SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5119

PRIJENOS UMJETNIČKOG STILA OPTIMIRANJEM KONVOLUCIJSKOG MODELA

Lucija Ivković

Zagreb, srpanj 2017.

Poštovanom profosoru Siniši Šegviću zahvaljujem se na mentorstvu i pomoći pruženoj pri pisanju ovog završnog rada.

Sadržaj

1 Uvod

Prijenos stila s jedne slike na drugu u osnovi se svodi na problem razdvajanja informacija koje opisuju objekte odnosno sadržaj koji slika prikazuje, od informacija koje nose teksturu slike. Problem je poprilično složen jer je pojam teksture vrlo teško precizno definirati.

Što zapravo znači tekstura, odnosno stil slike? Različiti odgovori proizlaze iz ovog pitanja, ovisno o kakvoj slici se radi. U ovom radu će stoga fokus biti na umjetničkim slikama poznatih slikara kao izvorima teksture, jer je na njima lako prepoznati i definirati teksturu, prvo prirodnim vidom (npr. karakteristični potezi kistom određenog umjetnika) a zatim i matematičkom evaluacijom kojom ju računalo može percipirati. Ciljno pitanje jest - kako bi neka proizvoljna fotografija, koja može bez ograničenja prikazivati bilo kakav objekt, izgledala da ju je nacrtao recimo Van Gogh u stilu svoje čuvene Zvjezdane noći, ili Edvard Munch u stilu legendarnog Krika? Algoritam koji nalazi odgovor na ovo pitanje trebao bi moći sačuvati strukturu svih objekata koje proizvoljna fotografija prikazuje, uz to da ju izmijeni tako da finalni rezultat "podsjeća" na željenu umjetničku sliku, kao što je prikazano na slici 1.1.

Postoje dva temeljna pristupa ovome problemu. Prvi pristup obuhvaća sve dosad razvijene neparametrizirane algoritme koji mogu sintetizirati teksturu na željenoj fotografiji direktnim preoblikovanjem izvorne teksturirane slike na razini piksela. Temeljni preduvjet koji nedostaje ovim algoritmima u odnosu na drugu skupinu algoritama baziranim na parametriziranom pristupu jest reprezentacija slike takva da se njome istovremeno modelira i sadržaj i tekstura nove slike iz

željenih, izvornih slika. Nedostaje precizna definicija modela teksture.

Analizirat ću obje navedene skupine algoritama, te matematičke metode koje se kriju ispod njih. Osobito ću se osvrnuti na jedan odabrani među parametriziranim algoritmima koji koristi duboku konvolucijsku neuronsku mrežu, te uz analizu priložiti ocjenu rezultata tog algoritma u ovisnosti o različitim čimbenicima i parametrima mreže.

Slika 1.1 Rezultati prijenosa stila na digitalnu fotografiju (lijevo) koristeći sliku Joaquín Sorolle (gore desno) i Henri Matissea (dolje desno) kao izvor teksture. Za bolje rezultate poželjno je da objekti na fotografiji i umjetničkoj slici budu donekle slični.

2 Postojeći algoritmi i modeli

Prije ulaska u detalje metoda za sintetiziranje teksture dobro bi bilo odgovoriti na nekoliko tehničkih pitanja o teksturi. Što uopće predstavlja pojam 'tekstura'? Što zapravo znači 'sinteza teksture'?

U najširem smislu riječi, tekstura se koristi za oblikovanje raznih značajki površine, uključujući boju, transparentnost, i sl. U okvirima računalne grafike i računalnog vida, ovaj termin odnosi se na sliku vizualne ili taktilne površine koja sadrži proizvoljnu količinu ponavljajućih uzoraka. Te ponavljajuće jedinice nazivaju se 'texeli'. Tekstura kao takva stoga može se opisati karakteristikom strukturiranosti, prema čemu može biti deterministička, tj. sastavljena od očitih texela raspoređenih po određenim pravilima (primjerice parket na podu dnevnog boravka) ili stohastična, bez eksplicitno uočljivih texela (poput, recimo pješčane plaže). Predlošci teksture se u svrhu sintetiziranja mogu pribaviti iz ručno izrađenih slika ili skeniranih fotografija. Poželjno bi bilo naravno, da ti predlošci budu u što boljoj rezoluciji i po stupnju varijabilnosti texela koje sadrže, što bliže stvarnoj teksturi (čija je veličina neograničena, pošto ponavljajući uzorci od kojih se sastoji, u principu sežu u beskonačnost).

Sinteza teksture podrazumijeva proces stvaranja nove slike proizvoljne veličine iz danog predloška, takve da čovjek promatrajući generiranu sliku i ulazni predložak zaključi da su obje slike sačinjene od iste teksture. Cijeli proces se može razložiti na dvije komponente, analizu i sintezu teksture. Analiza obuhvaća procjenu i razvoj modela željene teksture iz danog primjerka. Model treba biti dovoljno precizan da sadržava i stohastičke i determinističke elemente

te teksture. Sinteza zatim, koristeći razvijeni model, izgrađuje novu sliku koja je vizualno slična ulaznom primjerku, te koja izgleda prirodno i konzistentno bez ikakvih deformacija. Sintetizirani rezultat se potom može koristiti u raznim poslovima unutar računalne grafike, poput prijenosa površine gradivnog materijala s jednog objekta na drugi, ili pak prijenosa stila umjetničke slike na digitalnu fotografiju, što je upravo tema ovoga rada.

2.1 Neparametrizirani algoritmi

Za svaki kvalitetan algoritam prvo je potrebno dobro definirati matematički model problema za koji se taj algoritam sastavlja. U ovom slučaju treba nam model sinteze teksture. Model koji je u polju grafičkih aplikacija pružio poprilično zadovoljavajuće rezultate u odnosu na mnoge druge koji su kroz povijest bili predlagani, baziran je na Markovljevim slučajnim poljima (Wei, 2009.). Markovljeva slučajna polja opisana su u domeni teorije vjerojatnosti kao skup slučajnih varijabli koji zadovoljava dva važna svojstva, svojstvo lokalnosti i svojstvo stacionarnosti. Unutar domene računalne grafike, opis navedenih svojstava je sljedeći. Svojstvo stacionarnosti ima slika koja je uzorkovana ili drugim riječima, slika koja se, gledana kroz mali pomični prozorčić ljudskom oku čini relativno slična po svim svojim dijelovima. Svojstvo lokalnosti je prisutno na slici ako se vrijednost svakog piksela te slike može procijeniti na temelju nekoliko susjednih piksela, neovisno o ostatku slike. Kompletan proces modeliranja teksture neparametriziranim metodama oslanja se na navedena dva svojstva (Wei, 2009.).

Iako se praktički sve neparametrizirane metode sinteze teksture, i njihove mnogobrojne ekstenzije, u osnovi svode na puko preoblikovanje ulaznog

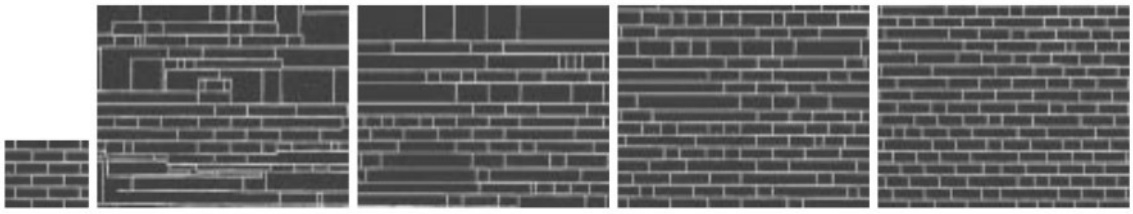
primjerka s određenom dozom nasumičnosti, svejedno daju poprilično dobre rezultate u radu s prirodnim teksturama. U nastavku ću u detalje obraditi dva odabrana algoritma iz ove skupine. Prvi (Efros, 1999.) je temeljen na neparametriziranom preoblikovanju ulaznog predloška prema pravilima distribucije slučajnih varijabli (*eng. non-parametric sampling*). Drugi (Efros, 2001.) je nešto noviji, jednostavniji i računski manje zahtjevan od prvog spomenutog, a bazira se na popločavanju nasumično odabranih isječaka ulaznog predloška (*eng. image quilting*).

2.1.1 Sinteza teksture preoblikovanjem ulazne slike

Inspiracija ovog algoritma (Efros, 1999.) nađena je u generaliziranom Markovljevom lancu, nizu od n uzastopnih jedinica koje tvore kontekst, tzv. histogram (stupčasti graf koji prikazuje distribuciju numeričkih podataka). Histogram prikazuje procjenu distribucije vjerojatnosti kontinuirane varijable, koja predstavlja jedinicu konteksta, po unaprijed određenim intervalima unutar domene kojoj ta varijabla pripada. Ovime je stoga moguće potpuno odrediti distribuciju vjerojatnosti sljedeće, nepoznate jedinice u nizu. Da bi ova ideja bila primjenjiva na spomenuti problem sinteze teksture, potrebno je odrediti što će biti jedinica i kontekst pri sintezi, te konstruirati prikladnu distribuciju vjerojatnosti kojom će se jedinicu po jedinicu sintetizirati tekstura.

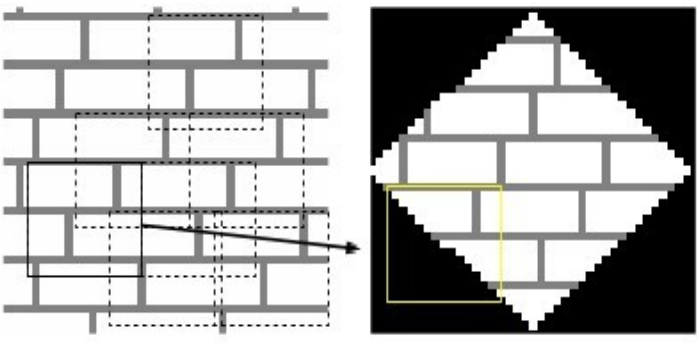
Za jedinicu sinteze odabran je jedan piksel, što znači da se sintetizirana slika gradi piksel po piksel. Svi prethodno sintetizirani pikseli u kvadratnom prozoru čije središte je trenutni, nepoznati piksel, čine kontekst sinteze. Ovo se još naziva i susjedstvo nepoznatog piksela. Ovdje valja napomenuti da se veličina prozora koji određuje susjedstvo treba odabrati pažljivo, uzimajući u

obzir da prozor treba biti dovoljno velik da obuhvati lokalnu strukturu teksture, ali istovremeno i dovoljno malen da se razmještaj i interakcija između tih struktura ostavi na slobodu rada algoritma. Na slici 2.1 prikazano je kako odabir veličine prozora utječe na konačni rezultat sinteze.



Slika 2.1 (preuzeta s: http://www.faculty.idc.ac.il/arik/seminar2009/papers/efros-iccv99.pdf) Uz dani predložak teksture (lijevo) algoritam je sintetizirao sljedeće četiri slike uz postupno povećavanje veličine susjedstva slijeva nadesno. Slike jasno prikazuju na koji način veličina prozora utječe na stupanj slučajnosti pri oblikovanju teksture, što je samo po sebi zapravo poprilično intuitivno.

Nadalje preostaje samo problem distribucije. Kako je praktično nemoguće distribuciju eksplicitno definirati za svaki mogući kontekst, dobra aproksimacija može se postići na mnogo jednostavniji način. Za svaki novi kontekst provede se ispitivanje ulaznog predloška teksture. Traže se susjedstva koja odgovaraju već sintetiziranim, poznatim pikselima u kontekstu. Distribucija nepoznatog piksela se može konstruirati kao histogram svih vrijednosti središnjeg piksela onih susjedstva pronađenih u predlošku. Ilustracija ove ideje prikazana je na slici 2.2, a detaljnije matematičko objašnjenje slijedi u nastavku.



Slika 2.2 Na slici lijevo je prikazan dani predložak teksture, a desno proces sinteze, gdje je žutim kvadratom označen trenutni kontekst. Na predlošku su isprekidanim kvadratnim okvirima označena sva nađena odgovarajuća susjedstva, iz čijih se središnjih piksela gradi distribucija nepoznatog središnjeg piksela unutar žutog kvadrata koji se sintetizira.

(slika preuzeta s: http://www.faculty.idc.ac.il/arik/seminar2009/papers/efros-iccv99.pdf)

Neka je I slika koja se sintetizira iz ulaznog predloška I_{ul} što je pravi podskup od I_{inf} stvarne, beskonačne teksture. Neka je p piksel koji se trenutno određuje, i neka je $\omega(p)$ njegovo dosad sintetizirano susjedstvo. Vizualnu udaljenost (ili sličnost) između neka dva susjedstva ω_1 i ω_2 možemo označiti s $d(\omega_1, \omega_2)$. Nadalje, možemo definirati skup

$$\Omega(p) = \{\omega' \subset I_{inf} : d(\omega', \omega(p)) = 0\} \tag{2.1}$$

koji sadrži sva pojavljivanja susjedstva piksela p u teksturi. No, problem je što nemamo na raspolaganju cijelu, beskonačnu teksturu već samo njezin mali predložak I_{ul}. Nemamo garanciju da ćemo u tom konačnom predlošku pronaći ijedno susjedstvo ω' koje odgovara susjedstvu nepoznatog piksela $\omega(p)$. Dakle, potrebna nam je dobra heuristika koja će pronaći skup $\Omega'(p)$ iz predloška I_{ul}, koji je dovoljno sličan skupu $\Omega(p)$, da bismo mogli uvijek sa sigurnošću konstruirati distribuciju. Jedna ideja za implementaciju spomenute heuristike je varijacija na 'pohlepnu' metodu najbližeg susjeda; pronaći susjedstvo $\omega_{najbolje}$ u I_{ul} koje najbolje odgovara susjedstvu $\omega(p)$, i u skup $\Omega'(p)$ uvrstiti sva susjedstva ω iz I_{ul} za koja vrijedi

$$d(\omega, \omega(p)) < (1+\varepsilon) \cdot d(\omega_{najbolje}, \omega(p)) \tag{2.2}$$

pri čemu je ε parametar tolerancije za odgovarajuća susjedstva u danom predlošku. Sada možemo od središnjih piksela svih susjedstva u skupu $\Omega'(p)$ konstruirati histogram za distribuciju nepoznatog piksela i odrediti ga. Jedino što nam još preostaje je prikladno definirati funkciju za udaljenost d između dva susjedstva. Mogli bismo koristiti normaliziranu sumu kvadriranih

razlika vrijednosti piksela d_{SSD}, no ta tehnika uvijek daje jednaku grešku za svaki neodgovarajući piksel, što se ne uklapa u svojstvo lokalnosti Markovljevog modela. Da bismo sačuvali svojstvo lokalnosti, funkcija bi trebala dati veću pogrešku za bliže piksele nego za one udaljenije. Taj problem lako se može riješiti sitnom nadogradnjom sume kvadriranih razlika; pomnožiti d_{SSD} s 2D Gaussovom jezgrom. Gaussova jezgra pomnožena s nekom aproksimacijskom značajkom ostvaruje eksponencijalno opadajući efekt vrijednosti te značajke, pri čemu je maksimalna vrijednost zadržana u središtu promatranja (u našem slučaju oko središnjeg, trenutno sintetiziranog piksela), te uniformno opada u svim smjerovima. Ovime je osiguran snažniji doprinos vrijednosti piksela bližih središnjem pikselu te slabiji ili nikakav doprinos udaljenih piksela, pri izračunu udaljenosti između dva susjedstva.

Mada generalno daje vrlo dobre rezultate, ipak valja spomenuti nekoliko ključnih nedostataka ovog algoritma. Pošto je za sintezu svakog piksela generirane slike potrebna kompletna pretraga ulaznog predloška, nije teško zaključiti da algoritam izrazito sporo radi. Naravno, efikasnost se uvijek može poboljšati raznim metodama optimizacije, no uz to je prisutan još jedan nezanemariv problem. Svaki algoritam baziran na 'pohlepnoj' ideologiji podložan je opasnosti da zaglavi u krivom području prostora pretraživanja, pa uzastopnim biranjem krivih piksela za sintezu počne razvijati 'smeće' na sintetiziranoj slici.

2.1.2 Popločavanje ulaznog predloška

Kako i sam naziv govori, riječ je o popločavanju nasumično odabranih kvadratnih isječaka iz priloženog predloška teksture tako da se rubna područja konzistentno preklapaju. Iako nije prikladan za strogo strukturirane teksture (npr.

pepita uzorak, ili šahovska ploča) vrlo učinkovito rješava problem sinteze mnogih stohastičkih tekstura. U odnosu na prethodno opisani, ovaj algoritam je poprilično jednostavan i računski jeftin. Također, otklonjena je opasnost od razvijanja 'smeća' jer se slika ne sastavlja piksel po piksel, već se za jedinicu sinteze izabire cijeli isječak, skupina piksela. Ideja algoritma u osnovnim koracima prikazana je na slici 2.3, a detaljan opis rada algoritma popločavanja dan je u nastavku.

Za jedinicu sinteze uzet ćemo kvadratni blok B_i čija veličina je ostavljena korisniku da ju odredi. Veličina bloka odgovara veličini isječaka od kojih se stvara nova slika. Neka je S_B skup svih mogućih isječaka zadane veličine koji se mogu izrezati iz ulaznog predloška. Prvi korak je slučajnim odabirom uzeti određeni broj isječaka iz skupa S_B i popločavanjem stvoriti prvu verziju nove slike (slika 2.3, druga podslika u nizu). Popločavanje se obavlja tako da se mali dijelovi površine isječaka međusobno preklapaju. Sada treba modificirati postupak da granice među isječcima ne budu toliko vidljive. Po svakom području preklapanja pretražuje se skup S_B za blok koji najbolje 'sjeda' među susjedne isječke. Rezultat izgleda znatno bolje nakon ovog koraka (slika 2.3, treća podslika), no još uvijek su prisutne nepravilnosti koje treba riješiti. Dobar trik je primjena algoritma za pronalazak najkraćeg puta površine pogreške kroz područja preklapanja, što će biti objašnjeno u nastavku. Pronađeni put se postavi kao granica između dva susjedna bloka i sintetizirana slika je gotova (slika 2.3, zadnja podslika).

Ideja je da se rez napravi na mjestu gdje je greška preklapanja među isječcima najmanja. Ako su B_1 i B_2 dva susjedna bloka u istom redu s područjima preklapanja po vertikalnom rubu B_1^{ov} i B_2^{ov}, tada je površina pogreške $e_{1,2}$ definirana kao

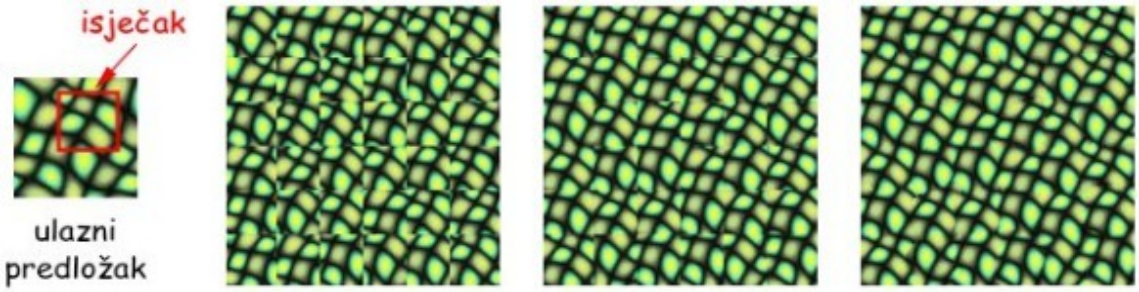
$$e_{1,2}=\left(B_1^{ov}-B_2^{ov}\right)^2. \tag{2.3}$$

Da bismo našli minimalni vertikalni rez kroz tu površinu, računamo kumulativnu minimalnu pogrešku $E_{i,j}$ po formuli

$$E_{i,j}=e_{1,2}+min\left(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1}\right), i=2..N. \tag{2.4}$$

Minimalna vrijednost zadnjeg reda označava kraj minimalnog vertikalnog puta, i tada se lako može rekonstruirati putanja po kojoj će se napraviti rez. Analogno se pronađe putanja za horizontalne rezove.

Veličina bloka je jedini parametar kojeg određuje korisnik. Kao i za prvi navedeni algoritam preoblikovanja, vrlo je važno za ispravan rad ovog algoritma odabrati adekvatnu veličinu kojom će se 'texel' obuhvatiti u potpunosti.



Slika 2.3 (preuzeta s: https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/papers/efros-siggraph01.pdf) Ulazni predložak teksture s jednim primjerom isječka za popločavanje prikazan je na prvoj slici s lijeva, a algotiram popločavanja u osnovnim koracima na sljedećim trima slikama.

2.2 Neuronski algoritam umjetničkog stila

Kada je iz malog ulaznog predloška generirana slika teksture željene veličine, na red dolazi prijenos iste na proizvoljnu fotografiju. Prethodno opisani neparametrizirani algoritmi (poglavlje 2.1) statističke podatke sintetizirane teksture i ciljne slike uspoređuju na poprilično niskoj razini piksela, pa se kod

generiranja nove slike oslanjaju na korespondenciju značajki poput intenziteta slike, kutova lokalne orijentacije slike, (Wei, 2009.) isl. Model Markovljevih slučajnih polja pokazao je zadovoljavajuće rezultate koristeći takve niskorazinske značajke, no konvolucijske mreže mogu provesti analizu slike na znatno višoj razini. Dvije različite teksturirane slike ljudsko oko će klasificirati kao istu teksturu ako se određene prikladne statističke mjere tih slika poklapaju (Julesz, 1962.). Te mjere mogu se pohraniti u obliku tzv. mapa značajki, odnosno određenih korelacijskih odnosa između tih mapa. Prijenos teksture, tj. stvaranje nove slike obavlja se uz ograničenja koja statistički podaci dviju ulaznih slika postavljaju da bi generirana slika zadržala značajke koje istovremeno odgovaraju i slici izvora teksture i slici koja nosi željeni sadržaj. Temeljni preduvjet ovog pristupa jest pronaći prikladnu reprezentaciju slike koja može modelirati statistička svojstva sadržaja i teksture neovisno jedno o drugome, što je poprilično težak problem računalnog vida. Međutim, značajni napredak dubokih konvolucijskih neuronskih mreža do kojeg je nedavno došlo, omogućio je razvoj računalnih sustava koji su sposobni naučiti izdvajati visokorazinske semantičke elemente iz prirodnih slika. Trenirane na specifičnim zadacima poput prepoznavanja objekata, s dovoljnim brojem unaprijed klasificiranih podataka, konvolucijske mreže razviju parametrizirane reprezentacije značajki koje se mogu kasnije primijeniti na raznim drugim problemima. Između ostaloga, tu spada i prijenos stila na digitalne fotografije (Gatys, 2015.).

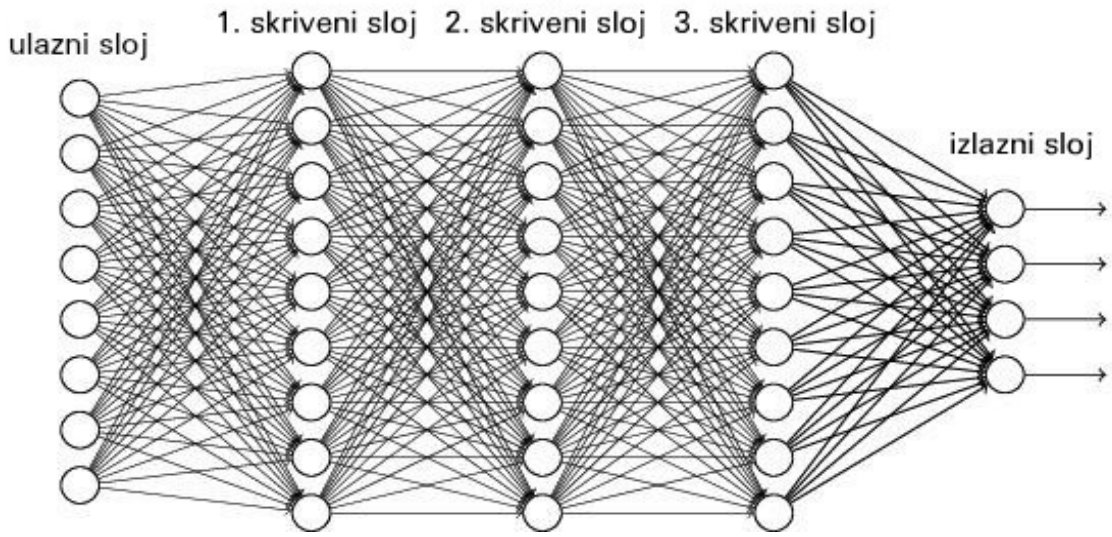
2.2.1 Konvolucijska neuronska mreža

Umjetna neuronska mreža sastoji se od međusobno spojenih osnovnih jedinica, neurona. Jedan neuron na ulaz prima m ulaznih vrijednosti $x_1, x_2, \ldots, x_m$ i

računa izlaznu vrijednost $y=f(z)$, pri čemu je z težinska suma svih ulaznih vrijednosti, uz dodani nelinearni element, pomak w_0

$$z=w_0+\sum_{i=1}^{m} w_i x_i. \tag{2.5}$$

Funkcija f zove se aktivacijska funkcija. Poželjno je da aktivacijska funkcija bude derivabilna, jer bi to mogao biti uvjet za treniranje mreže, tj. učenje težinskih koeficijenata. Stoga najčešće korištena aktivacijska funkcija je sigmoidalna funkcija. Neuroni su organizirani u slojeve, pa se prvi sloj naziva ulazni sloj, posljednji je izlazni, a svi slojevi između su tzv. skriveni slojevi. Mreža se smatra dubokom ako ima tri ili više skrivenih slojeva. Na slici 2.4 je prikazana pojednostavljena struktura duboke mreže. Učenje, odnosno treniranje mreže odnosi se na određivanje težinskih parametara mreže $w_{0,}w_1,w_2,\ldots,w_m$, koji su osnova za posao koji mreža obavlja. Skup za učenje sastoji se od parova (x_n,t_n) pri čemu se ulazni podaci označavaju s x, a očekivane izlazne vrijednosti s t. Učenje počinje inicijalizacijom težina na početne vrijednosti, te iterativnim podešavanjem istih tako da se minimizira neka funkcija koja mjeri pogrešku između očekivanih i izračunatih izlaznih podataka, primjerice funkcija srednje kvadratne pogreške (*eng. mean squared error, MSE*). Ova se minimizacija može obavljati do proizvoljnog praga preciznosti, no treba imati na umu da pri tome mreža može postati 'štreber', odnosno podesiti parametre tako da daju izrazito dobre rezultate za skup za učenje, ali jako loše rezultate za bilo koje druge podatke (*eng*. *overfitting*). Iz tog razloga se obično skup za učenje razdijeli na stvarni skup za učenje, na kojem se doista podešavaju parametri, te na skup za validaciju parametara koji su izračunati uz prvi skup. Na ovaj način umjetne neuronske mreže sposobne su koristeći znanje poznato od prije naučiti generalizirati razne probleme na novim podacima i obavljati mnoge poslove u polju umjetne inteligencije.



Slika 2.4 (preuzeta s: https://stats.stackexchange.com/questions/234891/difference-between-convolution-neural-network-and-deep-learning) Umjetna neuronska mreža. Krugovi predstavljaju pojedinačne neurone, a linije između neurona su veze kojima se izlazni podaci jednog prosljeđuju na ulaz drugog neurona.

Konvolucijska mreža je modifikacija neuronske mreže prilagođena da prima slike kao ulazne podatke. Bazirana je na diskretnoj konvoluciji, te pohranjuje prostorne informacije između piksela slike. Prije ulaska u detalje rada mreže, dobro bi bilo obraditi pojam konvolucije.

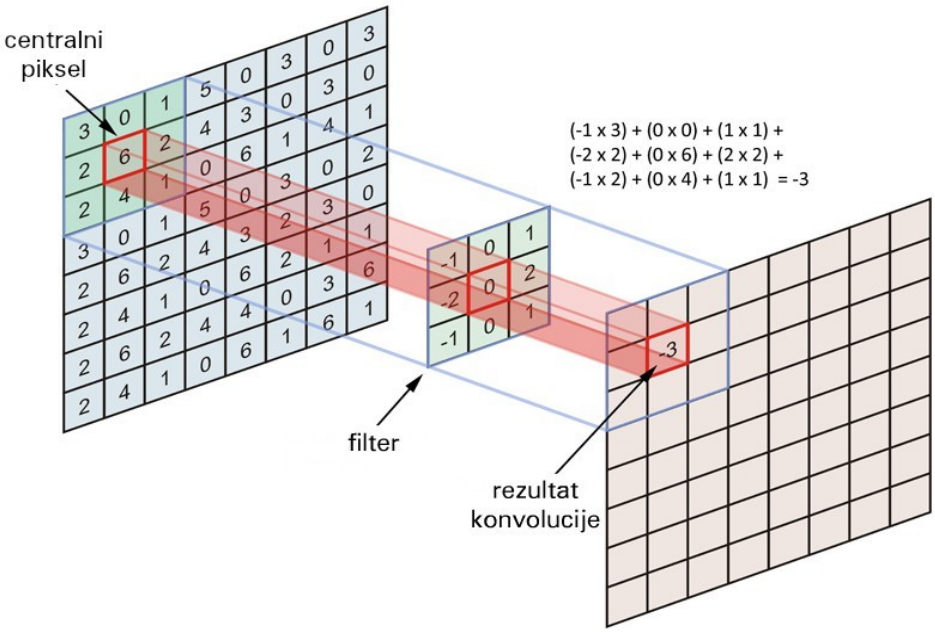
Uzmimo crno-bijelu sliku I kao 2D matricu veličine $n_1 \times n_2$

$$I : \{1, \dots, n_1\} \times \{1, \dots, n_2\} \to W \subseteq R, (i, j) \to I_{i,j}. \tag{2.6}$$

Diskretna konvolucija piksela na poziciji (r, s) slike I uz filter K dana je izrazom

$$(I * K)_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u, s+v} \tag{2.7}$$

pri čemu je K također 2D matrica veličine $(2h_1+1) \times (2h_2+1)$. Filter je zapravo matrica težinskih koeficijenata koji određuju koji će pikseli i u kojoj mjeri utjecati na promjenu vrijednosti centralnog piksela (slika 2.5).



Slika 2.5 (preuzeta s: https://stats.stackexchange.com/questions/114385/what-is-the-difference-between-convolutional-neural-networks-restricted-boltzma) Ilustracija diskretne konvolucije na konkretnom primjeru piksela označenog crvenim kvadratom uz filter veličine 3 x 3. Na ovaj način se sastavlja mapa značajki koja se prosljeđuje dalje kroz mrežu.

Konvolucijska mreža može se sastojati od nekoliko vrsta slojeva. Osnovni sloj je konvolucijski (*eng. convolutional layer*). Nazovimo ga l. Ovaj sloj na ulaz prima $m^{(l-1)}$ mapa značajki od prethodnog sloja. U slučaju kad je $l=1$ odnosno kad razmatramo ulazni sloj, tada je $m^{(l-1)}=1,$ odnosno ulaz je slika I. Izlaz konvolucijskog sloja sastoji se od $m^{(l)}$ mapa značajki, koje se računaju na sljedeći način (sljedeća formula se odnosi na $i-tu$ mapu $Y_i^{(l)}$):

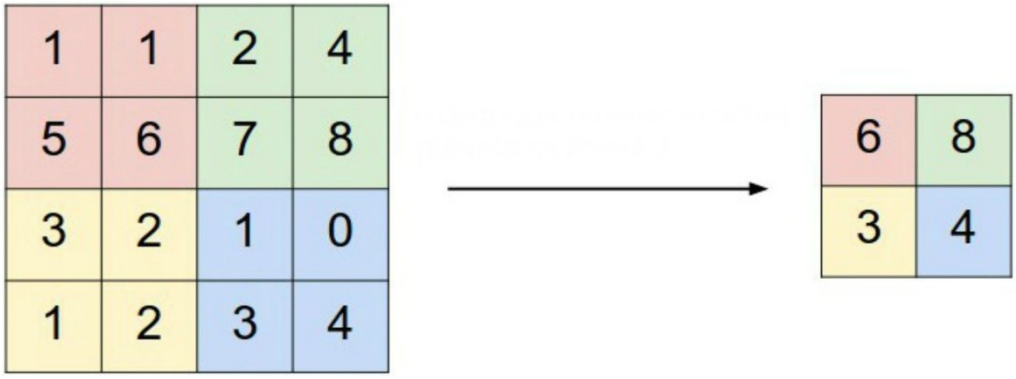
$$Y_i^{(l)}=\sum_{j=1}^{m^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)} \qquad (2.8)$$

gdje je $K_{i,j}^{(l)}$ filter koji povezuje $i-tu$ mapu značajki sloja l s $j-tom$ mapom iz prethodnog sloja. Obično se za jednu mapu značajki koristi isti filter, odnosno $K_{i,j}^{(l)}=K_{i,k}^{(l)}, j\neq k$. Vrijednosti elemenata u matricama svih filtera su upravo oni težinski parametri koje mreža određuje učenjem na skupu parova ulaza i očekivanih izlaza $\left(x_n, t_n\right)$. Korak konvolucije je još jedan parametar koji određuje za koliko piksela se pomiče filter po mapi nakon svake konvolucije.

Ideja konvolucijske mreže je da filteri u višim konvolucijskim slojevima obuhvaćaju veći prostorni opseg, dok pritom nije poželjno da veličina samog filtera raste, jer time raste broj parametara mreže. Stoga se obično izlaz konvolucijskog sloja spaja na ulaz sloja sažimanja (*eng. pooling layer*). Označimo sada s l sloj sažimanja koji dolazi nakon konvolucijskog sloja koji mu je na ulaz poslao $m^{(l-1)}$ mapa značajki. Izlaz sloja l je $m^{(l)}=m^{(l-1)}$ istih mapa u smanjenoj veličini. Općenito, sažimanje se provodi prolaskom kliznog prozorčića fiksne veličine kroz cijelu mapu pritom zadržavajući samo jednu vrijednost od svih vrijednosti piksela obuhvaćenih prozorčićem. Prema načinu određivanja vrijednosti piksela razlikujemo dvije osnovne metode sažimanja: sažimanje najvećom vrijednosti i sažimanje prosječnom vrijednosti u prozorčiću. Prvom spomenutom metodom se konvergencija prilikom učenja parametara mreže postiže znatno brže, (Stutz 2014.). Također moguće je podesiti vrijednost koraka za koji će se prozorčić pomicati u postupku, pa se prema tome prolazak može obavljati tako da se područja koja prozorčić obuhvaća ne preklapaju (slika 2.6), ili tako da se u svakom sljedećem koraku prozor pomakne za vrijednost manju od same veličine prozorčića, pri čemu dolazi do preklapanja. Praksa je pokazala da se kod sažimanja uz preklapanje smanjuje šansa da pri učenju mreža postane 'štreber' (Stutz 2014.).

Nelinearni sloj je još jedna vrsta sloja koji na izlaz šalje jednak broj mapa značajki jednakih veličina kao one koje prima na ulaz uz to da provede

aktivacijsku funkciju nad elementima mapa. Sloj rektifikacije (skraćeno ReLU) je podvrsta nelinearnog sloja koji na izlaz šalje sve pozitivne elemente mape, a negativne vrijednosti zamijeni s 0 (drugim riječima aktivacijska funkcija je $f(x)=max(0,x)$). Valja spomenuti još i potpuno povezani sloj koji spaja svaki neuron prethodnog sloja sa svakim neuronom sljedećeg sloja, te softmax sloj koji sve ulazne vrijednosti transformira u vrijednosti unutar intervala $(0,1)$. Ova dva sloja najčešće se primjenjuju na problemima klasifikacije slika, no za sintezu teksture nisu potrebni, pa se i ne koriste.



Slika 2.6 (preuzeta s: https://stats.stackexchange.com/questions/114385/what-is-the-difference-between-convolutional-neural-networks-restricted-boltzma) Primjer sažimanja mape najvećom vrijednosti piksela uz veličinu prozora 2 x 2 i korak od 2 piksela, bez preklapanja.

2.2.2 Reprezentacija sadržaja i stila slike

Pogledajmo sada kako točno definirati reprezentaciju sadržaja na slici i stila u kojem je sadržaj prikazan. Ovo je potrebno da bi se statistički podaci ulazne fotografije i umjetničke slike pohranili u precizan matematički kalup, pa se finalna slika može generirati jednostavno uz ograničenje da se njene reprezentacije sadržaja i stila istovremeno poklapaju s objema reprezentacijama

ulaznih slika.

Reprezentacija sadržaja, tj. objekata na slici konvolucijskog modela bazirana je na tzv. mapama značajki. Slojevi mreže sadrže različite filtere koji iz ulazne slike izvlače različita statistička svojstva. Sa svakim sljedećim slojem informacije koje se izvlače iz slike postaju sve složenije. Filteri mreže su zapravo oni nepoznati parametri koje je mreža prethodno naučila na podacima iz domene klasifikacije objekata. Odzivi slike na te filtere su mape značajki. Sve što je potrebno je generirati novu sliku takvu da ima približno jednake mape značajki kao i izvorna slika sadržaja, odnosno da vrijednost funkcije pogreške (razlike između mapa značajki tih dviju slika) bude što manja. Ovo se može ostvariti nekim algoritmom za minimizaciju.

Označimo vektoriziranu ulaznu fotografiju koju želimo stilizirati s $\vec{p}$, te novu sliku koju trebamo izgenerirati s $\vec{x}$. Da bismo dobili reprezentaciju sadržaja ulazne slike, prvo ju trebamo poslati kroz konvolucijsku mrežu. U svakom sloju l mreža izračuna odzive filtera kojima se formira skup mapa značajki. Za N_l filtera jedan sloj stvori N_l mapa, svaka veličine M_l kad se vektorizira. Te mape mogu se pohraniti u 2D matricu F^l veličine $N_l \times M_l$, pri čemu $F^l_{j,k}$ označuje odziv $j-tog$ filtera na poziciji k u sloju l. Neka je mapa značajki u sloju l ulazne slike P^l, a generirane slike F^l. Tada se srednja kvadratna pogreška u jednom sloju između mapa značajki, tj. reprezentacija sadržaja tih dviju slika, može definirati formulom

$$E^l_{sadržaj}(\vec{p}, \vec{x}) = \frac{1}{4 N_l M_l} \sum_{i,j} \left(F^l_{i,j} - P^l_{i,j} \right)^2, \tag{2.9}$$

a ukupna pogreška cijelih slika je težinska suma pogrešaka po svim slojevima

$$E_{sadržaj}(\vec{p}, \vec{x}) = \sum_{l=1}^{L} w_l E^l_{sadržaj}(\vec{p}, \vec{x}) \tag{2.10}$$

pri čemu težine određuju utjecaj pojedinog sloja u sintezi.

Prikaz teksture, tj. stila bit će nešto drugačiji. Pošto ove mape značajki pohranjuju određene prostorne informacije, što za ovu reprezentaciju nije baš poželjno, rezimirana statistička svojstva potrebna za sintezu teksture računaju se iz korelacija između mapa u različitim slojevima mreže. Novoj slici izmjenjujemo vrijednosti piksela sve dok korelacije između njenih mapa značajki ne budu jednake onima koje je konvolucijska mreža izračunala iz predloška teksture. Korelacijske odnose među mapama opisujemo pojmom Gramove matrice.

Neka je vektorizirana ulazna umjetnička slika označena s $\vec{t}$, te njena odgovarajuća Gramova matrica T^l veličine $N_l \times N_l$, pri čemu je N_l i dalje broj filtera u sloju l. Tada je element $T^l_{i,j}$ definiran formulom

$$T^l_{i,j} = \sum_{k=1}^{M_l} V^l_{i,k} V^l_{j,k}, \tag{2.11}$$

pri čemu su V^l odgovarajuće mape značajki teksture izračunate u sloju l, a M_l predstavlja veličinu jedne vektorizirane mape. Skup Gramovih matrica $\left\{T^1, T^2, \ldots, T^L\right\}$, gdje je L ukupan broj slojeva u mreži, predstavlja kompletnu reprezentaciju koja pruža potreban stacionarni opis željene teksture. Dalje, označimo li još i Gramovu matricu generirane slike $\vec{x}$ u sloju l s G^l, funkcija pogreške među ove dvije matrice dana je formulom

$$E^l_{tekstura}\left(\vec{t}, \vec{x}\right) = \frac{1}{4 N_l^2 M_l^2} \sum_{i,j} \left(G^l_{i,j} - T^l_{i,j}\right)^2, \tag{2.12}$$

dok je ukupna pogreška, analogno kao i za pogrešku među reprezentacijama sadržaja

$$E_{tekstura}\left(\vec{t}, \vec{x}\right) = \sum_{l=1}^{L} w_l E^l_{tekstura}\left(\vec{t}, \vec{x}\right). \tag{2.13}$$

Na ovaj način dobivene su parametrizirane reprezentacije koje modeliraju sadržaj i stil neovisno jedno o drugome, pa se u generacijskom procesu nova

slika istovremeno pokušava ugoditi sa statističkim značajkama sadržaja ulazne fotografije i teksture ulazne umjetničke slike. Drugim riječima, ključ algoritma je minimizirati obje funkcije pogreške $E_{sadržaj}$ i $E_{tekstura}$ mijenjajući vrijednosti piksela nove slike.

2.2.3 Prijenos stila na digitalnu fotografiju

Sad kad imamo prikladne reprezentacije i za sadržaj i za stil koje modeliraju sliku neovisno jedno o drugome, možemo započeti prijenos stila. Predstavimo Neuronski algoritam umjetničkog stila.

Uzmimo da je ulazna slika koja predstavlja predložak teksture (u našem slučaju to je odabrano umjetničko djelo) označena s $\vec{t}$, a ulazna fotografija s objektima koje želimo prikazati u danom stilu označena je s $\vec{p}$. Nova slika $\vec{x}$ koju generiramo inicijalizirana je nasumičnim vrijednostima piksela (*eng. white noise image*). Zadatak algoritma je u iterativnom postupku mijenjati vrijednosti piksela nove slike dok se ne zadovolji neki uvjet zaustavljanja. To može biti konačan broj iteracija ili maksimalan dozvoljeni prag pogreške između reprezentacija stila i sadržaja sintetizirane slike i Gramovih matrica ulazne umjetničke slike, odnosno mapa značajki ulazne fotografije. U tu svrhu, ukupna pogreška, ili funkcija koja se minimizira, glasi

$$E_{ukupna}(\vec{p},\vec{t},\vec{x}) = \alpha E_{tekstura}(\vec{t},\vec{x}) + \beta E_{sadržaj}(\vec{p},\vec{x}). \qquad (2.14)$$

Pri tome su α i β težinski faktori koji određuju utjecaj stila, odnosno sadržaja u sintezi, što znatno obilježava i finalni rezultat.

Navedeni postupak se može obaviti nekim numeričkim algoritmom optimizacije. Popularan algoritam u ovom području je BFGS algoritam

(Broyden–Fletcher–Goldfarb–Shanno), iterativna tehnika rješavanja nelinearnih optimizacijskih problema koja spada u klasu kvazi-Newtonovih metoda (Wikipedia, Quasi-Newton method). Za ovaj problem minimizacije funkcije $E_{ukupna}(\vec{p},\vec{t},\vec{x})$ odabrana je njegova inačica L-BFGS-B (Zhu, 1994.). L (nekad se zapisuje i kao LM), skraćeno za limited memory, označava rekurzivnu modifikaciju prilagođenu ograničenom memorijskom resursu računala, a posljednje B u kratici stoji za simple box ograničenje na varijable funkcije koja se minimizira, u ovom slučaju to se odnosi na vrijednosti koje piksel može poprimiti. Pogledajmo prvo što je općenito Newtonova metoda (Weisstein, 2005.), zatim u čemu se razlikuje od kvazi-Newtonove metode, i na kraju što je i kako radi BFGS algoritam.

Radi jednostavnosti označit ćemo ukupnu funkciju pogreške $E_{ukupna}(\vec{p},\vec{t},\vec{x})$ s $f(\vec{x})$. Ta funkcija u principu ovisi samo o pikselima izlazne slike $\vec{x}$ pošto su ulazne slike nepromjenjive, pa su time $\vec{p}$ i $\vec{t}$ konstante. Newtonova metoda koristi se za aproksimaciju korijena ili nul-vrijednosti funkcije $f(\vec{x})$. Ovo se nadovezuje na problem minimizacije, jer se minimalna vrijednost funkcije (ako postoji) dobije upravo u njenim stacionarnim točkama, tj. vrijednostima u kojima je gradijent te funkcije jednak 0. Počevši od inicijalnih vrijednosti rješenja $\vec{x_0}$, u svakoj iteraciji se izračuna vektor smjera u kojem se $\vec{x}$ pomiče da bi bio bliže stvarnim nul-vrijednostima. Metoda gradi kvadratni model funkcije f aproksimiran Taylorovim redom drugog stupnja

$$g(\vec{x}+\vec{\tau}) = f(\vec{x}) + \nabla f(\vec{x})^T \cdot \vec{\tau} + \frac{1}{2!} \cdot \vec{\tau}^T \cdot \nabla^2 f(\vec{x}) \vec{\tau}. \qquad (2.15)$$

Traženi vektor smjera označen je s $\vec{\tau}$, a $g(\vec{x}+\vec{\tau})$ se promatra kao funkcija od $\vec{\tau}$, pri čemu se traži takav $\vec{\tau}$ da minimizira funkciju g. Derivacijom te funkcije i izjednačavanjem s 0, dobivamo sljedeću jednadžbu

$$\nabla g = \nabla f(\vec{x}) + H(f) \cdot \vec{\tau} = 0. \tag{2.16}$$

Ovdje je $H(f) = \nabla^2 f(\vec{x})$ Hesseova matrica drugih parcijalnih derivacija po svim parovima varijabli funkcije f. Iz toga slijedi

$$\vec{\tau} = -H(f)^{-1} \cdot \nabla f(\vec{x}) \tag{2.17}$$

nakon čega je samo još preostalo ažurirati vrijednosti rješenja

$$\vec{x} \Leftarrow \vec{x} + \lambda \cdot \vec{\tau} \tag{2.18}$$

pri čemu λ označava prikladan pomak kojim će se funkcija pogreške maksimalno minimizirati u trenutnom koraku[1]. Opisani postupak iterativno se ponavlja dok se ne dosegne željeni prag preciznosti rješenja $\vec{x}$.

Osnovni problem Newtonove metode je složenost izračuna Hesseove matrice $H(f)$, u čemu je pronađena motivacija za razvoj tzv. kvazi-Newtonovih metoda. Umjesto da u svakoj iteraciji računamo Hesseovu matricu i njen inverz, računat ćemo Hesseovu aproksimaciju B takvu da zadovolji formulu

$$\nabla f(\vec{x}) + B \cdot \vec{\tau} = 0, \tag{2.19}$$

iz čega proizlazi

$$\vec{\tau} = -B^{-1} \cdot \nabla f(\vec{x}). \tag{2.20}$$

Ovo je iterativni proces koji se odvija paralelno uz aproksimaciju rješenja $\vec{x}$ funkcije f. Dakle, kreće se od početnih vrijednosti $\vec{x}_0$ i B_0 (inicijalna matrica mora biti pozitivno definitna) i u svakom koraku, nakon izračuna nove vrijednosti rješenja $\vec{x}$, ažurira se i vrijednost matrice B. Formula kojom se Hesseova aproksimacija ažurira drugačija je za svaki algoritam koji spada u skupinu kvazi-Newtonovih metoda. U BFGS algoritmu ova formula u $k-toj$

---

1 *Postoji niz strategija kojima se optimalni pomak λ može izračunati, a nazivaju se algoritmi pretraživanja u zadanom smjeru (eng. line search algorithms).*

iteraciji glasi

$$B_{k+1}=B_k+\frac{y_k\cdot y_k^T}{y_k^T\cdot\Delta\vec{x_k}}-\frac{B_k\cdot\Delta\vec{x_k}\cdot\left(B_k\cdot\Delta\vec{x_k}\right)^T}{\Delta\vec{x_k}^T\cdot B_k\cdot\Delta\vec{x_k}},\qquad(2.21)$$

pri čemu je $\Delta\vec{x_k}=\lambda\cdot\vec{\tau}=\vec{x}_{k+1}-\vec{x_k}$ pomak za koji smo u trenutnoj iteraciji 'popravili' trenutno rješenje, a $y_k=\nabla f\left(\vec{x}_{k+1}\right)-\nabla f\left(\vec{x}_k\right)$ razlika između aproksimiranih gradijenata. Analogno, i inverz $H=B^{-1}$ se može direktno ažurirati po formuli

$$H_{k+1}=\left(I-\frac{\Delta\vec{x_k}\cdot y_k^T}{y_k^T\cdot\Delta\vec{x_k}}\right)\cdot H_k\cdot\left(I-\frac{y_k\cdot\Delta\vec{x_k}^T}{y_k^T\cdot\Delta\vec{x_k}}\right)+\frac{\Delta\vec{x_k}\cdot\Delta\vec{x_k}^T}{y_k^T\cdot\Delta\vec{x_k}},\qquad(2.22)$$

da se izbjegne potreba za izračunom inverza iz B u svakoj iteraciji.

Utvrđeno je da opisani algoritam BFGS među svim drugim optimizacijskim algoritmima radi najbolje za sintezu teksture (Gatys, 2015.). Iteracije se provode dok vrijednost $f\left(\vec{x}_k\right)$ ne dosegne neki definirani prag pogreške koji se može tolerirati, ili dok program ne obavi maksimalan dozvoljeni broj iteracija.

3 Analiza programskog rješenja

Programski kod priložen u ovom radu koji implementira Neuronski algoritam umjetničkog stila preuzet je s GitHub repozitorija korisnika $log0^2$, te je na nekim dijelovima izmijenjen. Kod je pisan u programskom jeziku Python (verzija 3.5, uz vanjske pakete NumPy i ScyPy za složene matematičke operacije), a biblioteka korištena za rad s konvolucijskom mrežom je TensorFlow. Naučeni model mreže koji obavlja sintezu slike zove se model VGG-19. Prije analize same programske implementacije slijedi nekoliko uvodnih riječi o navedenim alatima, biblioteci i modelu.

3.1 Biblioteka TensorFlow

TensorFlow je open source biblioteka za strojno učenje, izgradnju i treniranje neuronskih mreža. Razvio ju je Google Brain tim s namjerom da detektira, istraži i implementira uzorke i korelacije procesa analognih onima u ljudskom mozgu (Wikipedia, TensorFlow), te po uzoru na čovjekove kognitivne funkcije, stvori umjetni računarski sustav koji može obavljati zahtjevne intelektualne poslove poput računalnog vida ili razumijevanja govora. Prvenstveno je TensorFlow trebao biti korišten iznimno unutar Google kompanije, ali su ga odlučili objaviti

2*https://github.com/log0/neural-style-painting*

pod Apache 2.0 open source licencom 9. studenog, 2015. godine. Pisan u jezicima Python i C++ na platformi CUDA, TensorFlow je dostupan na 64-bitnim Linux, Windows i macOS sustavima, te mobilnim sustavima Android i iOS, a uz njega su dostupna aplikacijska programska sučelja za Python, C++, Haskell, Javu, Go i Rust. Može se pokretati na višestrukim CPU-ima i GPU-ima, uz opcionalne CUDA ekstenzije za rad na grafičkim jedinicama.
U nastavku ću navesti i ukratko objasniti nekoliko osnovnih TensorFlow razreda i modula prilagođenih za programski jezik Python, koji se pojavljuju u kodu za Neuronski algoritam umjetničkog stila. Sve ostale mogućnosti i alati koje sadrži ova biblioteka dostupni su u dokumentaciji na službenoj TensorFlow web stranici.

- *Razred tensorflow.Graph*

Razred koji predstavlja model za obradu podataka u obliku grafa, a sastoji se od računalnih jedinica (*tensorflow.Operation*) i podatkovnih jedinica, odnosno tenzora (*tensorflow.Tensor*). U Pythonu se graf može izraziti rječnikom. Jedan čvor grafa odgovara jednom elementu rječnika i sadrži jednu računalnu jedinicu koja prosljeđuje podatkovne jedinice kroz graf.

- *Razred tensorflow.Operation*

Razred koji predstavlja pojedinu operaciju. Na ulaz prima nijedan ili više tenzora s podacima, obrađuje ih definiranom operacijom i na izlaz šalje obrađene podatke također kao tenzore. Povezivanjem izlaznih tenzora jedne operacije na ulaz druge operacije stvara se tok podataka koji omogućava izvršavanje cijelog grafa. Objekti ovog razreda stvaraju se pozivima Pythonovih operacijskih konstruktora, poput *tensorflow.matmul(...),* metode koja stvara operaciju množenja matrica danih u argumentima.

- *Razred tensorflow.Tensor*

Razred koji predstavlja podatke kojima operacije barataju, pri čemu zapravo ne sadrži same vrijednosti tih podataka, već služi samo kao omotač podataka. Na ovaj način se korisnika potiče na razvoj složenih računskih operacija koje grade potpun graf. Zahvaljujući tome, umjesto da izvršava jednu po jednu operaciju, TensorFlow može izvršiti graf u cjelosti mnogo efikasnije.

- *Razred tensorflow.Session*

Razred koji enkapsulira kompletno okruženje u kojem se izvršavaju operacije predstavljene *Operation* objektima i evaluiraju podaci predstavljeni *Tensor* objektima u jednom grafu. Osnovna metoda je *Session.run(...)* koja pokreće jedan korak zadanog postupka računanja definiranog argumentom kojeg prima. Argument je uobičajno jedan ili više zadanih čvorova grafa.

- *Razred tensorflow.Variable*

Razred koji sadrži parametre koji se optimiraju pozivom operacija treniranja. Za razliku od tenzora, objekti ovog razreda ostaju sačuvani i nakon poziva metode *Session.run(...).* Sadržaj varijable predstavljen je *Tensor* objektom. Pri konstrukciji *Variable* objekta potrebno je postaviti inicijalnu vrijednost varijabli koja se kasnije može promijeniti metodom *assign(value)*.

- *Modul tensorflow.nn*

Podrška za implementaciju neuronske mreže. Sadrži gotove metode za izvođenje operacija koje određeni slojevi obavljaju, poput konvolucije (*conv1d(...), conv2d(...), conv3d(...)*), sažimanja (*avg_pool(...), max_pool(...)*), rektifikacije (*relu(...)*), i drugih. Operacije se obavljaju nad *Tensor* objektima koje metode primaju kao argumente.

- *Modul tensorflow.train*

Podrška za treniranje i optimizaciju modela. Sadrži razrede koji implementiraju različite optimizacijske algoritme. Ti razredi imaju metodu *minimize(...)* koja kao argument prima *Tensor* objekt čija se vrijednost minimizira ažuriranjem podataka modela, a vraća *Operation* objekt s potrebnim operacijama za minimizaciju. Ako se varijable ne pošalju kao argument ovoj metodi, tada se uzimaju i ažuriraju podaci koji su prethodno bili referencirani kao varijable.

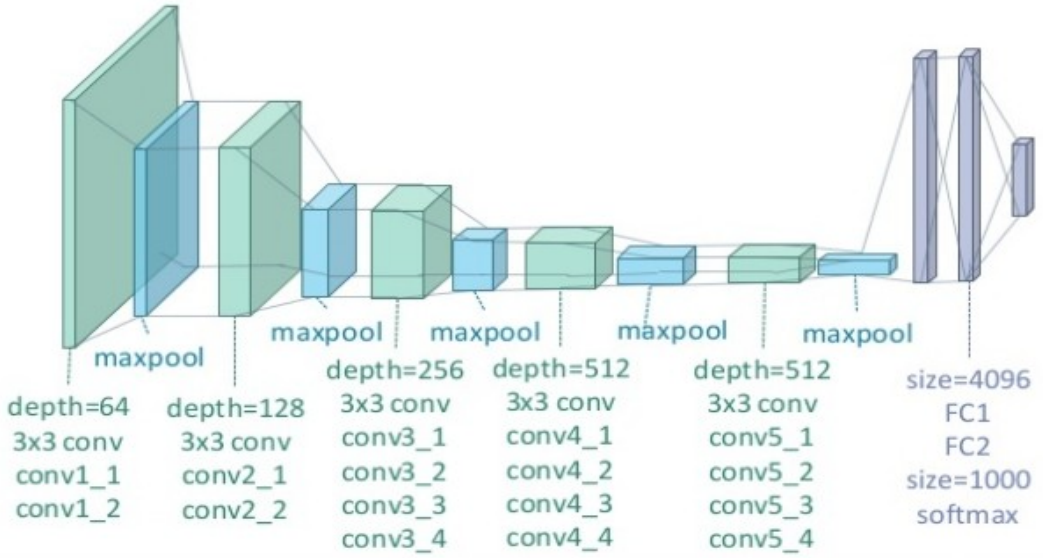
3.2 Model VGG-19

Grupa VGG sa Sveučilišta u Oxfordu razvila je nekoliko konfiguracija vrlo dubokih konvolucijskih mreža za širokosežnu klasifikaciju slika (*eng. large-scale image recognition*). Broj slojeva u njihovim mrežama kreće se između 11 i 19. Za predstavljeni algoritam prijenosa umjetničkog stila (Gatys, 2015.) odabran je model s 19 slojeva (slika 3.1), među kojima je 16 konvolucijskih i 5 slojeva sažimanja (Simonyan, 2015.). Ovim modelom se pokazalo koliko je dubina (tj. broj slojeva) značajan parametar za precizniju i djelotvorniju arhitekturu konvolucijske mreže. Mada postupno dodavanje konvolucijskih slojeva povlači pitanje efikasnosti i brzine mreže, ovo je ipak izvedivo zahvaljujući maloj dimenziji filtera korištenih u svim konvolucijskim slojevima. Izabrana je veličina filtera 3×3, jer je to najmanja moguća veličina koja obuhvaća sve četiri strane oko centralnog piksela (lijevo, desno, iznad i ispod). Korak konvolucije postavljen je na 1 piksel, tako da bi prostorna rezolucija ostala sačuvana nakon prolaska kroz konvolucijski sloj. Prostorno sažimanje se u originalnom modelu obavlja kroz pet slojeva sažimanja najvećom vrijednosti, no za ovaj zadatak sinteze teksture praksa je pokazala da sažimanje srednjom vrijednosti daje

nešto bolje rezultate (ovo se može vidjeti na slici 3.2). Veličina prozorčića izabrana za sažimanje je $2\times 2,$ uz korak 2, bez preklapanja. Svi konvolucijski slojevi opremljeni su operacijom rektifikacije, čime se svaki piksel koji nakon konvolucije ima negativnu vrijednost postavi na nulu kao minimalnu dozvoljenu vrijednost.

Gledajući sliku 3.1, primjećujemo da je arhitektura mreže konstruirana tako da se nekoliko konvolucijskih slojeva pojavljuje uzastopno jedan za drugim. Očigledno je da bi za rezultat konvolucije jednako efektivno bilo, a naizgled se možda čini i jednostavnije, umjesto primjerice dva uzastopna konvolucijska sloja s filterima veličine 3×3 staviti jedan sloj s filterom $5\times 5,$ no postoji razlog zašto je prvo rješenje bolje. Prvo, u mrežu se pripajaju 2 ReLU sloja umjesto samo jednoga. Drugo, i još važnije, dva sloja s manjim filterom imaju manje parametara nego jedan sloj s većim filterom. Uzmemo li konkretno u ovom primjeru da slika ima C kanala, tada prvo rješenje sadrži $2\cdot\left(3C\right)^2=18C^2$ parametara, dok ih drugo rješenje ima $\left(5C\right)^2=25C^2.$

Izvorno ovaj model mreže sadrži i dva potpuno povezana i jedan softmax sloj na samom kraju, no ti slojevi služe za klasifikaciju slika, stoga se u slučaju sinteze teksture izostavljaju.



Slika 3.1 (preuzeta s: https://www.slideshare.net/ckmarkohchang/applied-deep-learning-1103-convolutional-neural-networks) Vrlo duboka konvolucijska mreža s 19 slojeva. Konvolucijski slojevi (convx_x) prikazani u grupama, razdvojeni su slojevima sažimanja (maxpool). Dubina (depth) u konvolucijskim slojevima odnosi se na broj mapa značajki na izlazima. Slojevi nakon posljednjeg sloja sažimanja (FCx i softmax) se u ovom algoritmu ne koriste.

3.3 Implementacija algoritma i mreže

Na samom početku programskog koda postavljaju se globalne varijable koje čuvaju određene konstante vezane uz ulazne slike, te algoritam.

```
OUTPUT_DIR = 'output/'
STYLE_IMAGE = 'images/styleImage.jpg'
CONTENT_IMAGE = 'images/contentImage.jpg'
IMAGE_WIDTH = 800
IMAGE_HEIGHT = 600
COLOR_CHANNELS = 3
BETA = 5
```

```
ALPHA = 100
ITERATIONS = 1000
VGG_MODEL = 'imagenet-vgg-verydeep-19.mat'
MEAN_VALUES = np.array([123.68, 116.779, 103.939]).reshape((1,1,1,3))
NOISE RATIO = 0.6
```

Prve tri varijable, OUTPUT_DIR, STYLE_IMAGE, CONTENT_IMAGE, sadrže putanje do, redom, izlaznog direktorija, ulazne slike sadržaja i ulazne slike teksture. Nadalje, IMAGE_WIDTH, IMAGE_HEIGHT i COLOR_CHANNELS su konstante za veličinu slike (širinu i visinu u pikselima), te za broj kanala modela boja (u ovom slučaju je to RGB model). Ostale konstante odnose se na rad algoritma. APLHA i BETA su težine gubitaka stila i sadržaja u formuli za ukupnu pogrešku koju algoritam minimizira pri generiranju nove slike. Konstantom ITERATIONS postavlja se ukupan broj iteracija optimizacije. VGG_MODEL je *.mat* datoteka u kojoj se nalazi istrenirani VGG-19 model mreže, a MEAN_VALUES je polje srednjih vrijednosti slika korištenih za treniranje, koje se u početku programa oduzima od ulaznih slika, te na kraju dodaje završnoj slici. NOISE_RATIO nosi postotak šuma pri inicijalizaciji nove slike, čime se određuje koliko će sadržaj ulazne fotografije biti raspoznatljiv na početnoj *white-noise* slici. Metoda *generate_noise_image(...)* obavlja inicijalizaciju te slike.

```
def generate_noise_image(content_image, noise_ratio = NOISE_RATIO):
    noise_image = np.random.uniform(
        -20, 20
        (1, IMAGE_HEIGHT, IMAGE_WIDTH, COLOR_CHANNELS))
        .astype('float32')
    input_image = noise_image * noise_ratio
        + content_image * (1 - noise_ratio)
    return input_image
```

Također su potrebne metode za učitavanje ulaznih slika i za spremanje finalne slike (putanje datoteka određene argumentima *path*). Za ovo se koriste gotove

metode iz Pythonovih biblioteka NumPy i SciPy. VGG model očekuje da će ulaz biti umanjen za MEAN_VALUES, što se obavlja pri učitavanju, pa se završnoj slici prije pohranjivanja u datotečni sustav trebaju dodati te iste vrijednosti.

```
def load_image(path):
    image = scipy.misc.imread(path)
    image = np.reshape(image, ((1,) + image.shape))
    image = image - MEAN_VALUES
    return image

def save_image(path, image):
    image = image + MEAN_VALUES
    image = image[0]
    image = np.clip(image, 0, 255).astype('uint8')
    scipy.misc.imsave(path, image)
```

Nadalje, slijedi metoda *load_vgg_model(path)* za učitavanje naučenih parametara iz *.mat* datoteke (argument *path*), i konstrukcija mreže.

```
def load_vgg_model(path):
    vgg = scipy.io.loadmat(path)
    vgg_layers = vgg['layers']

    def _weights(layer, expected_layer_name):
        W = vgg_layers[0][layer][0][0][0][0][0]
        b = vgg_layers[0][layer][0][0][0][0][1]
        layer_name = vgg_layers[0][layer][0][0][-2]
        assert layer_name == expected_layer_name
        return W, b
    def _relu(conv2d_layer):
        return tf.nn.relu(conv2d_layer)

    def _conv2d(prev_layer, layer, layer_name):
        W, b = _weights(layer, layer_name)
        W = tf.constant(W)
        b = tf.constant(np.reshape(b, (b.size)))
```

```
        return tf.nn.conv2d(
            prev_layer, filter=W, strides=[1, 1, 1, 1],
            padding='SAME') + b

    def _conv2d_relu(prev_layer, layer, layer_name):
        return _relu(_conv2d(prev_layer, layer, layer_name))
    def _avgpool(prev_layer):
        return tf.nn.avg_pool(prev_layer, ksize=[1, 2, 2, 1],
            strides=[1, 2, 2, 1], padding='SAME')

    graph = {}
    graph['input'] = tf.Variable(np.zeros((1, IMAGE_HEIGHT,
        IMAGE_WIDTH, COLOR_CHANNELS)), dtype = 'float32')
    graph['conv1_1'] = _conv2d_relu(graph['input'], 0, 'conv1_1')
        # (3, 3, 3, 64)
    graph['conv1_2'] = _conv2d_relu(graph['conv1_1'], 2, 'conv1_2')
        # (3, 3, 64, 64)
    graph['avgpool1'] = _avgpool(graph['conv1_2'])
    graph['conv2_1'] = _conv2d_relu(graph['avgpool1'], 5, 'conv2_1')
        # (3, 3, 64, 128)
    graph['conv2_2'] = _conv2d_relu(graph['conv2_1'], 7, 'conv2_2')
        # (3, 3, 128, 128)
    graph['avgpool2'] = _avgpool(graph['conv2_2'])
    graph['conv3_1'] = _conv2d_relu(graph['avgpool2'], 10, 'conv3_1')
        # (3, 3, 128, 256)
    graph['conv3_2'] = _conv2d_relu(graph['conv3_1'], 12, 'conv3_2')
        # (3, 3, 256, 256)
    graph['conv3_3'] = _conv2d_relu(graph['conv3_2'], 14, 'conv3_3')
        # (3, 3, 256, 256)
    graph['conv3_4'] = _conv2d_relu(graph['conv3_3'], 16, 'conv3_4')
        # (3, 3, 256, 256)
    graph['avgpool3'] = _avgpool(graph['conv3_4'])
    graph['conv4_1'] = _conv2d_relu(graph['avgpool3'], 19, 'conv4_1')
        # (3, 3, 256, 512)
    graph['conv4_2'] = _conv2d_relu(graph['conv4_1'], 21, 'conv4_2')
        # (3, 3, 512, 512)
```

```
    graph['conv4_3'] = _conv2d_relu(graph['conv4_2'], 23, 'conv4_3')
        # (3, 3, 512, 512)
    graph['conv4_4'] = _conv2d_relu(graph['conv4_3'], 25, 'conv4_4')
        # (3, 3, 512, 512)
    graph['avgpool4'] = _avgpool(graph['conv4_4'])
    graph['conv5_1'] = _conv2d_relu(graph['avgpool4'], 28, 'conv5_1')
        # (3, 3, 512, 512)
    graph['conv5_2'] = _conv2d_relu(graph['conv5_1'], 30, 'conv5_2')
        # (3, 3, 512, 512)
    graph['conv5_3'] = _conv2d_relu(graph['conv5_2'], 32, 'conv5_3')
        # (3, 3, 512, 512)
    graph['conv5_4'] = _conv2d_relu(graph['conv5_3'], 34, 'conv5_4')
        # (3, 3, 512, 512)
    graph['avgpool5'] = _avgpool(graph['conv5_4'])

    return graph
```

Metoda gradi rječnik *graph* koji sadrži VGG-19 model mreže. Ključevi rječnika odgovaraju imenu i rednom broju pojedinog sloja, a vrijednosti su varijable modela na izlazima slojeva. Za ovo su korištene metode iz TensorFlow modula *tf.nn*. U priloženom isječku koda može se primijetiti kako su umjesto *maxpool* slojeva (sažimanje najvećom vrijednosti) sa slike 3.1 iskorišteni *avgpool* slojevi (sažimanje prosječnom vrijednosti). Testiranje je pokazalo da sažimanje prosječnom vrijednosti daje bolje rezultate u sintezi (slika 3.2). Metoda *_weights(layer, expected_layer_name)* iz *.mat* datoteke čita i vraća vrijednosti naučenih težina, uključujući nelinearni pomak w_0 (*eng. bias*) određenog konvolucijskog sloja zapisanog u argumentu *layer*. Preostale metode obavljaju operacije odgovarajućeg sloja, i vraćaju objekte koji nose varijable mreže, dok u argumentu primaju izlazne varijable sloja koji mu prethodi. Iznimno metode *_conv2d(...)* i *_conv2d_relu(...)* kao argument primaju redni broj i ime trenutnog sloja radi dohvaćanja potrebnih težina. Ispod svakog konvolucijskog sloja

zapisane su u komentaru dimenzije filtera tog sloja. Primjerice, (3, 3, 64, 128) znači da u tom sloju ima 128 neurona, od kojih svaki sadrži po 64 filtera veličine $3 x 3$. Potpuno povezani slojevi sa slike 3.1 su isključeni iz ove konfiguracije.

Slika 3.2 Rezultati prijenosa stila s Picassove Guernice uz prosječno (lijevo) i maksimalno sažimanje (desno). Na prvoj slici je zastupljenost karakteristika teksture u odnosu na sadržaj osjetno više ujednačena nego na drugoj slici.

Sljedeće je potrebno implementirati funkcije pogreške za teksturu $E_{tekstura}(\vec{t}, \vec{x})$ iz formule (2.13) i sadržaj $E_{sadržaj}(\vec{p}, \vec{x})$ iz formule (2.10), i ugraditi ih u model mreže. Metoda *content_loss_func(...)* računa i vraća ukupnu pogrešku za sadržaj, dok style*_loss_func(...)* čini isto za teksturu.

```
def content_loss_func(sess, model):

    def _content_loss(p, x):
        N = p.shape[3]
        M = p.shape[1] * p.shape[2]
        return (1 / (4 * N * M)) * tf.reduce_sum(tf.pow(x - p, 2))

    layers = [ ('conv1_1',1.0), ('conv2_1',1.0), ('conv3_1',1.0),
        ('conv4_1',1.0), ('conv5_1',1.0) ]
    E = [_content_loss(sess.run(model[layer_name]),model[layer_name])
        for layer_name, _ in layers]
    W = [w for _, w in layers]
    return sum([W[l] * E[l] for l in range(len(layers))])
```

```
def style_loss_func(sess, model):
    def _gram_matrix(F, N, M):
        Ft = tf.reshape(F, (M, N))
        return tf.matmul(tf.transpose(Ft), Ft)

    def _style_loss(t, x):
        N = t.shape[3]
        M = t.shape[1] * t.shape[2]
        T = _gram_matrix(t, N, M)
        G = _gram_matrix(x, N, M)
        result = (1/(4 * N**2 * M**2))*tf.reduce_sum(tf.pow(G-T,2))
        return result

    layers = [ ('conv1_1',0.5), ('conv2_1',1.0), ('conv3_1',1.5),
        ('conv4_1', 3.0), ('conv5_1', 4.0) ]
    E = [_style_loss(sess.run(model[layer_name]),model[layer_name])
        for layer_name, _ in layers]
    W = [w for _, w in layers]
    return sum([W[l] * E[l] for l in range(len(layers))])
```

Varijabla *N* u obje metode nosi broj mapi značajki u jednom sloju, a *M* veličinu pojedine vektorizirane mape, tj broj elemenata u njoj. Lista *layers* sadrži imena slojeva koji su uključeni u izračun pogreške. Ovaj dio koda može se po potrebi mijenjati. U obje metode lista *layers* se sastoji od tuple parova imena i težinskog faktora sloja čime se postavlja utjecaj pojedinog sloja na izračun pogreške. Ovi faktori značajno definiraju krajnji rezultat pošto slojevi na različitim dubinama drugačije djeluju na analizu i sintezu teksture.

Konačno, tu je glavna, main funkcija u kojoj se stvara TensorFlow sjednica (eng. session), te učitavaju model mreže i ulazne slike.

```
if __name__ == '__main__':
    with tf.Session() as sess:
        content_image = load_image(CONTENT_IMAGE)
        style_image = load_image(STYLE_IMAGE)
```

```
model = load_vgg_model(VGG_MODEL)
input_image = generate_noise_image(content_image)

sess.run(tf.initialize_all_variables())
sess.run(model['input'].assign(content_image))
content_loss = content_loss_func(sess, model)
sess.run(model['input'].assign(style_image))
style_loss = style_loss_func(sess, model)

total_loss = BETA * content_loss + ALPHA * style_loss

optimizer = tf.train.AdamOptimizer(2.0)
train_step = optimizer.minimize(total_loss)

sess.run(tf.initialize_all_variables())
sess.run(model['input'].assign(input_image))
for it in range(ITERATIONS+1):
    sess.run(train_step)
    if it == ITERATIONS:
        mixed_image = sess.run(model['input'])
        save_image('output/result.png', mixed_image)
```

Nakon inicijalizacije slike koja se treba sintetizirati, *input_image*, sjednica pokreće izgrađeni model nad ulaznim slikama da bi izračunala gubitak sadržaja i stila, a time i *total_loss*, ukupnu pogrešku $E_{ukupna}(\vec{p}, \vec{t}, \vec{x})$ iz formule (2.14). Zatim se stvara objekt *optimizer* razreda *AdamOptimizer* koji implementira algoritam stohastičke optimizacije Adam (Kingma, 2015.), što je dovoljno dobra zamjena za L-BFGS, jer TensorFlow još uvijek ne podržava BFGS algoritam. Objekt *optimizer* metodi *minimize(...)* koja vrši postupnu optimizaciju, šalje *total_loss* kao element koji se minimizira. Potom se za varijable mreže postavljaju početne vrijednosti *input_image*, i u iterativnom postupku se pokreće operacija minimizacije. Na samom kraju, u zadnjoj iteraciji završna slika se sprema i program se završava.

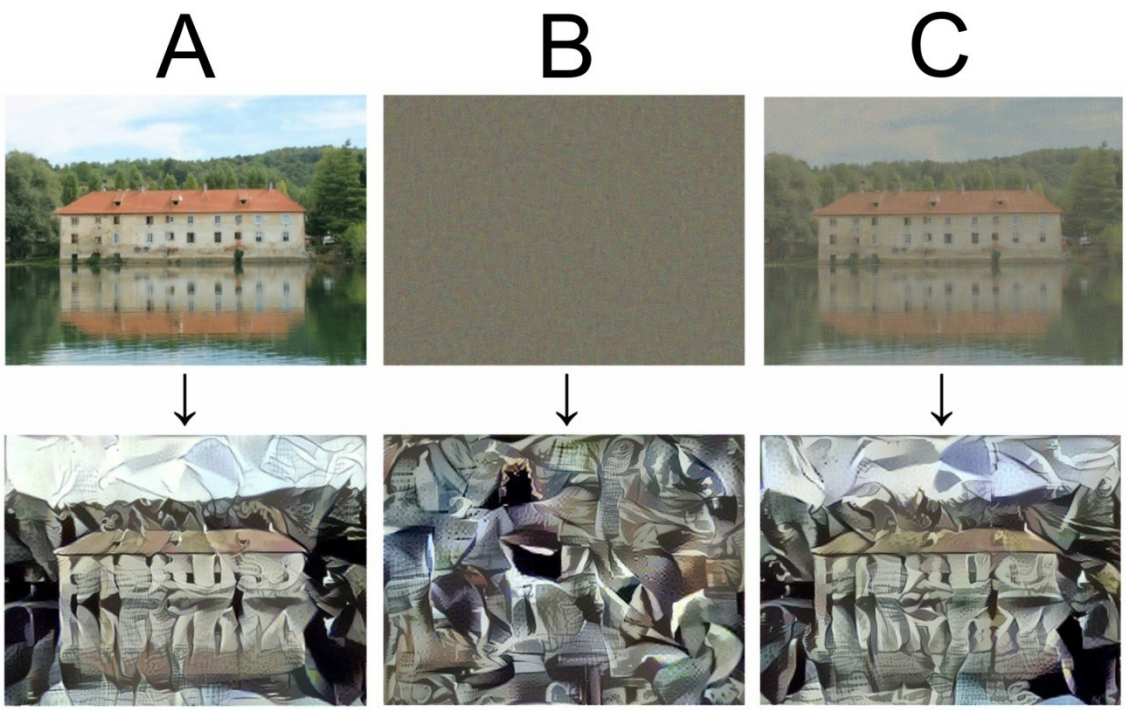
4 Ocjena rezultata

U ovom poglavlju slijedi osvrt na rezultate sinteze uz različito postavljene određene parametre u algoritmu. Testirat ćemo kako pojedine varijacije na implementaciju algoritma i model mreže utječu na završnu sliku. Za ovo ćemo radi lakše usporedbe koristiti uvijek iste dvije ulazne slike, fotografiju stare kuće na jezeru kao izvor sadržaja, te Guernicu, sliku poznatog španjolskog slikara Pabla Picassa, kao izvor teksture (slika 4.1).

Slika 4.1 Fotografija kuće na jezeru (lijevo) koja predstavlja sadržaj i Picassova Guernica (desno) koja predstavlja stil u kojem će se sadržaj prikazati.

4.1 Inicijalizacija sintetizirane slike

Prilikom inicijalizacije slike koja se kroz algoritam generira, parametar NOISE_RATIO određuje koliko će pikseli ulazne slike sadržaja utjecati na inicijalizaciju u odnosu na slučajno generiranu noise sliku. Ako NOISE_RATIO postavimo na 0, tada se kao početna vrijednost nove slike u potpunosti uzima slika sadržaja, a postavimo li ovu konstantu na 1, slika se inicijalizira cijela sa slučajnim vrijednostima piksela. Slika 4.2 prikazuje rezultate sinteze s različitim vrijednostima parametra NOISE_RATIO.



Slika 4.2 Rezultati sinteze teksture (dolje) s različito inicijaliziranim slikama (gore), uz parametar NOISE_RATIO postavljen na 0 (A), 1 (B) i 0.6 (C).

Obrađena su dva rubna slučaja (4.2A i 4.2B) te još jedan u sredini (4.2C). Vidimo da se najbolji rezultati postižu upravo tom zlatnom sredinom. Na prvoj

slici vidimo da su oblici objekata vrlo dobro sačuvani, no pod cijenu lošije strukture željenog stila. Suprotan slučaj, na drugoj slici, dao je rezultat na kojem nijedan objekt nije raspoznatljiv, što može biti posljedica premalog broja iteracija optimizacije, ali nećemo isključiti ni mogućnost da prilikom minimizacije algoritam zaglavi u nekom lokalnom minimumu. Ipak ovo je pokazalo da algoritam najbolje radi krene li od slike s već definiranim sadržajem uz izvjesnu količinu šuma. Element nasumičnosti koji se očituje parametrom šuma, daje algoritmu određenu dozu slobode da generira i stilizira sliku tako da rezultat bude zadovoljavajuć i sa strane sadržaja i sa strane teksture.

Još jedan razlog zbog kojeg se određeni postotak šuma unosi u sliku sadržaja pri inicijalizaciji je u tome što svako novo pokretanje programa stvara drugačiji krajnji rezultat. Nasumični element u generiranju se pobrine da inicijalna vrijednost slike nikad ne bude ista kao u prethodnom procesu, pa time osigurava dozu varijabilnosti u radu algoritma. Odaberemo li fiksne početne vrijednosti piksela (primjerice, postavimo li inicijalno stanje na čistu ulaznu fotografiju), algoritam će na kraju uvijek dati jednak rezultat.

4.2 Omjer težinskih faktora sadržaja i teksture

Mada se u ovom algoritmu reprezentacije sadržaja i stila izgrađuju odvojeno i neovisno jedna o drugoj, jasna je stvar da nije moguće u potpunosti raspetljati to dvoje. Nije moguće sastaviti sliku takvu da se savršeno podudara s objema ulaznim slikama, jer u pravilu takva slika ne postoji. Međutim, kako je funkcija ukupne pogreške koja se minimizira, upravo linearna kombinacija pogreški stila i sadržaja s težinskim faktorima α i β, mijenjajući te parametre lako se može odrediti što će se od ovog dvoje pri sintezi jače isticati. Slika 4.3 prikazuje

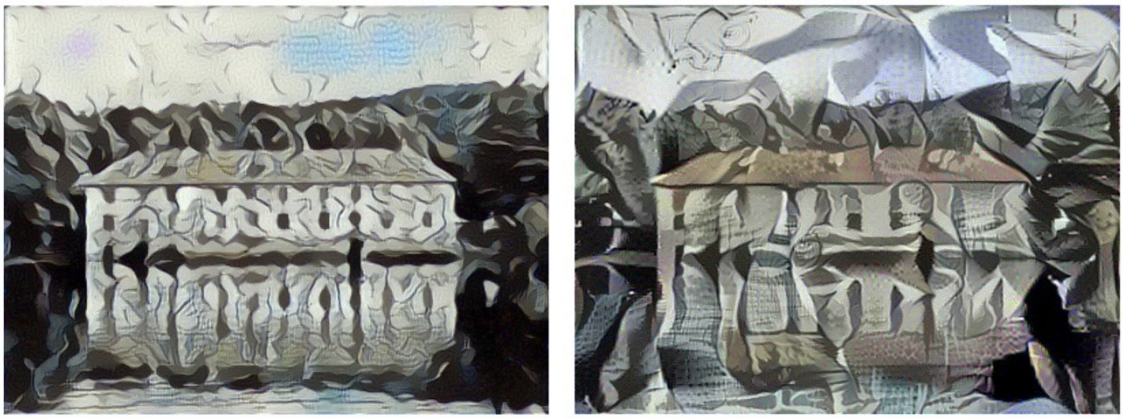
dva ekstremna slučaja, jedan u kojem je sadržaj značajno izražen u odnosu na teksturu, i drugi u kojem je situacija obrnuta. S manjom vrijednosti omjera α/β, objekti su vjerodostojno prikazani, no to ima za posljedicu vrlo slabu stilizaciju. Veći omjer ovih faktora daje teksturiranu verziju umjetničke slike dok su objekti potpuno izobličeni, pa i neraspoznatljivi. Postupnom interpolacijom između ovih rubnih konfiguracija naći će se idealan omjer među spomenutim parametrima koji će za odabrane ulazne slike dati prihvatljiv vizualan balans između stila i sadržaja.

Slika 4.3 Na lijevoj slici omjer α/β je reda veličine 10^{-5}, pri čemu je faktor stila α=0.01 i faktor sadržaja β=500. Na desnoj slici taj je omjer reda veličine 10^{5}, uz α=10000 i β=0.05.

4.3 Utjecaj broja korištenih slojeva u mreži

Još jedan vrlo bitan faktor koji utječe na rad Neuronskog algoritma umjetničkog stila je broj i dubina slojeva u konvolucijskoj mreži koji sudjeluju u izgradnji reprezentacije stila slike. Reprezentacija stila ovdje je interesantnija od reprezentacije sadržaja jer obuhvaća bitne korelacije među prostornim strukturama izvedenim kroz više slojeva mreže. Broj i dubina tih slojeva

određuju na kojem će se lokalnom razmjeru podudarati stilske reprezentacije ulazne i generirane slike. Slika 4.4 ilustrira kakav utjecaj ovo ima na rezultate.



Slika 4.4 Lijeva slika sintetizirana je koristeći slojeve 'conv1_1', 'conv2_1' i 'conv3_1' (sa slike 3.1), dok su za sintezu desne slike dodatno korišteni 'conv4_1' i 'conv5_1'. Težinski koeficijenti pridruženi navedenim slojevima redom su 0.5, 1, 1.5, 3, 4. Slojevi koji su pri sintezi lijeve slike isključeni iz računanja stilske reprezentacije, pomnoženi su koeficijentom 0.

Vidimo iz priloženog da više slojeva znači veći lokalni razmjer podudaranja, a time i bolju strukturu stilizacije. Na slici koja koristi samo 3 sloja karakteristike stila su tako slabo ekstrahirane da nimalo ne podsjećaju na Guernicu. Razlog tome je što niži slojevi izvlače informacije iz slike na vrlo niskom levelu piksela. Finalna slika sadrži niskorazinske strukturne značajke koje jedva obuhvaćaju kolorit i osnovne linije umjetničkog djela. S druge strane, na slici koja koristi 5 slojeva za reprezentaciju teksture detaljne informacije na razini piksela nisu toliko strogo ograničene pa se objekti i tekstura lijepo stapaju. Sadržaj je prikazan u željenom umjetničkom stilu na takav način da to oku djeluje prirodno i konzistentno. Rubovi, linije, boje na fotografiji su izmijenjeni tako da se slažu s umjetničkom slikom.

5 Zaključak

U ovom radu prezentirani su različiti načini i ideje kako stilističke karakteristike s jedne slike prenijeti na drugu. Najnovija i najuspješnija ideja u ovom području dosad, koja modelira slike dubokim konvolucijskim mrežama detaljnije je obrađena u teoriji, uz adekvatnu implementaciju i evaluaciju. No usprkos izvanrednim rezultatima koje daje, ipak ima nekoliko nedostataka.

Veliki problem predstavlja rezolucija završne slike. Brzina algoritma izravno je ovisna o ovoj karakteristici slike. U priloženom kodu veličina generirane slike u pikselima je 800x600, što je vrlo maleno, no uz veće rezolucije slika ovaj algoritam mogao bi se vrtjeti iznimno sporo.

Nadalje, pitanje razdvajanja stila i sadržaja ostaje djelomično nerazjašnjeno. Može li se uopće to dvoje semantički razdvojiti ne može se sa sigurnošću reći, jer je još uvijek upitna jasna definicija stila slike. Jesu li to karakteristični potezi kistom, ili specifičan kolorit, ili neki dominantni oblici i kalupi koje slika sadrži, ili je kombinacija svega toga i mnogo drugih značajki, ovo svakako nije jednostavno matematički precizno definirati.

Također valja spomenuti da su računske operacije ovog algoritma vrlo skupe i zahtjevne, te performansa programa značajno ovisi o snazi uređaja na kojem se program vrti. Duboka neuronska mreža predviđena je za obradu na grafičkim karticama visokih performansi. No, u svakom slučaju arhitektura računala je polje u kojem se danas napredak vrlo brzo odvija, stoga će ovaj problem s vremenom vjerojatno postajati sve manje bitan.

Literatura

- L. A. Gatys, A. S. Ecker, M. Bethge. Image Style Transfer Using Convolutional Neural Networks. Computer Vision and Pattern Recognition (CVPR), 2016.
- L. A. Gatys, A. S. Ecker, M. Bethge. Texture Synthesis Using Convolutional Neural Networks. Computer Vision and Pattern Recognition (CVPR), 2015.
- K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. Computer Vision and Pattern Recognition (CVPR), 2015.
- C. Zhu, R. H. Byrd, P. Lu, J. Nocedal. L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. ACM Transactions on Mathematical Software (TOMS), 1997.
- J. Portilla, E. P. Simoncelli. A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients. International Journal of Computer Vision, 2000.
- D. Stutz. Seminar Report: Understanding Convolutional Neural Networks. Fakultät für Mathematik, Informatik und Naturwissenschaften, prof.dr. B. Leibe, 2014.
- A. A. Efros, T. K. Leung. Texture Synthesis by Non-Parametric Sampling. Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference vol.2, str. 1033-1038. IEEE, 1999.
- A. A. Efros, W. T. Freeman. Image Quilting for Texture Synthesis and Transfer. The Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, str. 341-346. ACM, 2001.
- L.-Y. Wei, S. Lefebvre, V. Kwatra, G. Turk. State of the Art in Example-Based Texture Synthesis. Eurographics, 2009.
- B. Julesz. Visual Pattern Discrimination. IRE Transactions on Information Theory vol.8, 1962.
- D. P. Kingma, J. L. Ba. Adam: A Method for Stochastic Optimization. Third International Conference for Learning Representations (ICLR), 2015.
- Broyden–Fletcher–Goldfarb–Shanno algorithm, Wikipedia.
  URL *https://en.wikipedia.org/wiki/Broyden-Fletcher-Goldfarb-Shanno_algorithm*
- E. W. Weisstein. Newton's Method. *MathWorld*, A Wolfram Web Resource, 2005.
  URL *http://mathworld.wolfram.com/NewtonsMethod.html*
- Quasi-Newton method, Wikipedia.
  URL *https://en.wikipedia.org/wiki/Quasi-Newton_method*
- TensorFlow, Wikipedia.
  URL *https://en.wikipedia.org/wiki/TensorFlow*
- TensorFlow API Documentation
  URL *https://www.tensorflow.org/api_docs/*

Prijenos umjetničkog stila optimiranjem konvolucijskog modela

Sažetak

Rad obuhvaća dva temeljna pristupa za razvoj modela teksture i sintezu teksture na slici. Prvi pristup razvija neparametrizirani model teksture i oslanja se na preoblikovanje piksela ili skupine piksela ulaznog predloška teksture u svrhu generiranja novog, većeg primjerka. U poglavlju 2.1 ovog rada, opisana su dva algoritma bazirana na ovom pristupu. Drugi pristup sintezi teksture je nešto složeniji pošto se temelji na eksplicitnom parametriziranom modelu koji se sastoji od preciznih matematičkih reprezentacija određenih statističkih značajki. Veći dio rada usmjeren je prema ovom pristupu i detaljno obrađuje algoritam koji koristi konvolucijsku neuronsku mrežu kao model teksture. Za evaluaciju rezultata algoritma kao predlošci teksture, tj. stila odabrane su umjetničke slike poznatih svjetskih slikara, a zadatak algoritma je prenijeti stil s umjetničke slike na digitalnu fotografiju tako da objekti koje fotografija sadrži ostanu prepoznatljivi, ali da se stil u kojem su prikazani promjeni po uzorku na umjetničku sliku. Provedena je analiza rezultata uz različite vrijednosti parametara konvolucijske mreže. Uz izvorni kod algoritma u Pythonu i TensorFlowu priložene su i završne slike analize algoritma.

Ključne riječi: računalni vid, konvolucijska neuronska mreža, TensorFlow, Python, sinteza teksture, numerička optimizacija, prijenos stila