Primjena grafičkih procesora u projektiranju građevina korištenjem strojnog učenja

Tokalić, Adis

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Diplomski studij informatike, smjer informacijski i komunikacijski sustavi

Adis Tokalić

Primjena grafičkih procesora u projektiranju građevina korištenjem strojnog učenja

Diplomski rad

Mentor: doc. dr. sc. Vedran Miletić

Rijeka, 19. siječanj 2023.

Sažetak

Umjetna inteligencija je znanstvena grana informatike koja je u zadnjih nekoliko godina vidjela intenzivno razvijanje i usavršavanje, posebno zahvaljujući probitku metoda strojnog učenja u brojne domene primjene. U ovom diplomskom radu, napravit će se kratki osvrt na primjene strojnog učenja uz tehnologije koje se trenutno koriste i dosadašnja istraživanja koja su pomogla njihovom razvoju. Rad daje primjer praktične primjene strojnog učenja u projektiranju građevina, specijalno stvaranju digitalnih tlocrta korištenjem grafičkih procesora.

Ključne riječi

umjetna inteligencija, strojno učenje, arhitektura, neuronske mreže

Tablica sadržaja

1. Uvod

Automatizacija kao grana znanosti je jedna od najažurnijih tematika u informatici. Automatiziranje bilo kakve grane navedene znanosti se čini kao interesantan koncept, te je u današnjici prakticiran pomoću umjetne inteligencije. Ovaj diplomski rad služi kao osvrtnica i pregled trenutno popularnih metoda i alata u umjetnoj inteligenciji, detaljni opis navedenih alata, te kao konceptualni primjer njihove primjene na praktičnom primjeru.

Umjetna inteligencija i storjno učenje se koriste prvobitno kao alati automatizacije, te se navedena grana informatike širi na različita znastvena područja, te različite nauke. Kao primjer konceptualnog korištenja metodologije strojnog učenja, generirat ćemo interpertacije umjetne inteligencije digitalnih tlocrta prema nekoliko glavnih baznih podataka. Koristit ćemo dve baze podataka, te ćemo u njima pokazati kako se ovakvi alati koristi i njihove mogućnosti pri automatizaciji izrade digitalnih tlocrta.

Pri početku ovoga rada, moramo prvo objasniti same alate koje koristimo, pa onda i same istraživanja koja su relevantna i služe kao teorijski temelj ovog rada. Zatim ćemo usporediti naš model i postojeće u terminima dobivenih rezultata i završiti sa zaključkom.

2. Korišteni alati

2.1. Strojno učenje

Kako bismo razumjeli umjetnu inteligenciju kao alat, moramo prvo objasniti glavne referentne pojmove u navedenoj znanosti. Prema glavnoj adaptiranoj definiciji, "Umjetna inteligencija je naziv koji pridajemo svakom neživom sustavu koji pokazuje sposobnost snalaženja u novim situacijama." Ovo je gruba i općenita teza svakog algoritma strojnog učenja. Umjetna inteligencija se definira u strukturi proceduralnih algoritama, što znači da se fokusiramo na same rezultate danog programa, a ne način na koji dolazimo do rezulata.
Prvi pojam koji moramo objasniti je pojam neuralnih mreža, kako se alati i sama konceptualizacija ovih alata strogo temelji na ovome principu.[20]

2.1.1. Neuralne mreže

Neuralne mreže, poznatije kao umjetne neuronske mreže(CNN), područje su strojnog učenja i temelj su za sve teze dubokog strojnog učenja.[7]
Svaka umjetna inteligencija se komprimira od nekoliko do većih brojeva slojeva, te počinju s ulaznim podacima i završavaju s izlaznim podacima. Svaki sloj se sastoji od neurona, te svaki neuron ima određene granice i određenu težinu. Neuroni između slojeva su međusobno povezani, te ako je izlaz prijašnjeg neurona iznad navedene određene granice izlaza, neuronska jedinica se aktivira i šalje određeni izlaz naveden kao ulaz u sljedeći sloj neuronske mreže, odnosno ne šalje ništa, ovisno o parametru koji se dobiva u samoj mreži.

Glavni aspekt pri korištenju samih neuralnih mreža su podaci koji se koriste. Neuralne mreže su ovisno o ulazu koji im je pridijeljen. Glavni cilj neuralnih mreža je davanje izlaza koji se može mjeriti mjerom točnosti, te korelacija da navedena mjera bude optimalno maksimizirana.

Proces optimizacije točnosti navedenog mjere se naziva treniranjem neuralne mreže te je važan u korak u samom programiranju navedene mreže.

Svaki neuorn mreže se može razmatrati kao individualni linearni regresijski model, koji se sastoji od ulaza, težine, biasa/određene granice i izlaza.

Glavna formula za izračunavanje izlaza neuorna bi bila sastavljena na sljedeći način:

$$\sum wixi + bias = w1x1 + w2x2 + w3x3 + bias \text{ [5]}$$

Prilikom sume svih težina i svih neurona, korelacija izlaza i navedene jednažbe bi bila formulirana kao:

$$\text{Izlaz} = f(x) = 1 \text{ ako } \sum w1x1 + b >= 0,$$

$$0 \text{ ako } \sum w1x1 + b < 0$$

Kada utvrdimo ulazni sloj neuronske mreže, dodajemo težine na individualne neuorne. Uloga težina i težinskih funkcija je određivanje važnosti varijabla koje se kreću kroz neuronsku mrežu, te veće i važnije varijable imaju veću ulogu na samu izlaznu funkciju.[6] Svi ulazi su pomnoženi s njihovom težinom, te onda sumirani, kako bi standardizirali važnost određenih neurona. Izlazna funkcija također prolazi kroz aktivacijsku funkciju, te ako izlaz navedenog neurona prelazi određenu granicu, neuorni aktiviraju navedenu funkciju, te se ti podaci prosljeđuju dalje, u dolazeće slojeve. Drugim riječima, izlaz jednog neurona je ulaz neuorna sljedećeg sloja.

Kako bi izmjerili samu točnost našeg izlaza, bitno je izabrati prave mjere, ovisno o vrsti mreže koje imamo. Srednja kvadratna pogreška(MSE)[8], je jedna od standardiziranih mjera za računanje točnosti naše mreže.

Analizom opisa rada navedenih mreža možemo zaključiti da je cilj svake neuralne mreže, minimizirati funckiju točnosti, u ovom slučaju srednju kvadratnu pogrešku, kako bi mogli imati najveću točnost u mreži.

Proces optimizacije, gdje se težine mreže optimiziraju i mijenjaju, se događa u koracima, a naziva se treniranjem mreže. Prilikom treniranja naša baza podataka se dijeli na dva skupa, skup za treniranje i skup za testiranje mreže.

2.1.1.1. Konvolucijske neuralne mreže

Postoji nekoliko glavnih vrsta neuralnih mreža. Perceptron je najstarija vrsta neuralne mreže, te sadrži jedan neuron. Višeslojni perceptroni su prihvaćeni kao današnji standard, zbog korištenja više slojeva i više neurona u mreži. Međutim, u ovom diplomskom radu, fokusirat ćemo se na konvolucijske neuralne mreže. Konvolucijske mreže, su slične višeslojnih perceptronima, ali se diferenciraju u grani korištenja. Konvolucijske mreže[6] se koriste u prepoznavanju slika, patterna i obradi vizualnih i audiovizualnih problematika.

2.2. Vrste slojeva konvolucijske mreže

Kako bi razumjeli daljnje korake u praktičnoj primjeni I samu tezu iza koncepta konvolucijskih mreža, moramo započeti s analizom tipova slojeva u neuralnim mrežama

2.2.1. Konvolucijski sloj

Konovlucijski sloj je temeljni sloj neuralnih mreža, te konvolucijski slojevi drže veličinu opterećenja na samoj mreži. Ovaj sloj skalarno množi dvije matrice, od kojih prva predstavlja niz parametara karakteristika, poznatiji kao kernel. Druga matrica predstavlja ograničeni odjelak receptivnog polja, to jest polja kojeg rasčlanjujemo na više komponenata. Kernel je manji od samog ulaznog sloja, u ovom slučaju, naših slika, ali je puno detaljniji u opisu karakteristika. Za dani primjer, ako nam je slika kodirana u RGB kanalu, te se sastoji od 3 kanala, kernel će biti znatno manji od slike, ali će imati podatke koji nam daju detaljniji uvid za sva 3 kanala. Prilikom prelaska iz jednog sloja u drugi, kernel uzima podatke visine I širine dok se sama ulazna slika obrađuje. Ovakav pristup rezultira u dvodimenzionalnom polju, koji je ništa druga nego aktivacijska mapa. Drugim riječima aktivacijska mapa su nam parametri kojima određujemo koji pikesli prelaze preko granice određenja za našu aktivacijsku funkciju.
Ako imamo ulazno sliku određene iste širine I visine W , te brojem kanala D i određeni broj kernela sa veličinom F I S brojem segmenata, te graničnim parametrom P, izlazna veličina se može odrediti[6].

Ovakva izlazni podatak će biti Veličine W_{izl} x W_{izl} x D_{izl}, što nam pokazuje da dobivamo isti skalabalini omjer kao i na ulaznu. Drugim riječima, dobivamo sliku izlaza s istom širinom, visinom i brojem kanala kao i ona ulazna.

Glavna motivacije za korištenje konovlucijskih slojeva su: gusta interakcija, međusobno dijeljenje parametara i jednaka standardizacija ulaza i izlaza.

Svaka neuralna mreža koristi skalarne produkte kao produkt matrice parametara kako bi opisala interakciju između ulaza i izlaza. Međutim, konvolucijske mreže imaju gustu interakciju. Korištenjem manjeg kernela nego ulaznih dimenzija, segreriramo odjeljke samog ulaznog podatka, u ovom slučaju slike, za detaljnjiju karakteristika

i interakcije piksela koji su u zajedničkim kernelima. Ovakav pristup daje detaljnije parametre i bolju mjere točnosti, te replicira ulazne karakteristike s visokom preciznošću jer gleda korelaciju grupe piksela kao mogući entitet, a ne samo izoliranu jedinicu.

Dijeljenje parametara uzorokuje standardizaciju izlaza i ulaza, što drugim riječima znači, zbog zajedničkih kernela, ako smo promjenili ulaznu jedinicu, izlazna jedinica se također mijenja

2.2.2. Pooling sloj

Poling slojevi mijenjaju izlaz mreže na pojedinačnim mjestima u neuornskim mrežama, tako da deriviraju statističke karakteristike bližnjih izlaza. Ovakvim pristupom, smanjujemo veličinu same reprezentacije mreže, što smanjuje broj izračuna i težinskih optimizacija u mreži. Pooling slojevi mogu imati različite pooling funkcije, norme po kojima karakteriziraju derivacije same mreže u određenom sloju.
Najpopularnija vrsta poolinga je max pooling, koji određuje maksimalni izlaz iz susjednih neurona. Na primjer, matricu neuorna 4x4, možemo smanjiti na matricu 2x2[32] preko max pooling funkcije, tako da maksimalnu vrijednost u segmentima 2x2 [32] opisujemo kao jednu vrijednost u matrici 2x2.

Izlazna veličina pooling funkcija može biti određena funkcijom

$$\mathbf{Wizl = (W-F)/S + 1}$$

Pooling slojevi se koriste kako bi se minimizirali brojevi karakteristika, te smanjilo vrijeme izračuna i vrijeme optimizacije težinskih funkcija.

2.2.2.1. Nelinearni slojevi

Ovi slojevi se ne koriste u samoj konvolucijskoj mreži, kako je konvolucijska mreža linearno regresivni model, nego se koriste nakon samih konvolucijskih mreža, kako bi aktivacijske funkcije dobile karakteristiku nelinearnosti.
Sigmoid slojevi ima matematičku formulaciju.[31] Ovaj sloj uzima cijelobrojne brojeve iz same karakterizacije i standardizira ih na skalu od 0 do 1. Međutim kada je aktivacija blizu krajnjih vrijednosti karakterizacije, gradijent nema detaljnju vrijednost te je vrijednost zamijenjena nulom. Ako nam je gradijent vrijednosti malen, gradijent uništava sam sebe, te dobivamo nekorektnu standarizaciju.

Tanh sloj, koristeći tangens kao aktivacijsku nelinearnu funkciju, standardizira vrijednosti od -1 do 1. Problem saturacije i manjka detaljnosti je pristupan i na ovome sloju, međutim za razliku od sigmoid sloja, 0 se nalazi na sredini skali standardizacije.

ReLu sloj, rektificirana linearna jedinica, je postala popularna norma za nelinearnu aktivaciju u neuronskim mrežama. Aktivacijska funkcija je izražena u obliku $\boldsymbol{f(\kappa)=max\ (0,\kappa).}$ Drugim riječima, aktivacija ima određenu granicu stavljenu na nuli. ReLU je puno pouzdanija aktivacijska funkcija, te konvergira 6 puta brže nego druge dve navedene funckije. Međutim, najveća negativna strana ReLu aktivacije je da je osjetljiva prilikom treniranja same umjetne inteligencije, te korištenjem velikih gradijenata, može uzrokovati da se određeni neuroni ne aktiviraju daljnje, nakon inicijalne aktivacije.

2.2.3. Konvolucijska mreža - dizajn

Prilikom razumijevanja arhitekture konvolucijskih mreža, dobro je proći općenita pravila pri spajanju mreža i njihovom dizajniranju. Ovaj odijeljak služi kao analiza dizajna i dizajniranja poretka slojeva.
Jedan od poredaka koji možemo bi mogao biti kategoriziran kao

- Prvi skup
  - Konvolucijski sloj 1
  - Normalizacijski sloj 1
  - ReLu aktivacijski sloj 1
  - Pooling sloj 1
- Drugi skup
  - Konvolucijski sloj 2
  - Normalizacijski sloj 2
  - ReLu aktivacijski sloj
  - Pooling sloj 2
- Potpuno povezani sloj
- Izlaz

Ovakva konvolucijska mreža bi mogla biti opisana sljedećim kodom. Ovo je standarizirani primjer konvolucijske mreže, koji je postavio Google, te je ovakva mreža ukomponirana kao standard u TensorFlowu [citat]:

```
class convnet1(nn.Module):
    def __init__(self):
        super(convnet1, self).__init__()

        # Constraints for layer 1
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5,
stride = 1, padding=2)
        self.batch1 = nn.BatchNorm2d(16)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2) #default stride is equivalent
to the kernel_size

        # Constraints for layer 2
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5,
stride = 1, padding=2)
        self.batch2 = nn.BatchNorm2d(32)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2)

        # Defining the Linear layer
        self.fc = nn.Linear(32*7*7, 10)

    # defining the network flow
    def forward(self, x):
        # Conv 1
        out = self.conv1(x)
        out = self.batch1(out)
        out = self.relu1(out)

        # Max Pool 1
        out = self.pool1(out)

        # Conv 2
```

```
        out = self.conv2(out)
        out = self.batch2(out)
        out = self.relu2(out)

        # Max Pool 2
        out = self.pool2(out)

        out = out.view(out.size(0), -1)
        # Linear Layer
        out = self.fc(out)

return out
```

Kombinacijom konvolucijski slojeva sa slojevima normalizacije, smanjujemo vrijeme izračuna i težinske optimizacije. Nakon toga pooling funkcijom uzimamo maksimalne vrijednosti u matricama, te smanjujemo matricu, kako bi dodatno smanjili težinsku optimizaciju i vrijeme izračuna. Ponavljajući ovaj korak, smanjujemo sam broj neurona potrebnih za ovakav izračun, te se baze podataka s velikim brojem ulaznih podataka drastično brže izračunavaju.

Sada kada smo analizirali umjetnu inteligenciju i konvolucijske mreže, možemo preći na analizu Blendera programa.

2.3. Alat za 3D modeliranje Blender

Kao temeljni dio drugog koraka praktične primjene umjetne inteligencije, moramo analizirati Blender i parametrizacijski način rada, koji je nova karakteristika blendera.

Blender [3] je open-source alat za 3d modeliranje i animiranje baziran na poligonskom pristupu. Kao temeljni dio ovog rada, prije same analize navedenog programa, volio bi razjasniti zašto smo se odlučili za poligonski pristup, a ne vektorski.

U arhitekuri, svaka vizaulizacija se radi preko programa za modeliranje u vektorskom sustavu. Drugim riječima, svaka linija, ploha ili bilo koji kompleksniji model je sažet kao matrica različitih vektora. Neki do najpoznatijih programa temeljenih na vektorskom pristupu su SolidWorks, AutoCAD, Autodesk Revit i drugi.

Blender, koji ima opciju refaktoriranja poligonskih modela u vektorske modele, je program otvorenog koda. Zbog jake potpore zajednice, te značajnih programskih kontrivucija, Blender je postao jedna od vodećih sila u odabiru programa za modeliranje.

Glavni razlog odabira se svodi na činjenicu da, za razliku od navednih programa, Blender je dostupan bez ikakvih doplaćivanja. Programi navedeni, koji se smatraju kao standard u arhitekturalnom modeliranju, se plaćaju i nisu dostupni kao navedeni program.

2.3.1. Glavni prozor Blendera

Pri analizi Blendera [3][28], bitno je detaljno opisati glavne komponente koje su korištene u ovome radu. Jedan od glavnih komponenata je glavni prozor.

Glavni prozor se sastoji od nekoliko komponenata, međutim u sklopu ovog rada, fokusirat ćemo se samo na glavni prozor za modeliranje.

U ovom prozoru, možemo kreirati različite 3d objekte, koje prema potrebama možemo manipulirati i obrađivati kako bi dobili željeni rezultat.

Međutim, kako je svrha ovog rada parametarski i automatizirano proizvesti 3D model, koristit ćemo novu karakteristiku Blendera.

2.3.2. Modul za rad s geometrijskim čvorovima

Modul za rad s geometrijskih čvorovima (engl. geometry nodes) je dio Blendera koji je dodan u najnovijoj inačici programa.
Sam modul se temelji na čvornom, vizualnom programiranju, gdje imamo veliki broj sastavljenih skripti koje su prikazane pomoću čvorova. Spajanjem čvorova, dobivamo vizualni način programiranja, preko kojega možemo raditi kompleksne i parametarski dostupne vizualne skripte. Daljnja implementacija naše skripte će detaljno biti objašenjena u odjelku Rezultati.

Navedeni čvorovi su zapravo objekti koji sadržavaju sturkturirane podatke, te transofrmiraju ulaze u izlaze, temeljenim na parametrima definiranim u grupi čvorova, vizualiziranih kao stablo spojenih čvorova.

Jedan od prednosti ovog pristupa je samnjivanje repeticije modeliranje istih objekata, kako ih sada možemo parametarski generirarit bez poteškoće.

Sljedeći korak je detaljno objašnjenje svih čvorova koji Blender koristi i njihovu upotrebu.

2.3.3. Vrste čvorova u Blenderu

2.3.3.1. Instancirani čvorovi

Ovi čvorovi su čvorovi koji obrađuju podatke temeljeni na instancama danih objekata. Drugim riječima, kako bi mogli koristiti ove čvorove, moramo imati gnerirani objekt ili pomoću prijašnjh čvorova ili ručno modeliran. Instance su način kako bi se na brzi način mogla dodati ista geomtrija u 3d okolinu, bez dupliciranja samih poligona.

- Instance on Points čvor dodaje referencu na geometrijski objekt na sve točke koje su specifirani na ulazu.
- Rotate Instances okreće geometrijske instance u globalnom ili lokalnom mjestu.
- Instance to Points generira točke na izvornim točkama, kako bi dobili parent-child odnos
- Scale Instances skalira geometrijske instance u gloablnom ili lokalnom mjestu.
- Translate Instances premješta geometrijske instance u globalnom ili lokalnom okviru
- Realize Instances uzima bilo kakve instance i pretvara ih u prave geomtrijske objekte, ne samo instance. Drugim riječima primjena ovog čvora realizira objekte duplikata naše instance

2.3.3.2. Čvorovi vezani za materijale

Ovo su čvorovi koji se temelje na manipulaciji i obradi tekstura vezanih za naš objekt. Ovim čvorovima vizualno mijenjamo izgled našeg objekta.

- Replace Material Node mijenja jedan materijal s drugim. Mijenjanje materijala s čvorovima je više efikasno nego ručno mijenanje, jer se može primjeniti na broj većih instanca
- Material index node pokazuje koji materijal u listi materija korišteni u danoj ulaznoj geometriji se koristi.
- Material Selection Node daje uvid u koji dijelovi instanca, a ne cijele geometrije koriste dani materijal
- Set Material Node mijenja materijal koji je zadan danoj ulaznoj geometriji

2.3.3.3. Čvorovi koji se koriste za geometrijske mreže

Ovo su čvorovi koji se koriste za obradu i manipulaciju geomtrijskih mreža, to jest dijelova cijele geometrije. Neki od ovih čvorova su: - Extrude Mesh Node generira nove točke, bridove ili stranice, temeljeno na geometriji i načinu transformacije koji je zadan. - Mesh Boolean Node omogućuje nam da režemo ili spajamo geometriju preko Boolean modifikacija. - Edge Angle Node kalkulira kut u radijanima između dvije stranice koje se nalaze na istoj točki

Ovo su samo neki od najvžanijih čvorova koji se koriste. Blender trenutno sadrži više od 150 čvorova, te je većina koja nije opisana u radu primjenjena za svrhe izvan ovog rada.

3. Pregled dosad objavljenih radova

Kako bi mogli opisati motivaciju iza ovog diplomskog rada, te postaviti glavne temeljne teze, potrebno je opisati trenutna standardizirana istraživanja vezana za umjetnu inteligenciju i konvolucijske neuralne mreže.

3.1. StyleGAN

Rezolucija i kvaliteta slika rezultirana iz generativinih adversalnih mreža, rapidno se povećava tijekom zadnjih godina. StyleGAN je razvijen sa strane NVIDIA-e kako bi povećavo točnost i poboljšao arhitekturu neuralnih mreža koje se temelje na translaciji karakteristika između uređenih parova.

Općenito, generator koji se koristi u StyleGAN kreće od naučenih konstanti koje se koriste kao ulaz, te prema uređenom paru koji pokušava replicirat, mijenja navedene karakteristike u svakom koraku konovlucijskih slojeva. Dodatni korak kod StyleGAN-a je generiranje ulaznih slika u generatora kao šum dimenzije ulazne slike, mreža postaje automatizirana nenadgledani entititet koji razlikuje karakteristike i atribute uređenog para, te obavlja različite operacije nad mrežom.

Kod generatora, StyleGAN pridružuje uređeni par ulaznom šumu, te onda kroz adaptivnu normalizaciju instanci kontrolira sam rad, skalu i brzinu generatora pri svakoj epohi. Gauzijski šum je kasnije dodan, nakon svake konvolucije, prije nego što se promatra nelinearnost. Generator se sastoji od mreže za sintetiziranje i mreže za mapiranje. Opisana je mreža za sintezu slike, te se sastoji od 18 slojeva. Mreža za samo mapiranje slike se sastoji od 8 puno povezanih slojeva. Izlazni rezultat se sastoji od sloja koji konvertira zadnji sloj mreže za sintezu u 3 RGB kanala te koristi 1 x 1 konvoluciju.

StyleGAN ukupno, ima 26.2 milijuna parametara koji se mogu trenirati, dok tradicionalni generator prosječno ima 23 milijuna parametara.

Sama aktivacijska funkcija za adaptivnu normalizaciju instanci se definira kao:

$$\mathbf{AdaIN(xi, y) = ys,i\ xi - \mu(xi)/\sigma(xi) + yb,i}$$

Gdje svako mapiranje karakteristike xi je normalizirano individualno, te onda skalirano pomoću odgovarajuće skalarne komponente iz karakteristike y.

3.1.1. StyleGAN - kvaliteta generiranih slika

Temeljeno na radu koji je napisan, iznimno je jasno da tradicionalna GAN arhitektura inferiorno generira izlaze u

usporedbi s StyleGAN arhitekturom. Prema testiranju mjera i metrika unutar tradicionalnih GAN arhitektura, StyleGAN pokazuje superiornost prema sljedećim podacima:

3.1.2. StyleGAN - praktična primjena

Refernciranjem rada, možemo vidjet neke od glavnih primjena na kojima se koristio StyleGAN, kako bi daljnje pokazali inovativnu primjenu umjetne inteligencije i konvolucijskih mreža
Jedna od tih primjena je generacija realističnih ljudskih portreta, koji se diferenciraju po dobi i spolu, kao što je prikazano na sljedećoj slici.

Slika 1 Primjena StyleGAN arhitekture za generiranje realističnih portreta

3.2. CUT

CUT[1] metoda, je metoda koja se temelji na translaciji karakteristika pri neuređenim parovima. To znači da želimo slike iz domene X □ RH×W ×C prikazati što sličnijima prema karakteristikama iz domene Y □ RH×W ×3. Dobivamo skup podataka neuparenih instanca X = {x □ X }, Y = {y □ Y}.
Ovo je poseban slučaj, zato jer ve metode, uključujuči i našu koju koristimo, u umjetnoj inteligenciji se temelje na translaciji karakteristika uređenih parova. Drugim riječima, setovi slika koji se translatiraju su unaprijed određeni.

Ova metoda zahtijeva samo mapiranje u jednom smjeru, za razliku od uređenih parova, gdje se translacija karakteristika vrši u oba smjera.

Kao glavnu mjeru točnosti, koristi se adversalni gubitak, koji se mjeri logaritamskom funkcijom:

$$\mathbf{LGAN(G, D, X, Y) = Ey□Y \log D(y) + Ex□X \log(1 − D(G(x))).}$$

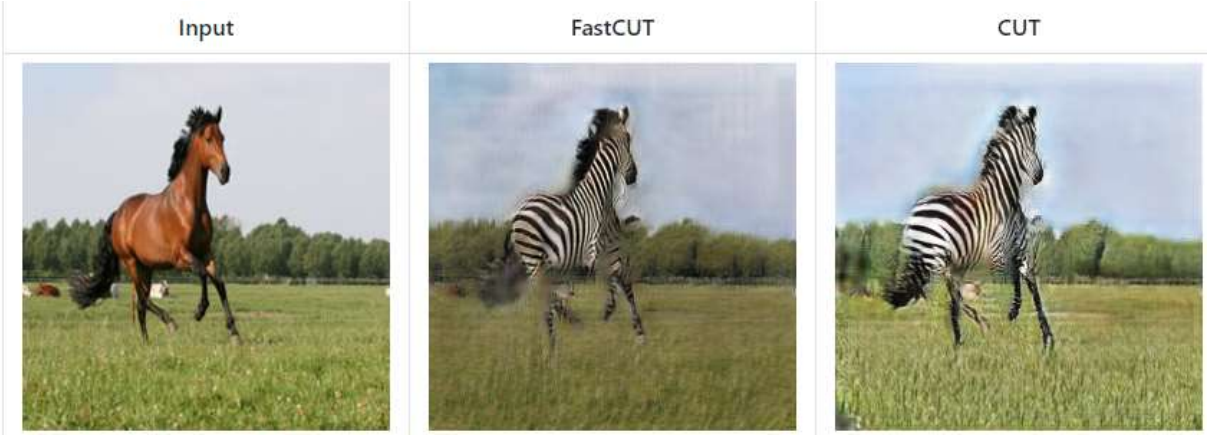
U ovoj se metodi također koristi šum za kontrastno određivanje okvirnih slojeva kako bi se maksimizirale zajedničke informacije između ulaza i izlaza.

Ova metoda koristi PATCHGan arhitekturu, o kojoj se detaljnija analiza provodi kasnije u radu.

Glavni cilj ove metode je generirati i obraditi slike koje su realistične, dok se zadržavaju karakteristike ulazne slike na izlazu. Što se tiče samih mjerenja vremena izvođenja, metoda je najbrža prema statistici od 3 navedene ovdje.

3.2.1. CUT - Rezultati

Rezultati su navedeni u obliku slike kao refernca iz navedenog znanstvenog rada. Vidimo da se uspoređuju različite metode neuređenog para, te je CUT puno precizniji od navedenih metoda. Translacija karakteristika je puno detaljnija, te se zbog ispravne mjere točnosti, mogu dobiti bolji rezultati



Slika 2 CUT Generiranje slike konja u sliku konja s karakteristikama zebre

3.3. PIX2PIX

Pix2Pix [2] je detaljno opisan u nastavku rada kako se koristi u svrhu generiranj digitalnih tlocrta, kao primjer praktične koristi.
Cilj kondicijskog GAN-a se može izraziti formulom:

$$\mathbf{LcGAN\ (G, D) = E_{x,y}\ [\log D(x, y)] + E_{x,z}\ [\log(1 - D(x, G(x, z))],}$$

gdje G želi minimizirati ovu formulaciju, a D želi maksimizirati

G predstavlja generator, dok je D naš diskriminator.

Međutim, kod tradicionalnih GAN tehnika, koristila se mješavina maksimizacije GAN točnosti s tradicionalnim mjerama točnosti, kao što je MSE.

Pix2Pix se diferencira u ovom području, te pokušava, pomoću L1 udaljenosti, odrediti mjeru točnosti:

$$\mathbf{L_{L1}(G) = E_{x,y,z}\ [\|y - G(x, z)\|_1].}$$

Formulacija našeg modela se onda svodi na:

$$\mathbf{G^{*} = \arg\min_G \max_D L_{cGAN}(G, D) + \lambda L_{L1}(G).}$$

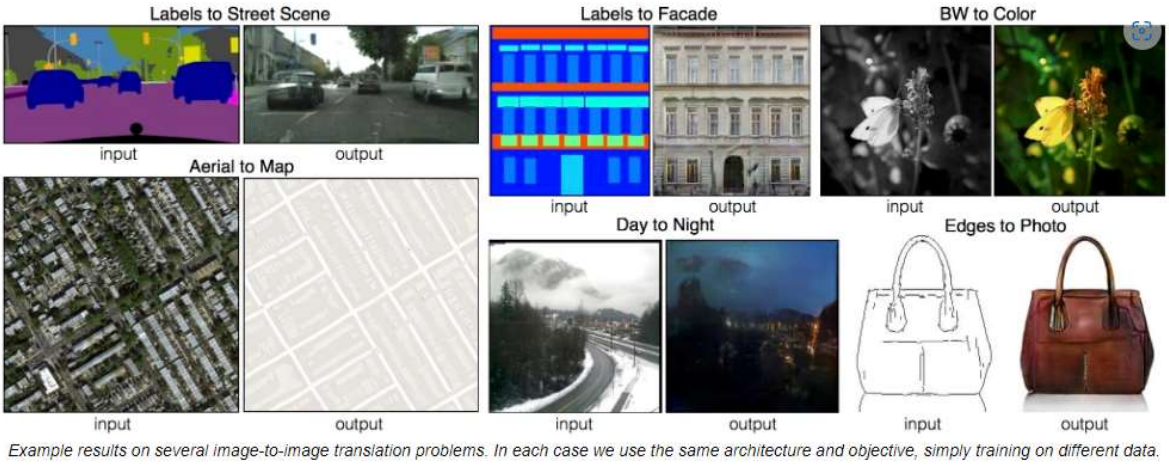
Daljnja analiza i diskusija Pix2Pix algoritma će biti detaljno analizirana kasnije u radu.

3.3.1. Rezultati - Pix2Pix

Ponovo su predstavljeni rezultati Pix2Pix arhitekture, te se vidi, da iako funkcionira na uređenim parovima, izgledi vizualno izgledaju kao realistične slike.
#slika

Neki od glavnih indikatora ročnosti ovog algoritma su znastvene studije provedene u sklopu referentnog rada.

AMT perceptualne studije, gdje su sudionici predstavljeni set slika, od kojih su neke generirane umjetnom inteligencijom, a neke su izvorne. Za svaki set slika, sudionici su vidjeli slike po 1 sekundu, te su onda anvedene slike estale. Rezultati su onda sumirani u tablicu i prezentirani:



Slika 3 Pix2Pix translacija karakteristika

3.4.

3.5. UMJETNA INTELIGENCIJA

Umjetna inteligencija je postala jedna od najbrže rastućih grana informatike zadnjih nekoliko godina. Sa samim ciljem definitivne automatizacije informatičkog sektora, kao i korak prema razvoju svijesti računalnih jedinica, potencijal i privlačnost ovakve grane informatike krije veliki potencijal.
U ovom radu spajamo neuralne mreže kao komponentu umjetne inteligencije i komponentu arhitekture kako bi jasno pokazali mogućnosti ovakvog pristupa.

Kako bi mogli analizirati i objasniti svaki korak navedenog rada, moramo objasniti osnovne temelje umjetne inteligencije.

U navedenom radu koristimo mrežu neuralnih mreža, jedna koja se koristi za izrađivanje slika tlocrta, te jednu koju koristimo kako bi mogli segregirati napravljene i izvorne slike. Ove dvije neuralne mreže se konstantno razvijaju, ovisna jedna o drugoj, kroz veliki broj koraka.

Ovakav model se zove General Adversarial Networks(GAN), a sastavljen je od dve nadeven komponente: generatora i diskriminatora.

Indikacija ovog eksperimenta je dosta pozitivno percepirana za budućnost umjetne inteligencije. Također,

proveden je test kolorizacije, koji je prikazan ispod:

#slika

Ovim testovima, pokazano je da Pix2Pix model, generira slike realistično, te traži i identificira karakteristike uređenih parova na vrlo precizan način.

3.6. CycleGAN

Kako bi razumjeli prvi dio rada, moramo objasniti kakav model koristimo. CycleGAN. Radi se treniranju modela uređenih parova kako bi efektivno mogli prenijeti karakteristike jedne slike u drugu. Na ovaj način karakteristike nezavisne slike xi pokušavamo prenijeti zavisni odgovarajući par yi. Ovaj problem takozvan “image to image translation”, riješen je u odgovarajućem radu kojim se razvija model koji radi upravo to. ## Generator Neuralne mreže sastavljene s ciljem da naprave kandidate slika koji sliče izvornom materijalu kako bi mogle zavarati bilo kakvu distinkciju između stvarne i nastale slike se zovu generatori.
Generatori u našem kontekstu, to jest generatori koji se koriste u CycleGAN modelu imaju nekoliko glavnih slojeva koje moramo objasniti prije nego što ih možemo analizirati[32]

Model generatora baziran je na obradi slike nasumičnog zvuka. Nasumični zvuk postepenim koracima u iteracijama se onda obrađuje po bazi podataka i podacima iz samog diskirminatora kako bi se vizualizirala obrada i kopija freferentnog materijala

Prvi sloj je Dense formata. Ulaz je latentni 1-dimenzionalni prostor kojeg onda uz drugi sloj reshape-a mijenjamo u rzaličite blokove. Nakon toga transponiramo sada 2 dimenzionalno polje kroz odredeđeni broj filtera i dobivamo novo 2-dimenzionalno polje, koje uz isti sloj, samo transponiran, hranimo u konvolucijski sloj kako bi mogli dobiti obrađenu sliku. Ovo je grubi redoslijed slojeva koje se koriste kako bi mogli uspostaviti osnovni generator.

Međutim, kako su naši tesnti skupovi kompleksniji od jednostavne upotrebe CycleGAN-a, proći ćemo kroz glavni kod generatora kako bi mogli objasnit točno kako koristimo ovaj princip u našem eksperimentu

```
def define_G(input_nc, output_nc, ngf, netG, norm='batch', use_dropout=False,
init_type='normal', init_gain=0.02, gpu_ids=[]):
    """Create a generator
    Parameters:
        input_nc (int) -- the number of channels in input images
        output_nc (int) -- the number of channels in output images
        ngf (int) -- the number of filters in the last conv layer
        netG (str) -- the architecture's name: resnet_9blocks | resnet_6blocks |
unet_256 | unet_128
        norm (str) -- the name of normalization layers used in the network:
batch | instance | none
        use_dropout (bool) -- if use dropout layers.
        init_type (str)    -- the name of our initialization method.
        init_gain (float)  -- scaling factor for normal, xavier and orthogonal.
        gpu_ids (int list) -- which GPUs the network runs on: e.g., 0,1,2
    Returns a generator
    Our current implementation provides two types of generators:
        U-Net: [unet_128] (for 128x128 input images) and [unet_256] (for 256x256
input images)
        The original U-Net paper: https://arxiv.org/abs/1505.04597
        Resnet-based generator: [resnet_6blocks] (with 6 Resnet blocks) and
[resnet_9blocks] (with 9 Resnet blocks)
        Resnet-based generator consists of several Resnet blocks between a few
downsampling/upsampling operations.
```

```
        We adapt Torch code from Justin Johnson's neural style transfer project
(https://github.com/jcjohnson/fast-neural-style).
    The generator has been initialized by <init_net>. It uses RELU for non-
linearity.
    """
    net = None
    norm_layer = get_norm_layer(norm_type=norm)

    if netG == 'resnet_9blocks':
        net = ResnetGenerator(input_nc, output_nc, ngf, norm_layer=norm_layer,
use_dropout=use_dropout, n_blocks=9)
    elif netG == 'resnet_6blocks':
        net = ResnetGenerator(input_nc, output_nc, ngf, norm_layer=norm_layer,
use_dropout=use_dropout, n_blocks=6)
    elif netG == 'unet_128':
        net = UnetGenerator(input_nc, output_nc, 7, ngf, norm_layer=norm_layer,
use_dropout=use_dropout)
    elif netG == 'unet_256':
        net = UnetGenerator(input_nc, output_nc, 8, ngf, norm_layer=norm_layer,
use_dropout=use_dropout)
    else:
        raise NotImplementedError('Generator model name [%s] is not
recognized' % netG)
    return init_net(net, init_type, init_gain, gpu_ids)
```

U navedenom kodu se nalazi funckija koja radi generator koji koristimo u svim našim eksperimentima. Ulaz funkcije `define_g` se sastoji od nekoliko parametra koje moramo objasniti i analizirati prije nego što možemo objasniti kod.

Prvi parametar je `input_nc`. Navedeni parametar služi broj kanala u slikama koje koristimo kao ulaz u naš generator. Kako koristimo RGB slike imamo 3 kanala, te je ovaj parametar postavljen na broj 3. Da koristimo slike koje imaju samo crnu i bijelo boju, grayscale ili bilo kakav drugačiji format, morali bi mijenjati broj kanala. Razlog ovome je da su svi pikseli spremljeni kao n-dimenzionalna polja, te svaki n predstavlja kanal. Na ovaj način svaki red je spremljen kao polje od n dimenzionalnih polja.

Drugi parametar je `output_nc`. Navedeni parametar služi broj kanala u slikama koje koristimo kao izlaz u naš generator. Ovaj konkretan parametar nam služi da možemo definirati koliko kanala ima naša izlazna slika, koja će se koristiti kao generirana slika temeljena na izvornim parovima.

Sljedeći parametar koji moramo objasniti je `ngf` je broj filtera u zadnjem konvolucijskom sloju, te se koristi kako bi mogli lakše prepoznati same akrakteristike regije koju promatramo.

`norm` služi kao specifikacija normalizacije koje ćemo koristiti u našoj neuralnoj mreži generatora. Normalizacija je proces skaliranja veličine podataka na istu skalu, kako bi karakteristike naših slika mogle lakše vidljivu korelaciju međusobno. Proces normalizacije pomaže u dizanju točnosti samog modela, te ubrzava vrijeme izvršavanja modela.

`use_dropout` koristimo kako bi signalizirali modelu da koristi takozvane dropout slojeve, koji temeljno ređu dio podataka iz regije koje gledamo, kako bi minimalizirali broj piksela, i ugrubo klasificirali grupe piksela ili srezali nekoliko kanala karakteristike kada ekspanzivno gledamo mrežu. Ova metoda se može primjeniti kako bi mogli ubrzati sam rad mreže jer efektivno odbacujemo dio regije koje se mora analizirati i sa svakim slojem radimo s manje piksela u regiji.

`init_type` - Inicijalizacijska metoda, moramo je specificirati ako je koristimo te određujemo `init_gain` koji skalira samu inicijalizacijsku metodu.

Zadnji parametar je `net_G` te je ovo najbitniji parametar koji koristimo pri samom generaciji jer određuje vrstu neuralne mreže koju će generator adaptirati kako bi mogao obraditi ulazne komponente i generirati izlazne komponente. Postoje dvije vrste mreža koje su nam na raspolaganju, te ćemo ih u sljedećem dijelu analizirati.

3.6.1. UNet mreže

UNet[23] mreže su dobile naziv prema arhitekturi konvolucijskih i pooling mreža koji izgleda kao U, prikazano na slici ispod.

U-net arhitektura minimizira vrijeme treniranja setova podataka koji maksimizira efektivnost hvatanja njabitnijih karakteristika i konteksta svih piksela slike kako bi reducirao vrijeme klasifikacije i treniranja individualnih podataka. U CycleGAN arhitekturi, koriste se UNet mreže za slike 128 i 512 piksela klasificirani kao `Unet_512` i `Unet_128`

Unet mreža ima dve glavne karakteristike: kontrakcijski put i ekpanzijski put. Kontracijski put služi za kontrakciju samog ulaza, te koristi konvolucijske slojeve s ReLU aktivacijskim slojevima kako bi ulaznu regiju uzorka mogao smanjivati(downsampling). Svakim korakom smanjivanjem uzorka , udvostručavamo broj kanala vezanih za karakteristike posebne za regiju koja se obrađuje. Drugim riječima, samnjujemo veličinu uzorka, ali spremamo karakteristike navedenih uzoraka u navedene kanale. Pri kraju kontrakcijskog procesa dobivamo smanjenju regiju uzorka s velikim brojem kanala vezanih karakteristika. Nakon kontrakcije krećemo sa ekspanzijom uzorka, inverzni proces smanjivanja uzorkovanja. Svakim korakom ekspanzije povečamao regiju uzorka i umanjujemo broj kanala karakteristika za dva puta. Nakon svake konvolucije režemo granicu uzorka, zbog gubitka ghraničnih piksela, zbog same ekspanzivne konvolucije koje radimo kroz svaki sloj. Na kraju ekspanzivnog procesa mapiramo svako 128/512 pikselnu komponentu u regiji na broj klasa kojima kalsificiramo. Drugim riječima, kroz samu kompresiju i retrakciju svake regije slike, model klasificira svaku regija individualno po navedenim karakteristikama, te može generirati regije slike po klasifikacijama i kalkucijama koje izračuna. Na ovaj način Unet model dobiva klasifikaciju karakteristika i generira svaku regiju slike našeg tlocrta po 128x128 pikseliziranim regijama

Modeli Uneta koje možemo koristiti generalnu s regije od 128x128 piksela i 512x512 piksela.

Generiranje UNet mreže se odvija u sljedećoj klasi koda, kao što je objašnjeno u navedenom opisu iznad:

```
        class UnetGenerator(nn.Module):
            """Create a Unet-based generator"""

            def __init__(self, input_nc, output_nc, num_downs, ngf=64,
norm_layer=nn.BatchNorm2d, use_dropout=False):
                """Construct a Unet generator
                Parameters:
                    input_nc (int)  -- the number of channels in input images
                    output_nc (int) -- the number of channels in output images
                    num_downs (int) -- the number of downsamplings in UNet. For
example, # if |num_downs| == 7,
                                        image of size 128x128 will become of
size 1x1 # at the bottleneck
                    ngf (int)       -- the number of filters in the last conv
layer
                    norm_layer      -- normalization layer
                We construct the U-Net from the innermost layer to the outermost
```

```
layer.
                It is a recursive process.
                """
                super(UnetGenerator, self).__init__()
                # construct unet structure
                unet_block = UnetSkipConnectionBlock(ngf * 8, ngf * 8,
input_nc=None, submodule=None, norm_layer=norm_layer, innermost=True)  # add the
innermost layer
                for i in range(num_downs - 5):          # add intermediate
layers with ngf * 8 filters
                    unet_block = UnetSkipConnectionBlock(ngf * 8, ngf * 8,
input_nc=None, submodule=unet_block, norm_layer=norm_layer,
use_dropout=use_dropout)
                # gradually reduce the number of filters from ngf * 8 to ngf
                unet_block = UnetSkipConnectionBlock(ngf * 4, ngf * 8,
input_nc=None, submodule=unet_block, norm_layer=norm_layer)
                unet_block = UnetSkipConnectionBlock(ngf * 2, ngf * 4,
input_nc=None, submodule=unet_block, norm_layer=norm_layer)
                unet_block = UnetSkipConnectionBlock(ngf, ngf * 2,
input_nc=None, submodule=unet_block, norm_layer=norm_layer)
                self.model = UnetSkipConnectionBlock(output_nc, ngf,
input_nc=input_nc, submodule=unet_block, outermost=True, norm_layer=norm_layer)
# add the outermost layer

            def forward(self, input):
                """Standard forward"""
                return self.model(input)


        class UnetSkipConnectionBlock(nn.Module):
            """Defines the Unet submodule with skip connection.
                X -------------------identity----------------------
                |-- downsampling -- |submodule| -- upsampling --|
            """

            def __init__(self, outer_nc, inner_nc, input_nc=None,
                         submodule=None, outermost=False, innermost=False,
norm_layer=nn.BatchNorm2d, use_dropout=False):
                """Construct a Unet submodule with skip connections.
                Parameters:
                    outer_nc (int) -- the number of filters in the outer conv
layer
                    inner_nc (int) -- the number of filters in the inner conv
layer
                    input_nc (int) -- the number of channels in input
images/features
                    submodule (UnetSkipConnectionBlock) -- previously defined
submodules
                    outermost (bool)    -- if this module is the outermost
module
                    innermost (bool)    -- if this module is the innermost
module
                    norm_layer          -- normalization layer
                    use_dropout (bool)  -- if use dropout layers.
                """
                super(UnetSkipConnectionBlock, self).__init__()
                self.outermost = outermost
                if type(norm_layer) == functools.partial:
                    use_bias = norm_layer.func == nn.InstanceNorm2d
                else:
                    use_bias = norm_layer == nn.InstanceNorm2d
                if input_nc is None:
                    input_nc = outer_nc
                downconv = nn.Conv2d(input_nc, inner_nc, kernel_size=4,
                                     stride=2, padding=1, bias=use_bias)
                downrelu = nn.LeakyReLU(0.2, True)
```

```
        downnorm = norm_layer(inner_nc)
        uprelu = nn.ReLU(True)
        upnorm = norm_layer(outer_nc)

        if outermost:
            upconv = nn.ConvTranspose2d(inner_nc * 2, outer_nc,
                                        kernel_size=4, stride=2,
                                        padding=1)
            down = [downconv]
            up = [uprelu, upconv, nn.Tanh()]
            model = down + [submodule] + up
        elif innermost:
            upconv = nn.ConvTranspose2d(inner_nc, outer_nc,
                                        kernel_size=4, stride=2,
                                        padding=1, bias=use_bias)
            down = [downrelu, downconv]
            up = [uprelu, upconv, upnorm]
            model = down + up
        else:
            upconv = nn.ConvTranspose2d(inner_nc * 2, outer_nc,
                                        kernel_size=4, stride=2,
                                        padding=1, bias=use_bias)
            down = [downrelu, downconv, downnorm]
            up = [uprelu, upconv, upnorm]

            if use_dropout:
                model = down + [submodule] + up + [nn.Dropout(0.5)]
            else:
                model = down + [submodule] + up

        self.model = nn.Sequential(*model)

    def forward(self, x):
        if self.outermost:
            return self.model(x)
        else:   # add skip connections
            return torch.cat([x, self.model(x)], 1)
```

3.6.2. Mreža ResNet

Glavna distinkcija ResNet[24] mreža preko UNet mreža je što u ResNet mrežama svaki sloj je spojen na sljedeći sloj u mreži, ali također i na svaki n-ti, najviše je n=2. To Znači da . Ideja iza ResNet mreža je da u svakoj iteraciji slojeva povećavamo broj slojeva tako da ukomponiramo prijašnje slojeve mreže, te dignemo točnost traženja samih karakteristika u svakoj individualnoj slici. Svaki rezidualni blok, to jest sloj, pokušava naučiti glavni izlaz H(x), međutim kako imamo rezidualnu komponentu x, zapravo model pokušava naučiti originalni identitet, to jest R(x)

$$\mathbf{H(x) = R(x) + x}$$

te zbog činjenice da sam model pokušava identificirati samu rezidualnu komponentu u svakom bloku, ima odgovrajuće ime.

Rezdiualne mreže pokušavaju naučiti rezidualnu komponentu x samog ulaza, te se ne fokusiraju na učenje rezdiualu izlaza.

Također, ako treniramo mrežu da nam je rezidual na nuli, možemo naučiti sami identitet mreže, to jest H(x), što znači da se rezidual može lakše identicifirati i odrediti, te ovaj postupak dovodi do bržeg vremena treniranja, kao i bolje točnosti same konvolucijske mreže

U našem CycleGAN sklopu, možemo koristiti rezidualne mreže od 6 ili 9 blokova, što znači da će svaka dva koraka ispred mreža bit spojena s rezidualnom komponentom od 6 ili 9 blokova.

Generiranje Rezidualne mreže je objašnjeno u sljedećem kodu:

```
        class ResnetGenerator(nn.Module):
           Resnet-based generator that consists of Resnet blocks between a few
downsampling/upsampling operations.
            We adapt Torch code and idea from Justin Johnson's neural style
transfer project(https://github.com/jcjohnson/fast-neural-style)



        def __init__(self, input_nc, output_nc, ngf=64,
norm_layer=nn.BatchNorm2d, use_dropout=False, n_blocks=6,
padding_type='reflect'):
            """Construct a Resnet-based generator
            Parameters:
                input_nc (int)      -- the number of channels in input images
                output_nc (int)     -- the number of channels in output images
                ngf (int)           -- the number of filters in the last conv
layer
                norm_layer          -- normalization layer
                use_dropout (bool)  -- if use dropout layers
                n_blocks (int)      -- the number of ResNet blocks
                padding_type (str)  -- the name of padding layer in conv layers:
reflect | replicate | zero
            """
            assert(n_blocks >= 0)
            super(ResnetGenerator, self).__init__()
            if type(norm_layer) == functools.partial:
                use_bias = norm_layer.func == nn.InstanceNorm2d
            else:
                use_bias = norm_layer == nn.InstanceNorm2d

            model = [nn.ReflectionPad2d(3),
                     nn.Conv2d(input_nc, ngf, kernel_size=7, padding=0,
bias=use_bias),
                     norm_layer(ngf),
                     nn.ReLU(True)]

            n_downsampling = 2
            for i in range(n_downsampling):  # add downsampling layers
                mult = 2 ** i
                model += [nn.Conv2d(ngf * mult, ngf * mult * 2, kernel_size=3,
stride=2, padding=1, bias=use_bias),
                          norm_layer(ngf * mult * 2),
                          nn.ReLU(True)]

            mult = 2 ** n_downsampling
            for i in range(n_blocks):       # add ResNet blocks

                model += [ResnetBlock(ngf * mult, padding_type=padding_type,
norm_layer=norm_layer, use_dropout=use_dropout, use_bias=use_bias)]

            for i in range(n_downsampling):  # add upsampling layers
                mult = 2 ** (n_downsampling - i)
                model += [nn.ConvTranspose2d(ngf * mult, int(ngf * mult / 2),
                                             kernel_size=3, stride=2,
                                             padding=1, output_padding=1,
                                             bias=use_bias),
                          norm_layer(int(ngf * mult / 2)),
                          nn.ReLU(True)]
            model += [nn.ReflectionPad2d(3)]
            model += [nn.Conv2d(ngf, output_nc, kernel_size=7, padding=0)]
```

```
            model += [nn.Tanh()]

            self.model = nn.Sequential(*model)

        def forward(self, input):
            """Standard forward"""
            return self.model(input)
```

3.6.3. Biranje tipa mreže - Generator

Sama funkcija generiranja rezidualne ili UNet mreže je prikazana kao navedeno:

```
        if netG == 'resnet_9blocks':
            net = ResnetGenerator(input_nc, output_nc, ngf,
norm_layer=norm_layer, use_dropout=use_dropout, n_blocks=9)
        elif netG == 'resnet_6blocks':
            net = ResnetGenerator(input_nc, output_nc, ngf,
norm_layer=norm_layer, use_dropout=use_dropout, n_blocks=6)
        elif netG == 'unet_128':
            net = UnetGenerator(input_nc, output_nc, 7, ngf,
norm_layer=norm_layer, use_dropout=use_dropout)
        elif netG == 'unet_256':
            net = UnetGenerator(input_nc, output_nc, 8, ngf,
norm_layer=norm_layer, use_dropout=use_dropout)
        else:
            raise NotImplementedError('Generator model name [%s] is not
recognized' % netG)
```

U kodu iznad, možemo distinktno vidjet, da ovisno o parametru koji uvedemo u mrežu, generator će biti ili UNet ili ResNet mreža. Detaljnije o parametrima i generatoru koji koristimo će biti objašnjeno u rezultatima.
Nakon samog objašnjena Generatora, moramo objasniti kako funkckionira sam Diskriminator i njegovu ulogu, te objasniti kod povezan uz njega, te onda pokazati sam CycleGAN model i provjeriti rezultate.[31]

3.7. Diskriminator

Diskriminator je inverzni model samog generacijskog modela i njegova distinknta funkcija u modelu je preopznati koje slike su izvorne, a koje su generirane preko generatora.
Nekoliko glavnih karakteristika mora biti zadovoljeno kako bi naš model imao apsolutnu točnost. Uređeni parovi moraju imati korelaciju, to jest slike moraju biti slične po naravi, što znači da par mora imati zajedničke komponente kako bi se nezavisne i ne postojeće komponente u zavisnom paru mogle translatirati.[31]

U našem kontekstu, translatiramo nekoliko instanci ovog primjera. Na prvom sloju, prenosimo raspored soba u tlocrtu na ocrtani sami tlocrt. U drugom kontekstu, prenosimo ocrtavanje samih rasporeda soba u nacrtanu skicu samog tlocrta sa namještajem.

Kako bi mogli objasniti samu funkcionalnost diskrimnatora, objasnit ćemo i analizirati kod generiranja samog diskriminatora u nastavku:

```
        def define_D(input_nc, ndf, netD, n_layers_D=3, norm='batch',
init_type='normal', init_gain=0.02, gpu_ids=[]):
            """Create a discriminator
            Parameters:
                input_nc (int)     -- the number of channels in input images
                ndf (int)          -- the number of filters in the first conv
layer
                netD (str)         -- the architecture's name: basic | n_layers
```

```
| pixel
                n_layers_D (int)   -- the number of conv layers in the
discriminator; effective when netD=='n_layers'
                norm (str)         -- the type of normalization layers used in
the network.
                init_type (str)    -- the name of the initialization method.
                init_gain (float)  -- scaling factor for normal, xavier and
orthogonal.
                gpu_ids (int list) -- which GPUs the network runs on: e.g.,
0,1,2
            Returns a discriminator
            Our current implementation provides three types of discriminators:
                [basic]: 'PatchGAN' classifier described in the original pix2pix
paper.
                It can classify whether 70×70 overlapping patches are real or
fake.
                Such a patch-level discriminator architecture has fewer
parameters
                than a full-image discriminator and can work on arbitrarily-
sized images
                in a fully convolutional fashion.
                [n_layers]: With this mode, you can specify the number of conv
layers in the discriminator
                with the parameter <n_layers_D> (default=3 as used in [basic]
(PatchGAN).)
                [pixel]: 1x1 PixelGAN discriminator can classify whether a pixel
is real or not.
                It encourages greater color diversity but has no effect on
spatial statistics.
            The discriminator has been initialized by <init_net>. It uses Leakly
RELU for non-linearity.
            """
            net = None
            norm_layer = get_norm_layer(norm_type=norm)

            if netD == 'basic':  # default PatchGAN classifier
                net = NLayerDiscriminator(input_nc, ndf, n_layers=3,
norm_layer=norm_layer)
            elif netD == 'n_layers':  # more options
                net = NLayerDiscriminator(input_nc, ndf, n_layers_D,
norm_layer=norm_layer)
            elif netD == 'pixel':     # classify if each pixel is real or fake
                net = PixelDiscriminator(input_nc, ndf, norm_layer=norm_layer)
            else:
                raise NotImplementedError('Discriminator model name [%s] is not
recognized' % netD)
            return init_net(net, init_type, init_gain, gpu_ids)
```

Diskriminator uzima iste funkcijske parametre kao generator, tako da njihova objašnjenja nećemo daljnje analizirati. Međutim, glavna distinkcija diskriminatora i generatora je mrežna arhitektura, koju biramo sa sljedećim parametrom `netD`.

`netD` ima nekoliko različitih arhitektura pa ćemo u sljedećem odlomku analizirati mogućnosti samih arhitektura navedenog diskriminatora.

3.7.1. Arhitekture mreže diskriminatora

3.7.1.1. PATCHGan

Kako bi mogli razumjeti `basic` arhitekturu diskriminatora, moramo se podsjetiti na funkcionalnost diskrimanitcnih mreža u problamtici translacije karakteristika u slikama. Normalna konvolucijska diskriminatorska mreža uzima ulaz X te daje izlaz veličine cijele slike kao format vjerojatnosti da navedeni ulaz

spada u klasifikaju odgovarajučeg skalarnog vektora. Drugim riječima, ako slika pripada izvornoj referenci, ili je slika umjetno generirana.[29]
PATCHGan mreže se razlikuju po izlazu, umjesto skalarnog vektora, izlaz je oblikovan u formatu NxN vektora. NxN ovisi o ulaznoj veličini slike, međutim svaki izlaz vektora vjerojatnosti za nxn odlomke slike, za razliku od normalne konvolucijske mreže, koja daje vjerojatnost za cijelu sliku.

Ovisno o broju filtera, veličini kernela i samom paddingu, PATCHGan mreža downsamplea sami ulaz, kako bi mogli dobiti određeni odlomak navedene slike koju analiziramo.

Kao primjer možemo uzeti ulaz kao sliku dimenzija 256x256, te očekivani izlaz odlomak dimenzija 70x70. PRva tri sloja, C1,C2 i C3 bi dvostruko smanjivali broj uzoraka i filtera, te same dimenzije slike, te bi C4 maknuo granične piksele. Rezultat bi bio odlomak od 30x30 piksela, te svaka regija bi se koncipirala od 70x70 regija piksela.

Ovakav diskriminator modelira same slike kao nasumično Markov polje, neusmjereni graf, gdje svaki N čvor ima nasumičnu vrijednost. Na ovaj način, pretpostavljamo samostalnost piksela koji su ograđeni u različitim regijama, te nam se točnost samog modela diže ekvivalentno

```
        class NLayerDiscriminator(nn.Module):
            """Defines a PatchGAN discriminator"""

            def __init__(self, input_nc, ndf=64, n_layers=3,
norm_layer=nn.BatchNorm2d):
                """Construct a PatchGAN discriminator
                Parameters:
                    input_nc (int)  -- the number of channels in input images
                    ndf (int)       -- the number of filters in the last conv
layer
                    n_layers (int)  -- the number of conv layers in the
discriminator
                    norm_layer      -- normalization layer
                """
                super(NLayerDiscriminator, self).__init__()
                if type(norm_layer) == functools.partial:  # no need to use bias
as BatchNorm2d has affine parameters
                    use_bias = norm_layer.func == nn.InstanceNorm2d
                else:
                    use_bias = norm_layer == nn.InstanceNorm2d

                kw = 4
                padw = 1
                sequence = [nn.Conv2d(input_nc, ndf, kernel_size=kw, stride=2,
padding=padw), nn.LeakyReLU(0.2, True)]
                nf_mult = 1
                nf_mult_prev = 1
                for n in range(1, n_layers):  # gradually increase the number of
filters
                    nf_mult_prev = nf_mult
                    nf_mult = min(2 ** n, 8)
                    sequence += [
                        nn.Conv2d(ndf * nf_mult_prev, ndf * nf_mult,
kernel_size=kw, stride=2, padding=padw, bias=use_bias),
                        norm_layer(ndf * nf_mult),
                        nn.LeakyReLU(0.2, True)
                    ]

                nf_mult_prev = nf_mult
                nf_mult = min(2 ** n_layers, 8)
                sequence += [
```

```
                nn.Conv2d(ndf * nf_mult_prev, ndf * nf_mult, kernel_size=kw,
stride=1, padding=padw, bias=use_bias),
                norm_layer(ndf * nf_mult),
                nn.LeakyReLU(0.2, True)
            ]

            sequence += [nn.Conv2d(ndf * nf_mult, 1, kernel_size=kw,
stride=1, padding=padw)]  # output 1 channel prediction map
            self.model = nn.Sequential(*sequence)

        def forward(self, input):
            """Standard forward."""
            return self.model(input)
```

3.7.1.2. N_Layers

`n_layers` parametar instancira normalnu konvolucijsku mrežu s brojem definiranih slojeva. Ovo je najednostavniji pristup diskriminatoru, te nam je izlaz definiran kao skalarni vektor s mogućnostima navedene slike da bude izvorna ili umjetna. Isti kod se koristi za PATCHGan arhitekturu i konovlucijsku arhitekturu koji je naveden gore.

3.7.1.3. Pixel Diskriminator

Zadnja arhitektura same mreže koju možemo koristiti je `n_pixels` parametar koji se umjesto slike kao cijeline temelji na prepoznavanje svakog individualnog piksela i njegove izvornosti. Drugim riječima, ova arhitektura pokušava odrediti ako je piksel iz izvornog materijala ili je umjetno generiran.
Sama arhitektura je bazirana na konvolucijskog skevencijskog mreži kao i u ostalim diskriminatorskim mrežama, samo uzimamo piksel po piksel svake slike kao ulaz, umjesto cijele slike u prethodna dva slučaja. Izlaz ove mreže je skalarna vjerojatnost ako je svaki piksel stvaran ili umjetna.

```
    class PixelDiscriminator(nn.Module):
        """Defines a 1x1 PatchGAN discriminator (pixelGAN)"""

        def __init__(self, input_nc, ndf=64, norm_layer=nn.BatchNorm2d):
            """Construct a 1x1 PatchGAN discriminator
            Parameters:
                input_nc (int)  -- the number of channels in input images
                ndf (int)       -- the number of filters in the last conv
layer
                norm_layer      -- normalization layer
            """
            super(PixelDiscriminator, self).__init__()
            if type(norm_layer) == functools.partial:  # no need to use bias
as BatchNorm2d has affine parameters
                use_bias = norm_layer.func == nn.InstanceNorm2d
            else:
                use_bias = norm_layer == nn.InstanceNorm2d

            self.net = [
                nn.Conv2d(input_nc, ndf, kernel_size=1, stride=1,
padding=0),
                nn.LeakyReLU(0.2, True),
                nn.Conv2d(ndf, ndf * 2, kernel_size=1, stride=1, padding=0,
bias=use_bias),
                norm_layer(ndf * 2),
                nn.LeakyReLU(0.2, True),
                nn.Conv2d(ndf * 2, 1, kernel_size=1, stride=1, padding=0,
bias=use_bias)]
```

```
                self.net = nn.Sequential(*self.net)
```

Nakon same generacije diskirminatorske mreže, možemo objasniti kako je cijeli model sastavljen i objasniti kako mjeriti samu uspješnost modela.

3.8. Način rada

CycleGAN se temelji na osnovnog aspektu GAN-ova

U navedenom radu koristimo neuralnu mrežu specifičnog tipa CycleGAN kako bi trenirali model koji prepoznaje slike tlocrta i generira nove slike posotjećih modela

Nakon same generacije diskirminatorske mreže, možemo objasniti kako je cijeli model sastavljen i način na koju mjerimo samu točnost modela

Kada smo definirali način rada Generatora i diskriminatora, možemo objasniti rad cijele mreže.

```
        class CycleGANModel(BaseModel):
            """
            This class implements the CycleGAN model, for learning image-to-
image translation without paired data.
            The model training requires '--dataset_mode unaligned' dataset.
            By default, it uses a '--netG resnet_9blocks' ResNet generator,
            a '--netD basic' discriminator (PatchGAN introduced by pix2pix),
            and a least-square GANs objective ('--gan_mode lsgan').
            CycleGAN paper: https://arxiv.org/pdf/1703.10593.pdf
            """
            @staticmethod
            def modify_commandline_options(parser, is_train=True):
                """Add new dataset-specific options, and rewrite default values
for existing options.
                Parameters:
                    parser          -- original option parser
                    is_train (bool) -- whether training phase or test phase. You
can use this flag to add training-specific or test-specific options.
                Returns:
                    the modified parser.
                For CycleGAN, in addition to GAN losses, we introduce lambda_A,
lambda_B, and lambda_identity for the following losses.
                A (source domain), B (target domain).
                Generators: G_A: A -> B; G_B: B -> A.
                Discriminators: D_A: G_A(A) vs. B; D_B: G_B(B) vs. A.
                Forward cycle loss:  lambda_A * ||G_B(G_A(A)) - A|| (Eqn. (2) in
the paper)
                Backward cycle loss: lambda_B * ||G_A(G_B(B)) - B|| (Eqn. (2) in
the paper)
                Identity loss (optional): lambda_identity * (||G_A(B) - B|| *
lambda_B + ||G_B(A) - A|| * lambda_A) (Sec 5.2 "Photo generation from paintings"
in the paper)
                Dropout is not used in the original CycleGAN paper.
                """
                parser.set_defaults(no_dropout=True)  # default CycleGAN did not
use dropout
                if is_train:
                    parser.add_argument('--lambda_A', type=float, default=10.0,
help='weight for cycle loss (A -> B -> A)')
                    parser.add_argument('--lambda_B', type=float, default=10.0,
help='weight for cycle loss (B -> A -> B)')
                    parser.add_argument('--lambda_identity', type=float,
default=0.5, help='use identity mapping. Setting lambda_identity other than 0
has an effect of scaling the weight of the identity mapping loss. For example,
if the weight of the identity loss should be 10 times smaller than the weight of
```

```
the reconstruction loss, please set lambda_identity = 0.1')

                return parser
```

CycleGAN model se temelji na translaciji karakteristika uređenih parova, te njegova arhitektura je objašnjena na sljedeći način.
Naš generator kao ulaz dobiva latentne prostore šuma određenih dimenzija slike, te pokušava generirati izlaznu sliku, na temelju referentnog izvornog materijala. Kako se radi o uređenim parovima, cilj generatorske mreže je generirati Y varijablu para, prema karakteristikama prve slike. Drugim riječima, naš generator pokušava sve glavne karakteristike slike X prenijeti u sliku Y.

Diskriminator dobiva ulaz izmješan izvornim parovima i umjetnim parovima, onim parovima gdje je Y generiran sa strane generatora i ima zadatak da klasificira svaki par kao izvorni(1) ili umjetni(0). Arhitektura je načinjena tako da je Generator spojen u ulaz diskriminatora sa mješavinom raznih izvornih referentnih slika i radi se feedback loop gdje izlazi iz Diskriminatora su povezani na Generator.

Glavni cilj ovog modela je dovesti točnost Diskriminatora na 0.50, što znači da sam diskriminator ne može distinktirati razliku između referentnog i izvornog materijala. Naše slike, u ovom slučaju tlocrti onda postaju ekstremno slični izvornom materijalu, te smatramo da smo generirali apstraktno identičnu sliku onima iz izvornog materijala.

Međutim, još jedna distinktna karakteristika samog CycleGAN modela je da translaciju radi u oba smjera, to jest nezavisna i zavisna varijabla mijenjaju mjesta.

U svakom CycleGAN modelu imamo dve translacije: `x-y` i `y-x`.

```
            def __init__(self, opt):
    """Initialize the CycleGAN class.
    Parameters:
        opt (Option class)-- stores all the experiment flags; needs to be a
subclass of BaseOptions
    """
    BaseModel.__init__(self, opt)
    # specify the training losses you want to print out. The training/test
scripts will call <BaseModel.get_current_losses>
    self.loss_names = ['D_A', 'G_A', 'cycle_A', 'idt_A', 'D_B', 'G_B',
'cycle_B', 'idt_B']
    # specify the images you want to save/display. The training/test scripts
will call <BaseModel.get_current_visuals>
    visual_names_A = ['real_A', 'fake_B', 'rec_A']
    visual_names_B = ['real_B', 'fake_A', 'rec_B']
    if self.isTrain and self.opt.lambda_identity > 0.0:  # if identity loss is
used, we also visualize idt_B=G_A(B) ad idt_A=G_A(B)
        visual_names_A.append('idt_B')
        visual_names_B.append('idt_A')

    self.visual_names = visual_names_A + visual_names_B  # combine
visualizations for A and B
    # specify the models you want to save to the disk. The training/test scripts
will call <BaseModel.save_networks> and <BaseModel.load_networks>.
    if self.isTrain:
        self.model_names = ['G_A', 'G_B', 'D_A', 'D_B']
    else:  # during test time, only load Gs
        self.model_names = ['G_A', 'G_B']

    # define networks (both Generators and discriminators)
    # The naming is different from those used in the paper.
```

```
    # Code (vs. paper): G_A (G), G_B (F), D_A (D_Y), D_B (D_X)
    self.netG_A = networks.define_G(opt.input_nc, opt.output_nc, opt.ngf,
opt.netG, opt.norm,
                                    not opt.no_dropout, opt.init_type,
opt.init_gain, self.gpu_ids)
    self.netG_B = networks.define_G(opt.output_nc, opt.input_nc, opt.ngf,
opt.netG, opt.norm,
                                    not opt.no_dropout, opt.init_type,
opt.init_gain, self.gpu_ids)

    if self.isTrain:  # define discriminators
        self.netD_A = networks.define_D(opt.output_nc, opt.ndf, opt.netD,
                                        opt.n_layers_D, opt.norm, opt.init_type,
opt.init_gain, self.gpu_ids)
        self.netD_B = networks.define_D(opt.input_nc, opt.ndf, opt.netD,
                                        opt.n_layers_D, opt.norm, opt.init_type,
opt.init_gain, self.gpu_ids)

    if self.isTrain:
        if opt.lambda_identity > 0.0:  # only works when input and output images
have the same number of channels
            assert(opt.input_nc == opt.output_nc)
        self.fake_A_pool = ImagePool(opt.pool_size)  # create image buffer to
store previously generated images
        self.fake_B_pool = ImagePool(opt.pool_size)  # create image buffer to
store previously generated images
        # define loss functions
        self.criterionGAN = networks.GANLoss(opt.gan_mode).to(self.device)  #
define GAN loss.
        self.criterionCycle = torch.nn.L1Loss()
        self.criterionIdt = torch.nn.L1Loss()
        # initialize optimizers; schedulers will be automatically created by
function <BaseModel.setup>.
        self.optimizer_G =
torch.optim.Adam(itertools.chain(self.netG_A.parameters(),
self.netG_B.parameters()), lr=opt.lr, betas=(opt.beta1, 0.999))
        self.optimizer_D =
torch.optim.Adam(itertools.chain(self.netD_A.parameters(),
self.netD_B.parameters()), lr=opt.lr, betas=(opt.beta1, 0.999))
        self.optimizers.append(self.optimizer_G)
        self.optimizers.append(self.optimizer_D)
```

U navedenom kodu, vidimo obradu uređenih parova i inicijalizaciju same CycleGAN mreže. Rade se klasifikacije uređenih parova, te se prikazuju pravi izvorni podaci i njihovi umjetni duplikati.

Instanciramo dve vrste generatora i diskriminatora, kako bi mogli koristiti mrežu u svim smjerovima našeg uređenog para. Jedna smjer prebacuje karakteristike Y slike u X sliku, a drugi obratno. Ovo je važan princip, kako nam obosmjerna translatacija pomaže u utvrđivanju korelacije samog modela i same korelacije karakteristike među slikama. Treniramo diksriminatore i dalje počinjemo s generacijom samih referentnih materijala.

CycleGAN arhitektura se odvija u epohama. Drugim riječima, diskriminator i generator dobivaju rezultate kontrasnog modela iz prethodne epohe, kako bi mogli analizirati svoje greške i raditi točnije distinikcije/analize.

Na ovaj način, ova dva modela se cijelo vrijeme konstantno natječu, kako bi mogli utvrditi svoje mjesto.

Detaljniju analizu smjerova možemo vidjeti u sljedećem referentnom kodu

```
        def forward(self):
    """Run forward pass; called by both functions <optimize_parameters> and
<test>."""
    self.fake_B = self.netG_A(self.real_A)  # G_A(A)
    self.rec_A = self.netG_B(self.fake_B)   # G_B(G_A(A))
    self.fake_A = self.netG_B(self.real_B)  # G_B(B)
```

```
    self.rec_B = self.netG_A(self.fake_A)   # G_A(G_B(B))

def backward_D_basic(self, netD, real, fake):
    """Calculate GAN loss for the discriminator
    Parameters:
        netD (network)      -- the discriminator D
        real (tensor array) -- real images
        fake (tensor array) -- images generated by a generator
    Return the discriminator loss.
    We also call loss_D.backward() to calculate the gradients.
    """
    # Real
    pred_real = netD(real)
    loss_D_real = self.criterionGAN(pred_real, True)
    # Fake
    pred_fake = netD(fake.detach())
    loss_D_fake = self.criterionGAN(pred_fake, False)
    # Combined loss and calculate gradients
    loss_D = (loss_D_real + loss_D_fake) * 0.5
    loss_D.backward()
    return loss_D

def backward_D_A(self):
    """Calculate GAN loss for discriminator D_A"""
    fake_B = self.fake_B_pool.query(self.fake_B)
    self.loss_D_A = self.backward_D_basic(self.netD_A, self.real_B, fake_B)

def backward_D_B(self):
    """Calculate GAN loss for discriminator D_B"""
    fake_A = self.fake_A_pool.query(self.fake_A)
    self.loss_D_B = self.backward_D_basic(self.netD_B, self.real_A, fake_A)

def backward_G(self):
    """Calculate the loss for generators G_A and G_B"""
    lambda_idt = self.opt.lambda_identity
    lambda_A = self.opt.lambda_A
    lambda_B = self.opt.lambda_B
    # Identity loss
    if lambda_idt > 0:
        # G_A should be identity if real_B is fed: ||G_A(B) - B||
        self.idt_A = self.netG_A(self.real_B)
        self.loss_idt_A = self.criterionIdt(self.idt_A, self.real_B) * lambda_B
* lambda_idt
        # G_B should be identity if real_A is fed: ||G_B(A) - A||
        self.idt_B = self.netG_B(self.real_A)
        self.loss_idt_B = self.criterionIdt(self.idt_B, self.real_A) * lambda_A
* lambda_idt
    else:
        self.loss_idt_A = 0
        self.loss_idt_B = 0

    # GAN loss D_A(G_A(A))
    self.loss_G_A = self.criterionGAN(self.netD_A(self.fake_B), True)
    # GAN loss D_B(G_B(B))
    self.loss_G_B = self.criterionGAN(self.netD_B(self.fake_A), True)
    # Forward cycle loss || G_B(G_A(A)) - A||
    self.loss_cycle_A = self.criterionCycle(self.rec_A, self.real_A) * lambda_A
    # Backward cycle loss || G_A(G_B(B)) - B||
    self.loss_cycle_B = self.criterionCycle(self.rec_B, self.real_B) * lambda_B
    # combined loss and calculate gradients
    self.loss_G = self.loss_G_A + self.loss_G_B + self.loss_cycle_A +
self.loss_cycle_B + self.loss_idt_A + self.loss_idt_B
    self.loss_G.backward()
```

Nakon svake epohe, iznimno je bitno da optimiziramo paramtere, kao što je anvedeno:

```
 def optimize_parameters(self):
```

```
    """Calculate losses, gradients, and update network weights; called in every
training iteration"""
    # forward
    self.forward()      # compute fake images and reconstruction images.
    # G_A and G_B
    self.set_requires_grad([self.netD_A, self.netD_B], False)  # Ds require no
gradients when optimizing Gs
    self.optimizer_G.zero_grad()  # set G_A and G_B's gradients to zero
    self.backward_G()             # calculate gradients for G_A and G_B
    self.optimizer_G.step()       # update G_A and G_B's weights
    # D_A and D_B
    self.set_requires_grad([self.netD_A, self.netD_B], True)
    self.optimizer_D.zero_grad()   # set D_A and D_B's gradients to zero
    self.backward_D_A()      # calculate gradients for D_A
    self.backward_D_B()      # calculate graidents for D_B
    self.optimizer_D.step()  # update D_A and D_B's weights
```

Footer

Ovdje vidimo da nakon svakog koraka renkostrukcije umjetnih slika, restiramo gradiente naših generatora, kalkuliramo nove gradiente i postavljamo nove težine na same mreže generatora. Također vrijedi i za same diskriminatore.

3.9. Mjerenje točnosti

Postoji nekoliko metoda za mjerenje točnosti CycleGAN arhitekture. Prva metoda je općenita, te se može koristiti sa distinkcijom pravih slika i umjetno genriranih slika. Glavni cilj ovakve arhitekture je micanje plauzibilnosti ljudskom oku. Drugim riječima, indeferncija referentnog i generiranog materijala.
Međutim, također jedan od glavnih načina koji ćemo implementirati u našem radu je Loss Scoring i FCN Scoring, koji ćemo detaljnije objasniti u rezultatima.[1][2][5]

3.10. Podaci

Kako bi mogli dobiti kvalitetne rezultate, bitno je imati kvalitetne podatke. Za naše testne podatke koristili smo dve baze podataka različitih tlocrta.
Prva baza podataka se satoji od 100 slika europskih tlocrta, te je segregirana u uređene parove ocrtanih granica tlocrta, te prirpadajućih parove samog rasporeda glavnih soba u tlocrtu.

Cilj prvog testnog primjera je smjestiti sobe na odgovarajuća mjesta u samom tlocrtu, postujući granice tlocrta, te usporediti slike s izvornim parovima.

Svrha prvog eksperimenta je da počnemo s temeljnim principima arhitekturalne izrade digitalnog tlocrta, to jest odrediti točnost umjetne inteligencije u raspoznavanju karakteristika tlocrta

Drugi eksperimentalni podaci je baza podataka kineskih tlocrta koje se sastoji od 3 imperativne komponente. Prva komponenta je granica samih tlocrta, bez ograda samih soba. Za razliku od prvog testnog skupa, sada puštamo model da sam stavlja granice soba, ovisno o kojem uređenom paru govorimo. Druga komponenta ovog testnog skupa su određeni rasporedi soba u samom testnom skupu, te se mogu koristiti kao uređeni parovi sa prvom komponentom. Naš model može precrtavati pozicije soba u same granice tlocrta, međutim glavna razlika između ovog i prvog testnog skupa je što nam granice soba nisu određene. OVdje model uzima apstrakniji pristup, te je ovaj tetni skup malo zahtjevniji nego prvi. Zadnja komponenta koju koristimo su gotovi nacrtani tlocrti prve i druge komponente. Ovo su gotovi grafički nacrti, te je glavni cilj ovakvog pristupa realizirati gotov grafički

digitalni tlocrt koji se može koristiti u komercijalne i arhitekturalne svrhe.

U drugom eksperitalnom skupu, imamo dva uređena para: prva komponenta se povezuje sa drugom kako bi mogli vidjeti korelaciju između nastajanja rasporeda soba u samom neispunjenom tlocrtu, te druga komponenta se povezuje s trećom kako bi mogli vidjeti obradu i generaciju pravog grafičkom tlocrta na temelju određenih pozicija rasporeda soba u tlocrtu.

Idealno, spajanje prve komponente s trećom je dosta neuspješan eksperiment trenutačno, kako proizvoljno nemamo dovoljno informacija u drugoj komponenti da generiramo dobar konačan rezultat, ali ovu temu ćemo kasnije razraditi.

3.11. Korištenje Blendera u cjevovodu obrade podataka

3.11.1. Integracija geometrijskih čvorova

Sada kada imamo generirane tlocrte, vizualizacija tlocrta u 3dimenzionalnom prostoru pomaže da detaljnije analiziramo greške koje su nastale pri samoj generaciji umjetnih tlocrta i simulaciji automatizacije generiranja 3d prostora samih tlocrta.
Kako bi sljedeći korak mogli parametarski izvodit, koristit ćemo Blender. Blender je alat za 3d modeliranje, međutim specifično ćemo se fokusirati na Blenderov parametarski uređivač, takozvani “Geometry Nodes”. Pomoću Geometry Nodesa, možemo napraviti parametarski set nodica, koji za svaku ubačenu sliku tlocrta, generira 3d model tog tlocrta.

Naš parametarski program izgleda kako je prikazano u navedenoj slici, a u daljnjem poglavlju ćemo objasniti kako funkcionira svaka individualna nodica.

Prvi dodani čvor je UV Unwrap. Ovaj čvor uzima našu digitalnu teksturu tlocrta, koji je označen kao crno gdje su zidovi, te transpraentno gdje nema ničega, te ga iz dvodimenzionalnog polja transformira u jednodimenzionalno polje. Ovo je bitan korak jer Blender mora znati kako naša tekstura leži u tridimenzionalnom prostoru

Nakon toga koristimo grid i cylinder čvorove, kako bi svaki piksel iz naše teksture označenim crnom bojom mogli pretvoriti u poligon. Ovi poligoni se onda međusobno spajaju, kako bi dobili definiranu crtu koja nam označuje granice tlocrta. Instance on points nam generira geometriju, kao što je anvedeno u objašnjenju čvorova. ovo je bitan korak jer želimo optimizirano raditi prema našem tlocrtu, a ne imati veliki broj poligona koji su individualno ne povezani, zbog optimizacije modela.

Kada imamo generiranu geometriju, koristimo set Material čvor, koji pridružujemo materijal po izboru. Za lakšu prepoznatljivost, dodan je bijeli materijal.

Sve što ostaje je generirati tlo tlocrta i gornji vrh tlocrta, te se u drugoj grupi čvorova generiraju instance s krajnjih strana tlocrta kao plohe.

Nakon detaljnjog objašnjenja kako Blender generira 3d modele naših slika, možemo krenuti sa analizom rezultata.

Što se samih rezultata tiče, budući da radimo s poligonima malog broja, generiranje ovakvog tlocrta traje nekoliko sekundi.

4. Rezultati modela

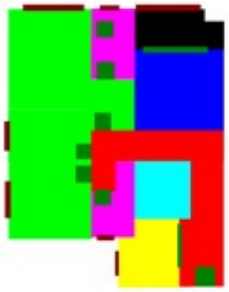
Kako bi analizirali same rezultate, moramo objasniti na koji način su se izvodili rezultati.
Za generiranje samih tlocrta pomoću umjetne inteligencije,s vi eksperimenti su se izvodili na GTX 1660 Super grafičkoj kartici, u različitim epohama s različitim strukturama mrežqa generatora i diskriminatora

U sljedećim slikama, uspoređujemo arhitketuru našeg CycleGAN modela s drugačijim modelima, koji su odradili istu praktičnu primjenu, s istim bazama.

Skupovni podaci su dve baze, od kojih svaka ima 100 uređenih parova.

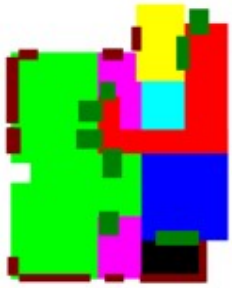
Daljnje rezultate možemo vidjeti u tablicama. #slike

Što se tiče samog FCN scoringa i loss scoringa, naš model je imao zadovoljavajuću uspješnost, kao što je prikazano na sljedećoj slici



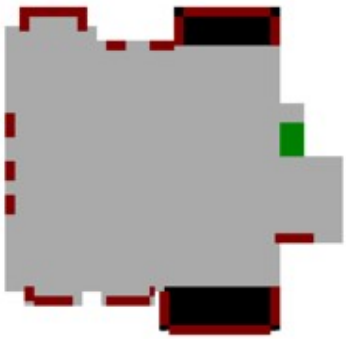
Slika 4 Prva model koji je geneiriran

Slika 5 Prvi model koji generiramo



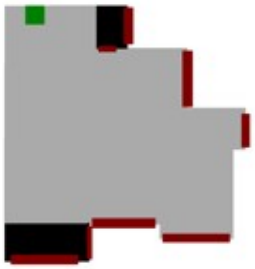
Slika 6 Drugi model koji je generiran

Slika 7 Drugi model koji generiramo

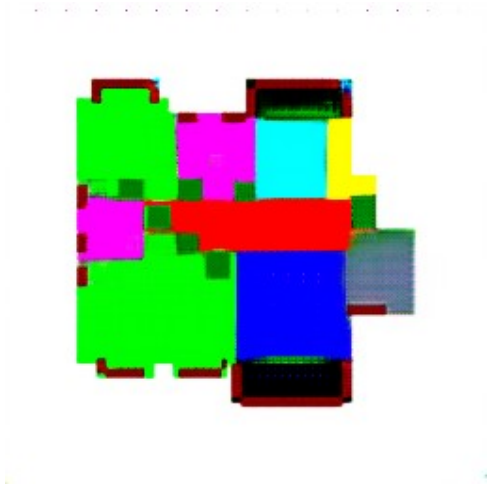


Slika 8 Treci model koji je geneiriran

Slika 9 Treći model koji generiramo



Slika 10 Cetvrti model koji je generiran



Slika 11 Četvrti model koji generiramo

4.1. Usporedba rezultata s drugim modelima

Kod usporedbe rezultata, moramo analizirati okoline na kojima su se ove epohe vrtile.
U našoj okolini, GTX 1660 je izrazito slabije kvalitete, te je naše vrijeme treniranja setova iznosilo svega 26.5 sati, dok su CycleGAN i CUT metodologije se izvršavale na superračunalima i imali vrijeme izvršenja od 7 sati.

Prilikom daljnje analize, vidimo negativne strane ovakve tehnologije. Iako je analiza izvršena na superračunalima, zahtijeva nekoliko sati, te postoji još mjesta za daljnji napredak.

Također daljnja parametrizacija na Pix2Pix modelu je isto moguća, kako vidimo da naši modeli dilutiraju pozicije namještaja i same slike graničnih odjeljaka mogu imati bolje definirane granice između soba

Tablica 1 Mjerenja različitih modela

Model	**Pix2Pix**	**StyleGan**	**CycleGAN**
Vrijeme izvođenja	367820s	262222s	222160s
MSE	5.0929292	2.092931	0.9702231

4.2. Mogući daljnji koraci

Pri daljnjem nastavljanju istraživanja u okviru ovog diplomskog rada, naveo bi sljedeće. Moguće je optimizirati i poziciju smještaja u digitlanim tlocrtima, te generirati 3d modele uskladnu u Blenderu. Ovakav pristup je također bio planiran za ovaj diplomski rad, međutim, nije bio uspješno ukomponiran. Problem je u skupnim podacima i njihovim generiranim modelima, te modeli nisu bili izrazito detaljno definirani, pa se nisu mogli modelirati u Blenderu.
Daljnje, predlažem promjenu skupnih podataka, kao i testiranje na različitim ostalim metodologijama konvolucijskih neuralnih mreža. Iako su u ovom rado analizirane 3 standardizirane metodologije, postoje različiti faktori koji se mogu analizirati.

Također, daljnja parametrizacija i testovi, te eksperimentacija u koracima je također potrebna, kako bi se mogao dobiti detaljniji uvid u problematiku.

5. Zaključak

Kao zaključak naveo bi sljedeće. Naša praktična primjena je izvedena na prosječnoj opremi, međutim, sama činjenica da je ovakva tehnologija lagano pristupačna te se može koristi za komercijalne svrhe pokazuje koliko umjetna inteligencija napredovala kroz nekoliko godina.
Sama parametrizacija naših rezultata je prihvatljiva, međutim, u sklopu daljnjih optimizacija ovog rada, definitivno se preporučaju daljnji testovi s parametrima. Promjena UNeta s ResNet-a, te ostali glavni parametri bi mogli poboljšati uspješnost ovog eksperimenta.

Kao općeniti zaključak, također vidimo brzinu generaciju 2d modela u 3d okolinu pomoću Blendera, te zbog brzine generacije, ovaj dio nije bio mjeren. Sve 3 metode neuralnih mreža navedene u ovom diplomskom radu generiraju zadovoljavajuće rezultate, po vizualnim i mjernim jedinicama.

Ovim diplomskim radom je pokazano da se umjetna inteligencija može koristiti na različitim područjima primjene, kao automatizacijsko sredstvo za različite grane različitih znanosti.

6. Literatura

[1] https://arxiv.org/pdf/2007.15651.pdf, T.Park,A. Efros, R. Zhang, Jun-Yan Zhu(2019.) Contrastive Learning for Unpaired Image-to-Image Translation, University of Berkley

[2] https://arxiv.org/pdf/1611.07004.pdf, Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros(2020.) ,Image-to-Image Translation with Conditional Adversarial Networks, University of Berkley

[3] http://www.blender.org, Community BO. Blender - a 3D modelling and rendering package Geometry Nodes, Amsterdam; 2022.

[4] Ripley, Brian D. (1996) Pattern Recognition and Neural Networks, Cambridge

[5] Bishop, C.M. (1995) Neural Networks for Pattern Recognition, Oxford: Oxford University Press.

[6] Charu C. Aggarwal (2018.), Neural Networks and Deep Learning: A Textbook,Springer International Publishing

[7] Pat Nakamoto (2017), Neural Networks & Deep Learning, Createspace Independent Publishing, 2017

[8] Ech-Chouyyekh, Monir & Omara, Hicham & LAZAAR, Mohamed. (2019). Scientific paper classification using Convolutional Neural Networks.

[9] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7557439/ Jingwen Zhang, Yoo Jung Oh, Patrick Lange, Zhou Yu, Yoshimi Fukuoka (2020.), Artificial Intelligence Behavior Change, Physical Activity and a Healthy Diet: Viewpoint, Department of Communication, University of California,

[10] https://www.mdpi.com/2673-5172/3/1/2/html de-Lima-Santos M-F, Ceron W. Artificial Intelligence perceptions

[11] https://news.miami.edu/stories/2022/07/can-artificial-intelligence-help-journalists.html Iann Howie, (2021.), Can artificial intelligence help journalists?, , Miami Education News

[12] https://www.futurity.org/ai-journalism-2778042-2/, Owsley, C.S.,Greenwood, K. Awareness and perception of artificial intelligenceoperationalized integration industry and society. AI& Soc (2022).

[13] https://iksz.fsv.cuni.cz/en/research/center-artificial-intelligence-journalism, Vaclav Moralec (2021.), AI and Arhitecture

[14] https://reutersinstitute.politics.ox.ac.uk/news/uk-media-coverage-artificial-intelligenceć-dominated-industry-and-industry-sources, : J.Scott Brennen, Philip N. Howard, and Rasmus Kleis Nielsen,(2021.), An Industry-Led Debate in Artificial Intelligence, University of Oxford

[15] https://aijournalism.net/everything-you-need-to-know-about-the-ai-writing-assistant-and-how-its-changing-our-lives/Abass Alzanjne(2021.),Everything You Need to Know About the AI Writing Assistant and How it's Changing Our Lives, AIJRF

[16] https://www.semanticscholar.org/paper/Artificial-Intelligence-and-Automated-Journalism%3A-Ali-Hassoun/880a5eef74b89e5beadc0eb106643864820cd659, Waleed Ali, Mohamed Hassoun (2021.), Artificial Intelligence and Automated Arhitecture: Contemporary Challenges and NewOpportunities, Department of Media, Tanta University, Egypt

[17] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8003152/ Stefan Reitmann, Lorenzo Neumann, and Bernhard Jung(2020.), BLAINDER—A Blender AI Add-On for Generation of Semantically Labeled Depth-Sensing Data

[18] https://arxiv.org/abs/2205.07540 Anaïs Tack, Chris Piech, (2022.), The AI Teacher Test: Measuring the Pedagogical Ability of Blender and GPT-3 in Educational Dialogues

[19] https://www.optica.org/en-us/events/webinar/2019/how_to_use_blender_to_create_attractive_scientific/ Nathaniel Kinsey,How to Use Blender to Create Attractive Scientific Figures and Journal Cover Art, Virginia Commonwealth University

[20] Alan Britto, Blender 2.9: The Beginner's Guide, Amazon Digital Services LLC - KDP Print US, 2020.

[21] Oliver Villar(2017.),Learning Blender: A Hands-On Guide to Creating 3D Animated Characters,Addison-Wesley Professional

[22] Tero Karras, Samuli Laine, Timo Aila, (2018.), A Style-Based Generator Architecture for Generative Adversarial Networks

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2021.), Deep Residual Learning for Image Recognition, Universitasy of Berkeley

[24] https://arxiv.org/pdf/1505.04597.pdf Olaf Ronneberger, Philipp Fischer, and Thomas Brox (2021.), U-Net: Convolutional Networks for Biomedical Image Segmentation,Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany

[25] Ruan Lotter(2022.), Taking Blender to the Next Level: Implement advanced workflows such as geometry nodes, simulations, and motion tracking for Blender production pipelines, Packt Publishing

[26] Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition

[27] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[28] Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization.

[29] Xie, Q., Hovy, E., Luong, M. T., & Le, Q. V. (2019). Self-training with Noisy Student improves ImageNet classification.

[30] Chen, T., Zhai, X., Ritter, M., Lucic, M., & Houlsby, N. (2019). Self-supervised gans via auxiliary rotation loss. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 12154-12163).

[31] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., ... & Hassabis, D. (2017). Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences

[32] Miyato, T., Kataoka, T., Koyama, M., & Yoshida, Y. (2018). Spectral normalization for generative adversarial networks

7. Popis tablica

8. Popis slika