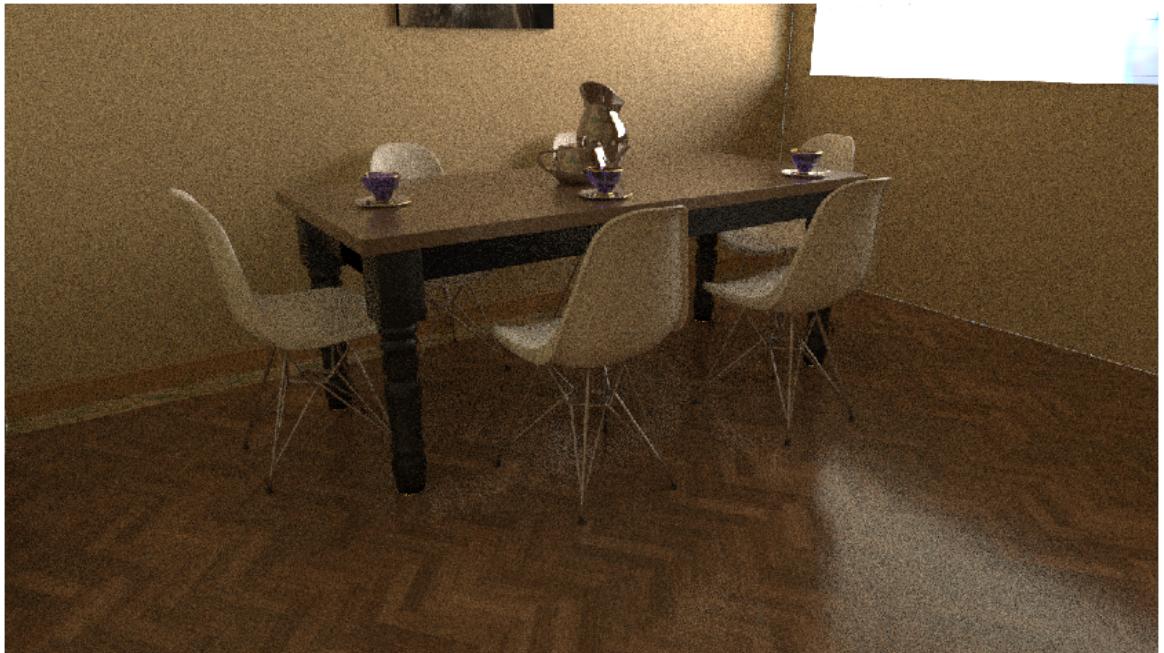
3.2. Značajke izlaznih slika

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 (engl. *high dynamic range, HDR*)) sadrže i dodatne značajke koje ćemo koristiti u učenju modela. Značajke svake EXR datoteke su (pojašnjena 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). Nadalje, difuzni kanal se može podijeliti na albedo i efektivno osvjetljenje (umnožak). Albedo predstavlja boju teksture bez osvjetljenja. Dubina predstavlja udaljenost od kamere prvog presjecišta zrake. 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 [4].

Svaki navedeni kanal ima još pridruženu i svoju varijancu (1 komponenta) te dodatni kanal za izračun procjene varijance (također 1 komponenta) ako je potrebno. Ukratko, svaka slika sadrži ukupno 21 kanal.



Slika 3.1: Glavni kanal (RGB) uz prisutan šum.



Slika 3.2: Glavni kanal (RGB) bez šuma.



(a) Difuzna komponenta boje.



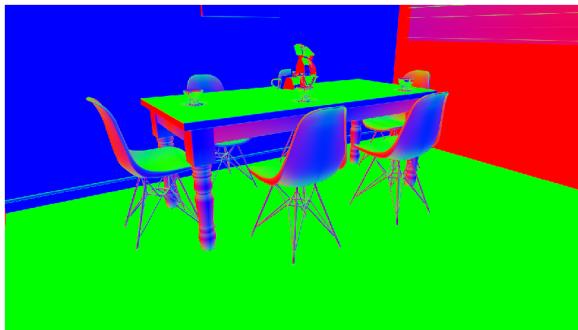
(b) Zrcalna komponenta boje.



(c) Albedo.



(d) Dubina.



(e) Normale.



(f) Varijanca normala.

Slika 3.3: Dodatne značajke.

3.3. Preprocesiranje

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).

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:

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

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

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

3.4. Generiranje ulaza neuronske mreže

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

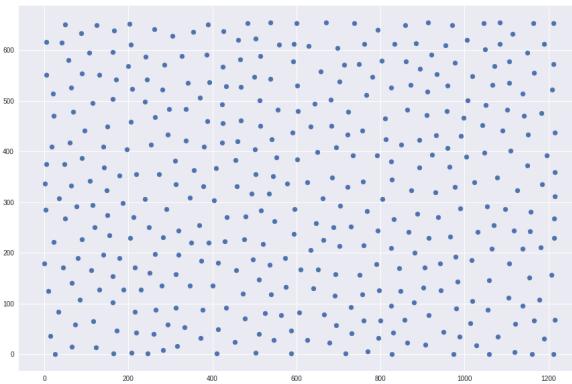
Jedna mogućnost je pozicije sličica odrediti uniformno nasumično, ali to će nam dati puno “laganih” primjera s kojih nije teško ukloniti šum (npr. glatka površina). Autori su stoga odabrali metodu uzorkovanja pomoću mape važnosti (engl. *importance map*) koja određuje vjerojatnost odabira sličice s pojedinog 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.



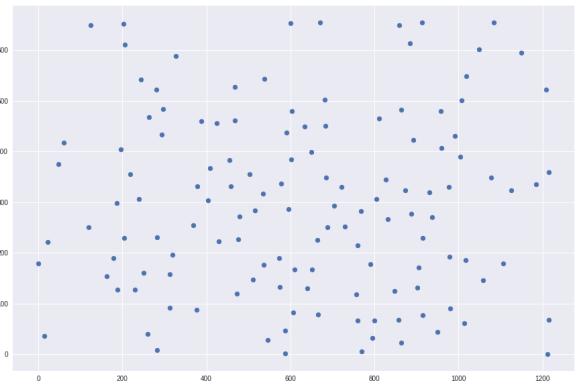
Slika 3.4: Mapa važnosti za sliku 3.1.

Na slici 3.4 možemo vidjeti primjer mape važnosti. Tamnija područja označavaju veću važnost, odnosno mesta gdje očekujemo da će biti teže otkloniti šum i želimo više primjer s takvih područja. Naravno, ne smijemo zanemariti ostala područja, i ukoliko određena pozicija bude više puta odbijena zbog male važnosti, automatski ćemo ju dodati u skup.

Na slikama 3.5a i 3.5b možemo vidjeti utjecaj mape važnosti. Slika 3.5a prikazuje početni uniformni odabir pozicija sličica na velikoj slici. Nakon toga primjenjujemo mapu važnosti i odbacujemo (engl. *pruning*) primjere za koje smatramo da neće suviše doprinijeti učenju. Na slici 3.5b možemo vidjeti kako je uistinu odabran veći broj primjera s tamnijih područja mape važnosti, ali je ipak odabrano i mnoštvo primjera sa svjetlijih područja



(a) Uniformno odabrane pozicije.



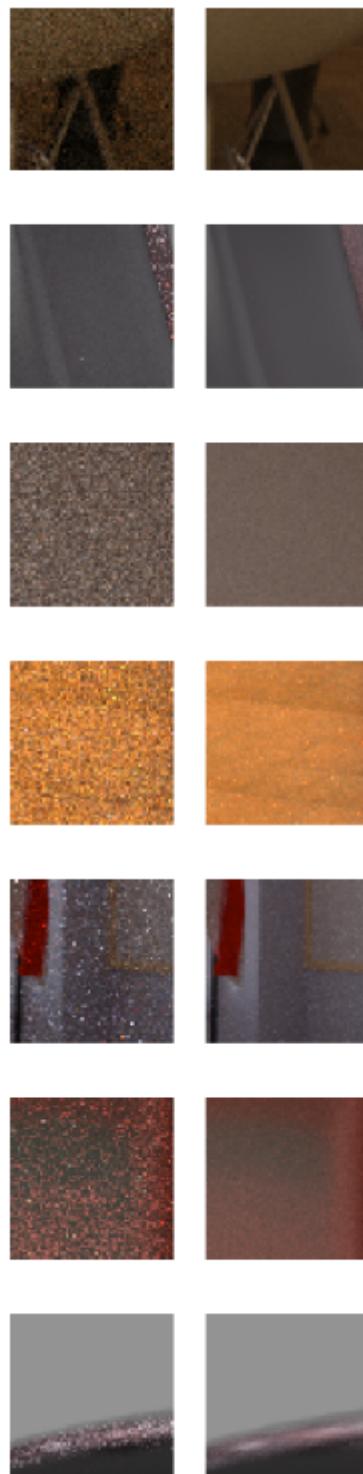
(b) Pozicije nakon odbacivanja.

Slika 3.5: Odabir pozicija prema mapi važnosti.

Nakon što smo obavili pretprocesiranje možemo “sastaviti” ulaz u model koji ćemo prikazati kao

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

pri čemu su G_x i G_y gradijenti odgovarajućih značajki u x i y smjeru, a $\tilde{\mathbf{c}}$ i $\tilde{\sigma}^2$ mogu pripadati difuznoj ili zrcalnoj komponenti.



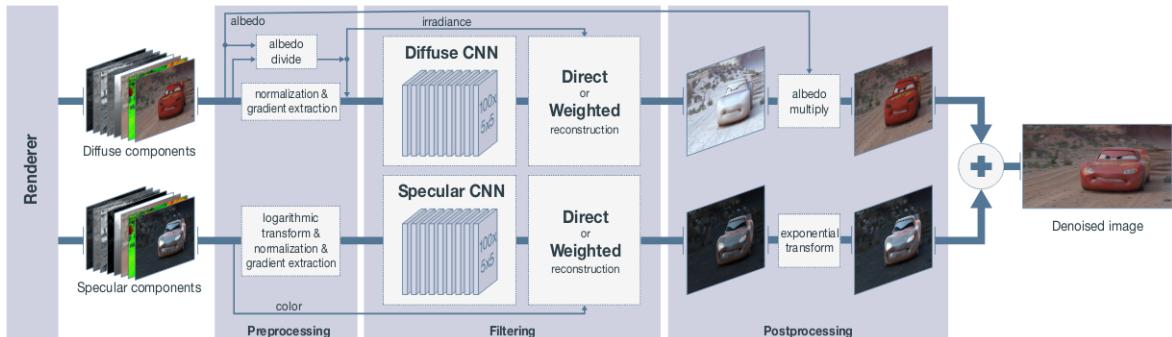
Slika 3.6: Primjeri parova odabralih sličica iz skupa za učenje (lijevo ulaz, desno referenca).

4. Model

Opisat ćemo dvije varijante modela:

1. izravna predikcija,
2. predikcija pomoću jezgre.

Obje varijante imaju istu generalnu strukturu, ali se razlikuju u izlazu. Glavni dijelovi su dvije konvolucijske podmreže koje ćemo zvati difuzna i zrcalna. Jedna podmreža je zadužena za uklanjanje šuma na difuznoj komponenti, dok je druga zadužena za zrcalnu komponentu boje. Razlog ovakve podjele leži u različitim karakteristikama tih dviju komponenti—difuzna ima manji raspon vrijednosti i glatkija je, dok zrcalna komponenta ima veći raspon i drugačiju distribuciju vrijednosti. Arhitekturu mreže možemo vidjeti na slici 4.1.



Slika 4.1: Arhitektura modela.

Konkretnije, podmreže se sastoje od L konvolucijskih slojeva (u našem slučaju $L = 9$), od kojih svi osim zadnjeg imaju aktivacijsku funkciju ReLU. Nema slojeva sažimanja. Posljednji sloj nema aktivacijsku funkciju i izlaz mu ovisi o varijanti modela. Također, svaki konvolucijski sloj osim posljednjeg sadrži 100 jezgri dimenzija 5×5 . U nastavku ćemo prvo opisati izravnu predikciju.

4.1. Izravna predikcija

Konvolucijski model s izravnom predikcijom (engl. *direct prediction convolutional network*, *DPCN*) kao izlaz izravno daje predikciju boje piksela bez šuma. Izlaz podmreža je istih

dimenzija kao i slika, a sadrži tri komponente–crvenu, zelenu i plavu. Prednost ove inačice je jednostavnost, ali se pokazala sporijom za učenje.

Dimenziije 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: $B \times 3 \times H \times W$.

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.

Jedan od problema ove varijante je zamućenje slike (engl. *blurring*), što će biti pokazano u rezultatima. Osim toga, kako nema ograničenja u vrijednostima, model treba više vremena kako bismo ga istrenirali jer ima veći prostor pretraživanja. To dovodi do pojave artefakata na slikama.

4.2. Predikcija pomoću jezgre

Druga varijanta modela je konvolucijska mreža s predviđanjem pomoću jezgre (engl. *kernel prediction convolutional network, KPCN*). Ona kao izlaz podmreža daje jezgru (engl. *kernel*) za svaki piksel čijom primjenom nad susjedstvom uklanjamo šum sa središnjeg piksela. Jezgru dobivamo tako da zadnji sloj kao izlaz da onoliko mapi značajki koliko težina jezgra sadrži.

Dimenziije 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: $B \times k^2 \times H \times W$.

Primjerice, u članku je dimenzija jezgre $k = 21$. Tada 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. To je otprilike 1.6 puta više parametara od prethodne varijante.

Dodatno, izlaz se prvo treba “provući” kroz aktivacijsku funkciju Softmax 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 Softmax je sljedeće [1]:

- Osigurava da procjena leži u konveksnoj ljudsci 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 [4].

4.3. Učenje modela

Model treniramo u dva koraka. Prvo treniramo svaku podmrežu zasebno, a zatim treniramo cijelu mrežu s manjim korakom učenja kako bismo ju fino podesili. Korištenjem mini grupa posjećujemo generaliziranje mreže i sprječavamo prenaučenost.

Kao algoritam učenja korišten je ADAM, a parametri su inicijalizirani Xavier metodom. Korak učenja za skup *Tungsten* iznosi 10^{-4} . Kao funkcija gubitka, korištena je L_1 koja računa apsolutnu razliku između RGB komponenti odgovarajućih piksela. Prilikom učenja, kod konvolucije nije korišteno nadopunjavanje (engl. *padding*).

5. Rezultati

Usporedimo prvo oba modela po njihovim svojstvima. Model s izravnom predikcijom u nastavku zvat ćemo DPCN, a model s predikcijom pomoću jezgre zvat ćemo KPCN. Svi eksperimenti su provedeni na Google Colab GPU instancama.

5.1. Usporedba modela

Podaci su prikazani u tablici 5.1.

Tablica 5.1: Usporedba modela

	DPCN	KPCN
Broj parametara	1828303	2923741
Broj slojeva	9	9
Veličina izlaza	$B \times 3 \times H \times W$	$B \times k^2 \times H \times W$
Aktivacijska funkcija na izlazu	-	Softmax

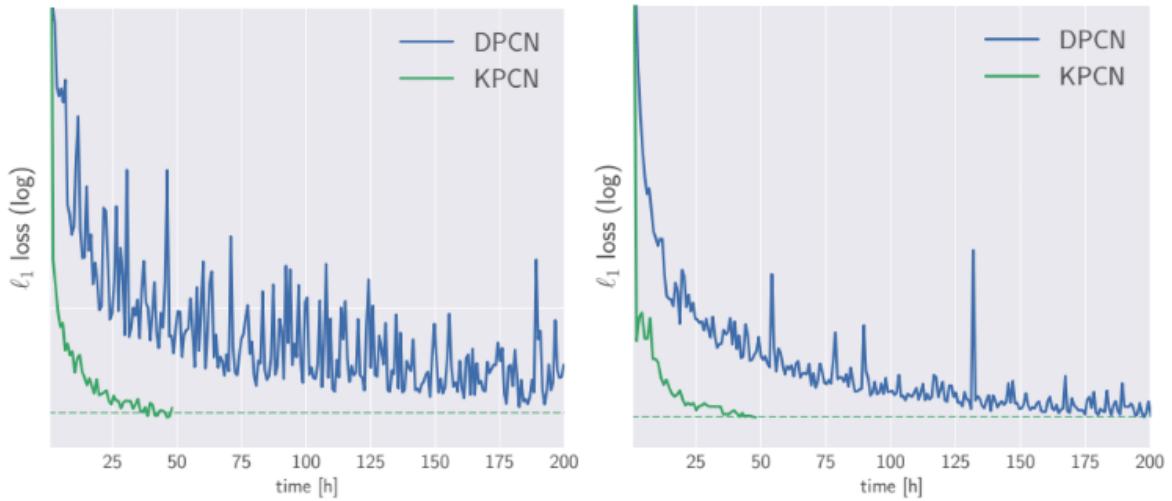
Kao što možemo vidjeti, mreže su vrlo slične, ali KPCN zbog većeg izlaza ima znatno više parametara. Također, jezgre koje dobijemo na izlazu potrebno je primijeniti na svaki piksel, odnosno njegovo susjedstvo što nije trivijalna operacija. To model KPCN čini inherentno sporijim od drugog modela. Autori članka su zbog toga operaciju primjene jezgara implementirali u jeziku niže razine (C/C++) kao ekstenziju koja se poziva iz biblioteke Tensorflow iz Pythona.

5.2. Trajanje učenja modela

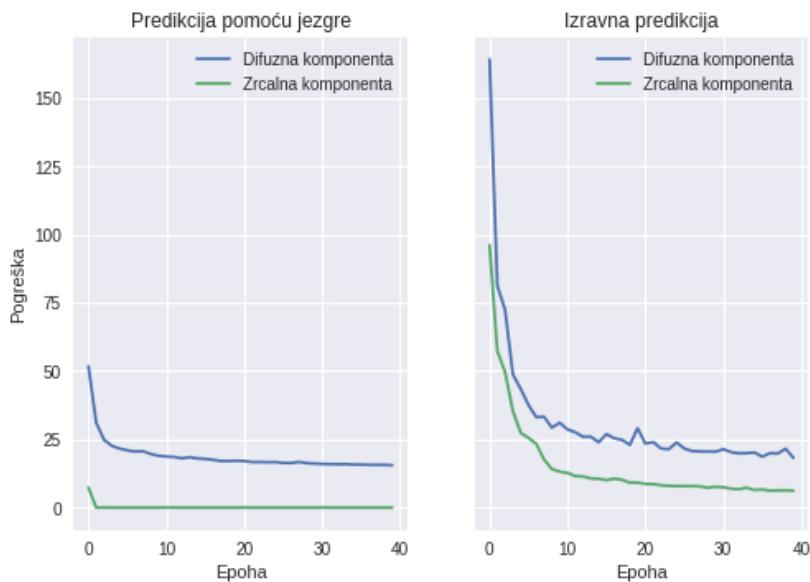
Druga zanimljiva usporedba je potrebno vrijeme učenja između modela. Oba modela mogu doći do iste pogreške, ali KPCN to čini znatno brže, otprilike 5-6 puta.

Kao eksperiment, provjereno je vrijedi li uistinu tvrdnja da je KPCN brži u učenju od DPCN-a. Učenje je provedeno na malom podskupu sa stopom učenja 10^{-4} u trajanju od 40

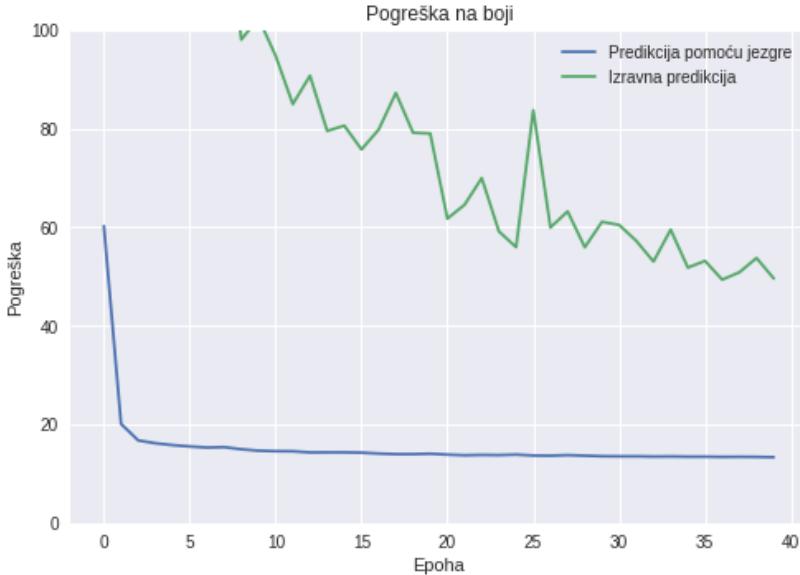
epoha. Rezultat možemo vidjeti na slici 5.2.



Slika 5.1: Usporedba vremena treniranja podmreža (lijevo difuzna, desno zrcalna), preuzeto iz članka [1].



Slika 5.2: Usporedba vremena učenja difuzne i zrcalne podmreže u oba modela.



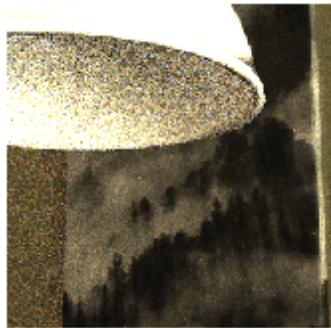
Slika 5.3: Usporedba vremena učenja krajnjeg izlaza oba modela.

Prikazani grafovi pokazuju da je KPCN uistinu brži u učenju od konkurenta. Razlog tome je što DPCN nema ograničenja pa ima mnogo veći prostor pretraživanja. Mora se prilagoditi različitim rasponima vrijednosti, mora uskladiti RGB komponente kako ne bi došlo do pomaka u boji, itd. KPCN se “ogradio” od toga korištenjem jezgara i funkcije Softmax što mu omogućava da prije počne učiti ono glavno–kako ukloniti šum na slici.

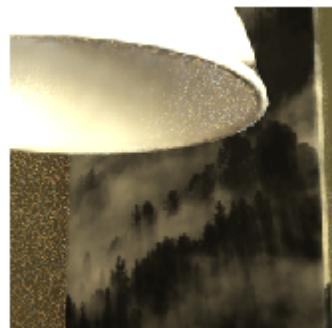
Kao dodatna informacija, učenje (40 epoha) DPCN-a je trajalo oko 60 minuta, dok je učenje KPCN-a trajalo oko 78 minuta što pokazuje kako je aplikacija jezgara prilično zah-tjevna operacija. U vlastitim eksperimentima je implementirana u Pythonu (PyTorch) i nije optimirana kao u članku.

5.3. Primjeri uklanjanja šuma

Sada ćemo prikazati neke primjere uklanjanja šuma na slikama koje model nije nikada vidio. Gornja slika je ulaz, srednja izlaz, a donja je referentna.



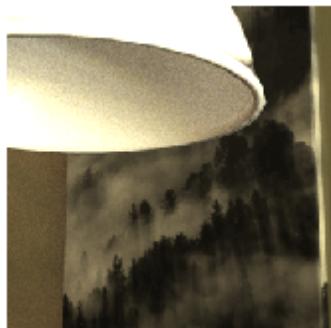
(a) Slika s prisutnim šumom.



(b) Izlaz iz KPCN-a.



(c) Izlaz iz DPCN-a.



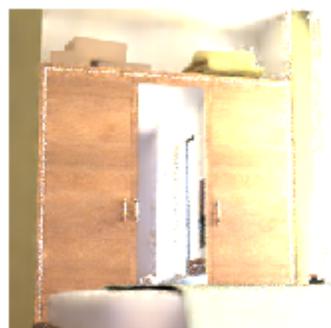
(d) Referentna slika.

Slika 5.4: Primjer uklanjanja šuma.

Na slici 5.4c možemo vidjeti probleme s kojima se DPCN suočava, a to su pomak u boji i zamućenje. Iako rekonstruirana slika nema vidljiv šum kao na ulaznoj slici, boja nije onakva kakva bi trebala biti—tamnija je od reference i ulaza. Za razliku, KPCN je zadržao razinu boje, ali mu je ponegdje ostao vidljiv šum koji bi bio uklonjen treniranjem na više primjera. Osim toga, rubovi na slici c) nisu oštri kao na ostalim slikama što je rezultat zamućenja.



(a) Slika s prisutnim šumom.



(b) Izlaz iz KPCN-a.

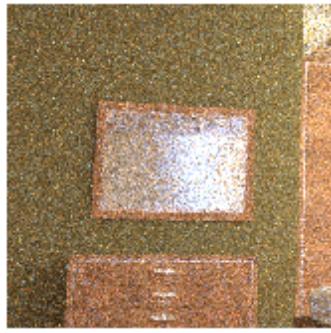


(c) Izlaz iz DPCN-a.

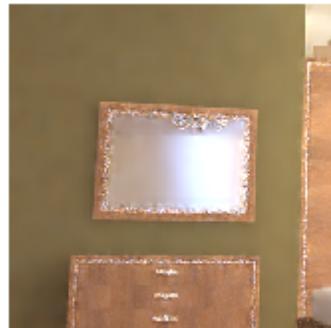


(d) Referentna slika.

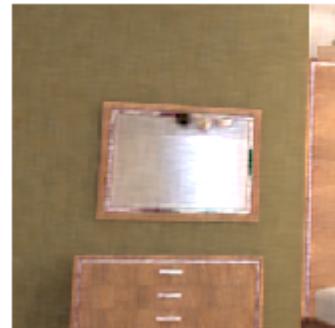
Slika 5.5: Primjer uklanjanja šuma.



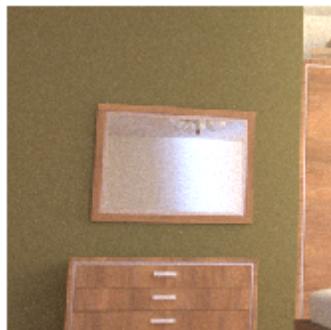
(a) Slika s prisutnim šumom.



(b) Izlaz iz KPCN-a.



(c) Izlaz iz DPCN-a.



(d) Referentna slika.

Slika 5.6: Primjer uklanjanja šuma.

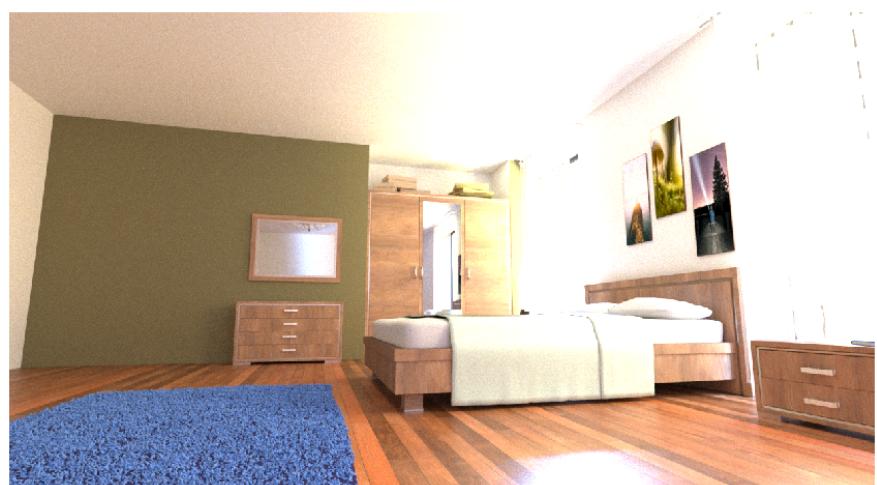
Na slici 5.6c možemo vidjeti još jedan problem s modelom DPCN, a to je pojava artefakata (ogledalo). Također, zid ne izgleda glatko i ponovo imamo pomak u boji.



(a) Slika s prisutnim šumom.



(b) Izlaz iz KPCN-a.



(c) Referentna slika.

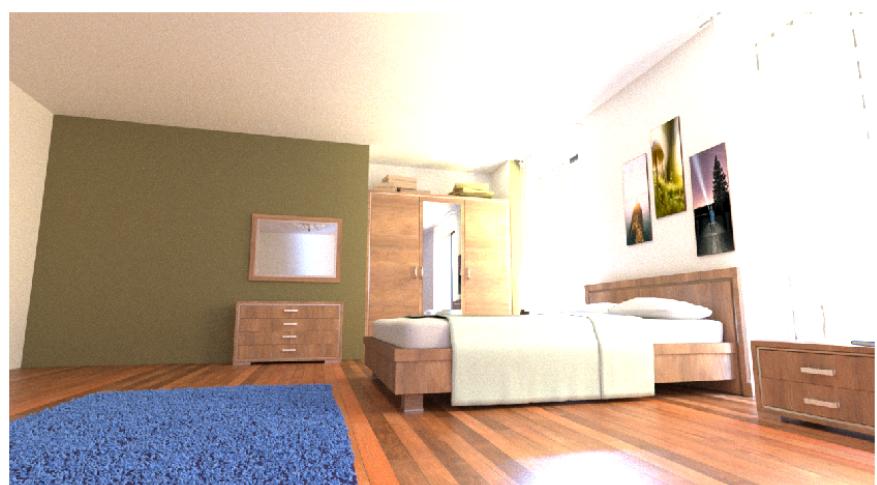
Slika 5.7: Primjer uklanjanja šuma (KPCN).



(a) Slika s prisutnim šumom.



(b) Izlaz iz DPCN-a.



(c) Referentna slika.

Slika 5.8: Primjer uklanjanja šuma (DPCN).



(a) Slika s prisutnim šumom.



(b) Izlaz iz DPCN-a.



(c) Referentna slika.

Slika 5.9: Primjer izraženog zamućenja modela DPCN.

Većina problema viđenih na slikama bi se mogla popraviti učenjem na većem skupu

podataka i više provedenih epoha. Prikazani rezultati ne koriste cijeli skup za treniranje, već mali podskup, ali unatoč tome modeli daju prilično dobre rezultate, barem na prvi pogled.

6. Daljnji rad

Opisana metoda se pokazala odličnom i u trenutku objave je bila *state-of-the-art*. Ipak, ima određenih nedostataka, a to je primjerice temporalna nestabilnost. Ako bismo ovu metodu primijenili na nekoliko uzastopnih slika iz animacije, dobili bismo efekt zrnatosti, odnosno pojavio bi se šum koji možda ne bi bio vidljiv na pojedinoj slici, ali bi došao do izražaja pri brzoj promjeni između njih.

Osim toga, autori su sugerirali da bi bilo dobro proučiti druge načine odabira sličica za učenje. Trenutna metoda odabira funkcioniра vrlo dobro, ali ima mogućnosti za napredak. Isto vrijedi i za funkciju gubitka.

Također, bilo bi zanimljivo vidjeti druge pristupe dubokog učenja na ovom problemu, npr. generativne modele. Paralelno s člankom [1] je pisan i članak¹ u kojem je naglasak bio na autoenkoderima i većoj interaktivnosti metode (korištenje u stvarnom vremenu).

¹https://research.nvidia.com/sites/default/files/publications/dnn_denoise_author.pdf

7. Tehnički detalji

Eksperimentiranje je provedeno u programskom jeziku Python verzije 3. Korišten je projekt Google Colab¹ koji besplatno nudi vremenski ograničene računalne resurse za potrebe strojnog učenja. Okolina je Jupyter Python bilježnica koja omogućava interaktivni rad te kombiniranje koda i teksta. Za manipulaciju EXR datotekama korištena je biblioteka pyexr² jednog od autora [1]. Za izradu i učenje neuronskih mreža korišten je računalni okvir PyTorch³ koji se pokazao jako intuitivnim i jednostavnim za korištenje. Jupyter bilježnica nalazi se na ovoj adresi.

7.1. Isječci koda

Isječak koda 7.1: Funkcija koja vraća inicijaliziranu podmrežu.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

recon_kernel_size = 21

def make_net(n_layers, mode):
    # create first layer manually
    layers = [
        nn.Conv2d(input_channels, hidden_channels, kernel_size),
        nn.ReLU()
    ]

    for l in range(n_layers - 2):
        layers += [
            nn.Conv2d(hidden_channels, hidden_channels, kernel_size),
            nn.ReLU()
        ]

    out_channels = 3 if mode == 'DPCN' else recon_kernel_size**2
    layers += [nn.Conv2d(hidden_channels, out_channels, kernel_size)]

    for layer in layers:
        if isinstance(layer, nn.Conv2d):
            nn.init.xavier_uniform_(layer.weight)

    return nn.Sequential(*layers)
```

¹<https://colab.research.google.com/>

²<https://github.com/tvogels/pyexr>

³<https://pytorch.org/>

Isječak koda 7.2: Funkcije za transformaciju podataka.

```
def preprocess_diffuse(diffuse, albedo):
    return diffuse / (albedo + eps)

def postprocess_diffuse(diffuse, albedo):
    return diffuse * (albedo + eps)

def preprocess_specular(specular):
    assert(np.sum(specular < 0) == 0)
    return np.log(specular + 1)

def postprocess_specular(specular):
    return np.exp(specular) - 1

def preprocess_diff_var(variance, albedo):
    return variance / (albedo + eps)**2

def preprocess_spec_var(variance, specular):
    return variance / (specular+1e-5)**2

def gradients(data):
    h, w, c = data.shape
    dX = data[:, 1:, :] - data[:, :w - 1, :]
    dY = data[:, :, 1:] - data[:, :h - 1, :, :]
    # padding with zeros
    dX = np.concatenate((np.zeros([h, 1, c]), dX), dtype=np.float32, axis=1)
    dY = np.concatenate((np.zeros([1, w, c]), dY), dtype=np.float32, axis=0)

    return np.concatenate((dX, dY), axis=2)
```

Isječak koda 7.3: Kostur postupka učenja podmreža.

```
criterion = nn.L1Loss()

optimizerDiff = optim.Adam(diffuseNet.parameters(), lr=learning_rate)
optimizerSpec = optim.Adam(specularNet.parameters(), lr=learning_rate)

for epoch in range(epochs):
    for i_batch, sample_batched in enumerate(dataloader):
        # get the inputs
        X_diff = sample_batched['X_diff'].permute(permutation).to(device)
        Y_diff = sample_batched['Reference'][:, :, :, :3].permute(permutation).to(device)

        # zero the parameter gradients
        optimizerDiff.zero_grad()

        # forward + backward + optimize
        outputDiff = diffuseNet(X_diff)

        if mode == 'KPCN':
            X_input = crop_like(X_diff, outputDiff)
            outputDiff = apply_kernel(outputDiff, X_input)

        Y_diff = crop_like(Y_diff, outputDiff)

        lossDiff = criterion(outputDiff, Y_diff)
        lossDiff.backward()
        optimizerDiff.step()

        # get the inputs
        X_spec = sample_batched['X_spec'].permute(permutation).to(device)
        Y_spec = sample_batched['Reference'][:, :, :, 3:6].permute(permutation).to(device)

        # zero the parameter gradients
        optimizerSpec.zero_grad()

        # forward + backward + optimize
```

```

outputSpec = specularNet(X_spec)

if mode == 'KPCN':
    X_input = crop_like(X_spec, outputSpec)
    outputSpec = apply_kernel(outputSpec, X_input)

Y_spec = crop_like(Y_spec, outputSpec)

lossSpec = criterion(outputSpec, Y_spec)
lossSpec.backward()
optimizerSpec.step()

# calculate final ground truth error
with torch.no_grad():
    albedo = sample_batched['origAlbedo'].permute(permuation).to(device)
    albedo = crop_like(albedo, outputDiff)

    outputFinal = outputDiff * (albedo + eps) + torch.exp(outputSpec) - 1.0

    Y_final = sample_batched['finalGt'].permute(permuation).to(device)
    Y_final = crop_like(Y_final, outputFinal)

    lossFinal = criterion(outputFinal, Y_final)

```

8. Literatura

- [1] Steve Bakó, 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] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016.
<http://www.deeplearningbook.org>.
- [3] Matt Pharr, Wenzel Jakob, i Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd izdanju, 2016. ISBN 0128006455, 9780128006450. URL <http://www.pbr-book.org/>.
- [4] Thijs Vogels. Kernel-predicting Convolutional Neural Networks for Denoising Monte Carlo Renderings. Magistarski rad, ETH Zürich, Switzerland, 2016.