UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Colorarea imaginilor alb-negru prin metode de Deep Learning

propusă de

Maria Hlușneac

Sesiunea: *Iulie, 2020*

Coordonator științific

Asist. Dr. Croitoru Eugen Nicolae

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Colorarea imaginilor alb-negru prin metode de Deep Learning

Maria Hlușneac

Sesiunea: *Iulie, 2020*

Coordonator științific

Asist. Dr. Croitoru Eugen Nicolae

Cuprins

Introducere

Motivația alegerii temei

Image-to-Image Translation este o clasă de probleme de Computer Graphics care are ca scop învățarea unei mapări între o image input și o imagine output. Problemele de sub umbrela acestei clase sunt de o mare diversitate, de la simpla ajustare a unei imagini până la style/season transfer (schimbarea stilului pictural sau a anotimpului unei scene dintr-o imagine) sau transformarea unui cal într-o zebră. Algoritmii de deep learning folosiți pentru probleme de translație dau rezultate impresionante, indiferent despre ce problemă este vorba. Astfel, varietatea și creativitatea task-urilor de translație existente și puterea algoritmilor din ultimii ani asupra acestora m-au determinat să îmi doresc să abordez o problemă de acest gen. Pentru a contribui în sens real și ca rezolvarea problemei să se bucure de un aspect pragmatic, am ales problema colorării imaginilor alb-negru.

Fotografiile vechi alb-negru nu beneficiază de profunzimea dată de culoarea fotografiilor noi, care redau realitatea așa cum este. Colorarea imaginilor se poate realiza prin mijloace manuale (de exemplu în Photoshop); acestea pot da rezultate foarte naturale, însă atingerea lor necesită foarte mult timp și efort. Colorarea prin metode automate, însă, poate rezolva această problemă și poate obține fotografii vii cu războaie, îmbrăcăminte specifică timpurilor respective sau poate împrospăta amintirea unei rude dintr-o poză veche de familie.

Obiectivele generale ale lucrării

Lucrarea de față își propune să realizeze colorarea de portrete și peisaje prin mai multe tipuri și configurații de rețele neuronale și a compara, rezultatele și configurațiile în urma cărora acestea s-au produs. Scopul de atins constă, pe de o parte, în a ajunge la o colorare cât mai verosimilă a imaginilor și, pe de altă parte, în a trage concluzii importante în legătură cu detaliile configurațiilor folosite, pentru ulterioare direcții de cercetare pe această problemă.

Descriere sumară a soluției

Colorarea se realizează prin rețele neuronale de tip autoencoder simplu, iar apoi de tip Conditional GAN (Conditional Generative Adversarial Network), ce sunt alcătuite dintr-un generator de tip rețea convoluțională, sub arhitecturi de autoencoder sau U-Net, și un discriminator convoluționat (dă verdicte de real sau fals pentru bucăți de imagine).

Rezultatele sunt analizate prin expunerea imaginii colorate (și a diferențelor față de imaginea originală) și prin grafice ce redau aspecte ale antrenamentului rețelelor.

Datele pentru antrenarea modelelor constau într-un set de portrete și un set de peisaje preluate de pe internet, cu mențiunea că setul de peisaje a fost mărit prin tehnici manuale.

Structura lucrării

În lucrarea de față se va descrie problema ce încearcă să fie rezolvată (Capitolul I), se vor prezenta câteva abordări anterioare ale acesteia (Capitolul II), iar apoi se va descrie detaliat soluția propusă (Capitolul III). Descrierea soluției se împarte în câteva subcapitole:

III.1. Tehnologii folosite

III.2. Pregătirea datelor (colectare, preprocesare, stocare)

III.3. Antrenarea unor rețele neuronale

III.4. Rezultate

La sfârșitul tezei se vor trage câteva concluzii și exprima opinii personale, iar apoi se vor propune posibile direcții viitoare ale lucrării.

De asemenea, în Anexa 1 sunt redate detalii despre conceptele folosite la etapa de antrenare și motivația alegerii lor.

Contribuții personale

Pentru rezolvarea problemei în jurul căreia se structurează lucrarea, colectarea datelor a constituit o etapă importantă. Realizarea unuia dintre seturile de date folosite reprezintă, într-o anumită măsură, contribuție proprie. O parte din acest set de date este preluată de pe internet, iar o altă parte conține imagini obținute prin realizarea capturilor de ecran de-a lungul unor videoclipuri pe Youtube și prin colectarea unor imagini prin intermediul unui API oferit de platforma Flickr.

O altă contribuție adusă constă în faptul că lucrarea de față se concentrează pe compararea mai multor moduri de a colora imagini, astfel ajungând la concluzii importante cu privire la influența unor anumite particularități din arhitectura unei rețele neuronale asupra modului în care aceasta colorează o anumită imagine. Aceste concluzii pot fi relevante pentru o eventuală cercetare viitoare pe acest segment de Computer Graphics.

În plus, una dintre arhitecturile implementate (un model de tip GAN) dispune de o rețea (discriminatorul) care, după cunoștințele mele, nu a mai fost utilizată până în prezent, îndrăznind să o numesc idee proprie, sau cel puțin, care nu a mai fost întâlnită de mine până acum.

I. Descrierea problemei

Apariția fotografiei și-a făcut loc în anul 1838. Prima fotografie făcută a avut nevoie de 10 minute de expunere și era compusă doar din alb, negru și tonuri de gri. Nu a durat mult până la apariția primei fotografii color, acest eveniment având loc în anul 1861. În România, deși prima fotografie alb-negru a fost realizată în anul 1848, prima fotografie color a apărut mult mai târziu decât aceasta și decât prima fotografie color din lume: în anul 1970. Așadar, bunicii și poate chiar și părinții generației de astăzi s-au bucurat, până la un moment dat, doar de fotografii lipsite de culoare, care le aduc astăzi aminte de copilăria și tinerețea lor.

Colorarea unei fotografii alb-negru este un proces extrem de laborios, care necesită mult timp, efort, răbdare și experiență în domeniu, dacă este realizat manual. Dacă se dorește colorarea unui film, efortul este cu mult mai mare, având în vedere că o secundă înglobează, de obicei, cel puțin 20 de cadre.

Totuși, progresele tehnologice din ultimul timp au condus la posibilitatea de a automatiza acest proces și a obține, totodată, și rezultate foarte bune. Datorită avansărilor din domeniul Deep Learning din ultimii ani, colorarea imaginilor alb-negru se realizează cu mult mai puțin efort uman. Există mai multe abordări ale problemelor din clasa Image-to-Image Translation, în general, și a problemei de colorare, în particular, însă nu toate abordările sunt puse, împreună cu rezultatele lor, la un loc, pentru a se putea dezvolta o idee clară asupra particularităților ce fac o abordare să fie mai bună decât o alta, în funcție de ce fel de categorie de imagini este folosită, de ce dimensiuni au imaginile etc.

O observație importantă este aceea că task-ul de colorare nu presupune recrearea culorilor din imaginea originală (dacă aceasta există), ci obținerea unei imagini cu culori plauzibile, diferite sau nu de cele din imaginea originală.

II. Abordări anterioare

În lucrarea [1] s-a obținut posibilitatea de a colora o imagine în timp real de către un utilizator oarecare. Utilizatorul poate alege zone din imagine care reprezintă persoane, animale sau obiecte și să le coloreze în orice mod dorește. Pe lângă acest lucru, creatorii oferă utilizatorului culori din gama potrivită zonei selectate. Abordarea problemei este printr-o rețea convoluțională de tip U-Net complexă, ce primește imaginea alb-negru și puncte în spațiu ce reprezintă opțiunile utilizatorului. Antrenamentul rețelei neuronale s-a realizat pe un set de date cu 1 milion de imagini.

Lucrarea [2] prezintă un framework ce rezolvă o gamă largă de probleme de tip Image-to-Image Translation. Framework-ul este creat astfel încât nimic din componența lui să nu trebuiască să se schimbe de la o problemă la alta. Are loc antrenarea unui model neuronal de tip GAN condiționat. Acesta are în componența sa un generator de tip U-Net, care are ca input un vector de zgomot și imaginea alb negru (în acest fapt constând condiționarea). Discriminatorul este de tip PatchGAN, având ca output mai multe verdicte, fiecare corespunzător unei bucăți de imagine. Pentru task-ul de colorare, antrenamentul GAN-ului condiționat a fost rulat pe un set de 1,2 milioane de imagini.

În lucrarea [3] se colorează atât imagini, cât și videoclipuri. Colorarea se realizează prin intermediul *Self-Attention GAN*, rețea care oferă imagini cu detalii profunde. O altă rețea prin care se rezolvă problema colorării este NoGAN, rețea dezvoltată de către autorul proiectului, în care la început, generatorul și discriminatorul se antrenează separat unul de celălalt, lucru care salvează timp. Funcția de loss folosită pentru GAN-uri este Perceptual Loss, iar partea de generator este implementată cu un U-Net.

Lucrarea [4] prezintă o abordare a problemei colorării cu o rețea neuronală VGG. Se folosește o funcție de loss cu ajutorul căreia se modelează o distribuție de probabilități din ce în ce mai bună pentru culorile posibile a unui pixel și de asemenea, loss-ul este remodelat la antrenare pentru a da șansă culorilor extreme să poată apărea în imagine. La sfârșit se folosește un test Turing de colorare prin care se cere participanților să identifice care este imaginea generată și care este cea reală. Participanții sunt păcăliți în 32% din cazuri, atingându-se astfel rezultate impresionante.

În lucrarea [5] se prezintă o abordare asemănătoare cu cea din lucrarea [3], cu o rețea neuronală de tip GAN. Aspectul diferit al implementării față de altele întâlnite este că modelul a fost antrenate pe imagini cu specific singaporean. Rezultatele sunt foarte bune, atât pe peisaje, cât

și pe imagini cu puțini sau mulți oameni. Proiectul este livrat ca o aplicație web, unde colorarea imaginii încărcate durează aproximativ 3 secunde.

Lucrarea [6] propune o abordare pe nivele de înțelegere a unei imagini. Se folosește o rețea neuronală convoluțională care este împărțită în patru componente: detectare de caracteristici de bază, intermediare și înalte (detalii) și, în final, colorarea. Colorarea se realizează în funcție de aceste nivele ale caracteristicilor imaginii și pentru o rafinare mai puternică se folosesc clase atribuite imaginilor.

III. Descrierea soluției

III.1. Tehnologii folosite

III.1.1. Mediul de lucru

Am ales să creez aplicația în limbajul Python, acesta fiind unul din limbajele preferate pentru domeniul de Deep Learning datorită framework-urilor și bibliotecilor scrise pentru el, care ușurează modul de lucru cu algoritmii și operațiile necesare în acest domeniu.

Mediul de lucru al aplicației a fost, inițial, PyCharm, însă s-a schimbat apoi în Google Colaboratory, iar apoi Google Colaboratory Pro, din considerente de creștere a complexității algoritmilor abordați și, deci, și a resurselor necesare.

Google Colaboratory versiunea basic pune la dispoziție GPU-uri performante, însă timpul în care se executau antrenamentele rețelelor neuronale era totuși nemulțumitor, chiar și comparativ cu GPU-ul laptop-ului propriu. În plus, *runtime*-ul în Google Colaboratory (timpul maxim alocat conectării la un GPU) este de 12 ore, durată mai mică decât cea a unui antrenament. Prin urmare, mediul de lucru s-a schimbat în Google Colaboratory Pro. Această versiune pune la dispoziție plăci video mai performante și un runtime mai mare, de 24 de ore.

Tipul plăcilor video folosite pentru lucrarea de față este NVidia Tesla (unul din modelele T4 sau P100), cu memorie RAM de 25GB. Antrenamentele s-au desfășurat, astfel, într-un timp mai scurt, cu toate că uneori timpii tot au depășit runtime-ul, realizându-se așadar doar parțial.

Datele de antrenare, script-urile și toate celelalte fișiere utilizate pentru proiect au fost stocate în mediul Google Drive.

III.1.2. Seturile de date

Setul de date ce conține imagini de tip portret a fost preluat de pe internet (platforma Kaggle [12]). Setul de imagini de tip peisaj a fost creat din două direcții: prima, preluând un set de date preexistent de pe site-ul oficial MIT [13] și selectând din acesta doar imaginile de interes (imaginile de tip peisaj reprezentau doar o parte din set). A doua direcție a constat în lucru manual, anume, pe de o parte, în redarea de videoclipuri cu peisaje de pe Youtube și realizarea de capturi de ecran în mod automat dintr-un script și pe de altă parte, în folosirea un API pus la dispoziție de site-ul Flickr (platformă cu milioane de imagini din diverse categorii).

Rețelele neuronale au nevoie, pentru antrenament, de imagini date ca array-uri Numpy, așa că imaginile au fost reținute în această formă. Întrucât nici memoria RAM mare a plăcilor video puse la dispoziție de mediul de lucru nu a fost de ajuns pentru a încărca toate imaginile la un moment dat, array-urile numpy au fost stocate în mai multe arhive (în format .npz), urmând ca pentru antrenare să fie încărcată câte o arhivă pe rând.

Pentru un mod la îndemână de a manipula imaginile (citire, salvare, postprocesare), a fost folosită biblioteca PIL.

III.1.3. Antrenamentul de rețele neuronale

Abordările cu modele neuronale utilizate se împart în 2 ramuri: autoencoder simplu și Conditional Generative Adversarial Network, fiecare dintre ele constând dintr-unul sau mai multe modele convoluționale. Atât descrierea cât și motivația alegerii acestor arhitecturi sunt prezentate în Anexa 1.

III.2. Pregătirea datelor

III.2.1. Colectarea datelor

Pregătirea seturilor de date pentru antrenarea unor modele neuronale poate fi, de multe ori, un proces laborios și dificil, uneori punând probleme poate chiar mai mari decât antrenarea modelelor în sine. Pentru problema de față, a trebuit ca seturile de date să îndeplinească câteva cerințe: imaginile să nu fie prea mici (mai mici de 150x150 de exemplu), seturile de date să fie suficient de mari (să conțină câteva zeci de mii de imagini), imaginile cu peisaje să nu conțină (prea mulți) oameni în componența lor, să cuprindă un cadru larg (de exemplu nu doar o clădire), de preferat să fie imagini din natură; imaginile cu portrete să conțină în principal doar chipul omului, nu și restul corpului, adică fotografiile să fie făcute de aproape și preferabil fundalul să nu fie prea complicat, să nu conțină alte obiecte.

În cazul portretelor, un set de date a îndeplinit toate cerințele menționate, set care a și fost folosit pentru antrenare: se numește CelebA Dataset (CelebFaces Attributes Dataset) și a fost preluat de pe Kaggle [12]. Acesta conține aproximativ 200 000 de imagini cu chipurile a aproximativ 10 000 de celebrități, iar fiecare imagine este adnotată cu 40 de atribute (acestea nefiind

semnificative pentru problema colorării, ci pentru alte posibile probleme ce ar avea nevoie de acest set de date). Imaginile au toate aceeași dimensiune, de 218x178.

În cazul peisajelor însă, nu a fost găsit, în primă instanță, un set de date suficient de mare și care să cuprindă cadre largi sau să conțină peisaje din natură, fără oameni. Un mod alternativ de obținere a imaginilor l-au reprezentat platforma Flickr și API-ul lor pus la dispoziție ca bibliotecă în limbajul Python. Flickr dispune de peste 2 milioane de imagini cu tag-ul "landscape", însă descărcarea lor prin API a fost problematică. În încercări repetate de a descărca 200 000 de imagini (doar 10 procente din totalul existent), de fiecare dată rezultatul era același: din cele 200 000, doar aproximativ 200 erau unice. Pe site-ul flickr.com problema nu pare să existe, totuși, spre deosebire de API, însă descărcarea manuală a acestora nu era fezabilă. Un alt mod de a colecta imagini a fost prin intermediul Youtube. De-a lungul câtorva videoclipuri lungi (2-6 ore) cu peisaje au fost făcute în mod automat capturi de ecran. Din păcate nici de aici nu s-au obținut suficiente imagini, deoarece în unele videoclipuri o imagine era afișată și pentru câteva minute, sau în același videoclip o imagine era afișată de mai multe ori. Numărul de imagini colectate de pe Youtube este de aproximativ 1000. Totuși, un set de date suficient de mare (2,5 milioane de imagini) și care să satisfacă toate cerințele menționate a fost găsit ulterior, pe site-ul MIT [13]. Setul conține imagini pe diverse teme, un număr de aproximativ 82 000 de peisaje fiind extrase din acesta.

III.2.2. Preprocesarea datelor

Imaginile cu peisaje prelevate de pe Youtube și Flickr aveau dimensiuni mai mari decât a celor din setul de date pus la dispoziție de MIT, așadar au fost redimensionate la 256x256 pixeli (dimensiunile imaginilor din setul de date cel din urmă menționat) pentru a nu exista diferențe între ele din punct de vedere al mărimii. Existența unor diferențe în dimensiuni nu ar fi făcut posibil antrenamentul de rețele neuronale. De asemenea, aceste imagini obținute manul au fost augmentate: au fost oglindite și tăiate în două părți (ce se suprapun) pentru a obține în plus de 4 ori mai multe imagini. Împreună cu cele din datele de la MIT au constituit setul final de imagini tip peisaj ce numără puțin peste 80 000 de elemente.

Seturile de date obținute erau constituite doar din imagini color, adică label-ul unei instanțe; instanțele (imaginile alb-negru), însă, lipseau. Pentru a le crea a fost nevoie de aplicarea unei simple funcții care le transformă în format grayscale. Seturile de date sunt acum complete, fiecare instanță fiind pusă în corespondență cu label-ul său.

Rețelele neuronale antrenate reduc dimensiunea pozei, iar apoi o expandează, factorii de downscale și upscale având valoarea 2. Reducerea și expandarea cu acest factor se realizează de 3 ori în cadrul fiecărei arhitecturi (fiecare rețea de tip autoencoder are 3 straturi de upscale/downscale). Dimensiunile actuale ale imaginilor de tip portret (218x178) pot fi împărțite o singură dată exact la 2 (și ar deveni 109x89). Dacă împărțirea la 2 ar continua, dimensiunile de după ar fi 54x44. Când imaginea/tensorul cu această dimensiune ar ajunge în faza de upscale, dimensiunea nouă ar deveni 108x88, care este diferită de 109x89, corespondenta ei. Când autoencoder-ul este de tip U-Net și se realizează concatenarea tensorilor rezultați din straturi simetrice, tensorii trebuie să aibă exact aceleași dimensiuni. Prin urmare, un tensor de dimensiunea 108x88 nu se poate concatena cu un tensor de dimensiunea 109x89. Așadar, dimensiunile originale ale imaginii trebuie ajustate pentru ca împărțirile la 2 să fie exacte de fiecare dată. Pentru acest lucru, am tăiat 2 pixeli de pe lățimea imaginii și 2 pixeli de pe lungime, obținând imagini de dimensiunea 216x176, aceste numere împărțindu-se exact la 2 de 3 ori, cât era necesar.

Ulterior, seturile de date au fost împărțite în date de antrenare, validare și test, în număr variabil pentru mai multe experimente (portrete: ~32 000, peisaje: ~32 000 sau ~82 000), însă, în general, datele de antrenare sunt în proporție de aproximativ 80%, iar cele de validare și test în proporție de aproximativ 10%.

III.2.3. Stocarea datelor

Imaginile au dimensiuni destul de mari, iar din acest motiv un set de date nu poate fi încărcat complet în memoria RAM a plăcii video la un moment dat, cum ar fi necesar pentru antrenarea unei rețele neuronale. Soluția la această problemă a fost încărcarea pe rând în timpul antrenamentului a câte unei bucăți din setul de date, ulterior aceasta fiind suprascrisă de fiecare dată când o nouă bucată este încărcată. Aceste bucăți se "traduc" în termeni de memorie prin arhive (extensia .npz), acesta fiind modul în care au fost stocate datele în Google Drive. Imaginile din arhive sunt reținute sub formă de array-uri Numpy, cele alb-negru pe un canal, iar cele color pe 3 canale, în format RGB.

Acest mod de a stoca imaginile a condus la necesitatea unor modificări în secvența de cod pentru încărcarea datelor în memorie. A fost nevoie de suprascrierea unei clase speciale din Keras (Sequence), datele de antrenare, validare și testare fiind date către rețele sub formă de instanțe ale clasei respective.

III.3. Antrenarea de rețele neuronale

Antrenamentul este constituit din 2 părți: o rețea de clasificare și o rețea de predicție. Rețeaua de clasificare este folosită pentru a determina dacă imaginea ce se dorește a fi colorată reprezintă un peisaj sau un portret. În funcție de verdictul obținut, va fi încărcat fie un model antrenat pe peisaje, fie unul antrenat pe portrete, pentru a face predicție pe imagine (a o colora). Antrenarea nu s-a făcut pe un set de date comun, atât cu peisaje, cât și cu portrete, pentru a obține rezultate mai bune.

Rețeaua de clasificare

Pentru rețeaua de clasificare s-a folosit o rețea convoluțională clasică. Setul de date a constat în 3635 de imagini de tip portret și 3635 de imagini de tip peisaj (redimensionate la dimensiunile portretelor). Pentru testare s-au folosit 512 portrete și 512 peisaje. Acuratețea obținută este de aproximativ 99%. După ce imaginea dată trece de etapa de clasificare, este dată mai departe la un model specific clasei din care face parte pentru a se colora

Rețeaua de predicție/colorare

Pentru rețeaua de predicție s-a mers pe 2 direcții. O direcție o reprezintă o rețea neuronală de tip autoencoder, iar cealaltă o rețea neuronală de tip GAN, fiecare având diferite implementări și variații de hiperparametri. În total au fost încercate 4 variante de autoencoder și 15 variante de rețea GAN. Detalierea din punct de vedere conceptual ale acestor tipuri de rețele și motivația dezvoltării și alegerii acestora se găsesc în Anexa 1. Detalierea din punct de vedere tehnic (implementarea fiecărui tip de arhitectură) se găsește în rândurile ce urmează.

Seturile de date folosite la antrenamente:

- portrete: antrenare: 32 768 de imagini, validare: 6144 de imagini, test: 3072 de imagini;
- peisaje:
  - set de date restrâns: antrenare: 32 768 de imagini, test: 4096 de imagini;
  - set de date extins: antrenare: 81 920 de imagini, test: 8192 de imagini.

Abordarea 1

Rețeaua de tip autoencoder are ca input o imagine alb-negru și ca output imaginea colorată corespunzătoare imaginii input.

Autoencoder-ul este formată din blocuri convoluționale și blocuri de deconvoluție, adică din mai multe secvențe ce conțin anumite layere.

Blocurile convoluționale conțin:

- un strat convoluțional (Conv2D) (acesta poate să micșoreze sau nu tensorul)
- un strat de normalizare (BatchNormalization) (opțional)
- un strat de activare (ReLU/LeakyReLU)
- un strat de MaxPooling (dacă nu s-a realizat micșorarea dorită cu stratul convoluțional)

Blocurile de deconvoluție conțin:

- un strat de convoluție transpusă (Conv2DTranspose) sau un strat de *upsampling* (UpSampling2D), urmat de un strat convoluțional (Conv2D) (tipul de strat este parametrizat)
- un strat de Dropout (opțional)
- un strat de concatenare (Concatenate) a stratului curent cu stratul său simetric (opțional) (parametrizat)
- un strat de normalizare (BatchNormalization) (opțional)
- un strat de activare (ReLU/LeakyReLU)

Fiecare strat convoluțional are, printre parametrii săi, un parametru ce constituie numărul de filtre folosite și un parametru ce reprezintă dimensiunea filtrului. Dacă notăm cu `C(x,z)` și `D(x,z)` un bloc convoluțional, respectiv deconvoluțional, ce au numărul de filtre egal cu x de dimensiunea z*z, atunci arhitectura de tip autoencoder descrisă mai sus are următoarea structură:

`C(64,3)-C(128,3)-C(256,3)-D(256,3)-D(128,3)-D(64,3)-C(3,3).`

Ultimul strat are rolul de a da ca output un tensor corespunzător unei imagini pe 3 canale (de aici cele 3 filtre folosite).

4 tipuri de autoencoder au fost testate, numărul acestora venind din cei 2 parametri menționați mai sus la blocurile deconvoluționale: metoda de upsampling (fie prin stratul Conv2DTranspose, fie prin stratul UpSampling2D) și dacă autoencoder-ul prezintă *skip-connections* sau nu (prin introducerea sau nu a stratului Concatenate).

Abordarea 2

Această abordare constă într-o arhitectură de tip GAN (Generative Adversarial Networks), mai exact o rețea GAN de tip condițional (cGAN). În fapt, partea generativă este eliminată aici, din motive de acuratețe (explicația se regăsește în Anexa 1); va fi, totuși, referită, din acest punct, ca generator.

Arhitecturile de tip GAN sunt formate din (cel puțin) 2 rețele: generatorul și discriminatorul, acestea jucând un joc adversarial de sumă 0.

GAN-ul primește ca input o imagine alb-negru. Este dată la generator, iar acesta o colorează. Imaginea colorată împreună cu imaginea alb-negru sunt date ca input la discriminator, iar output-ul acestuia, care este de fapt și output-ul rețelei GAN, reprezintă o matrice cu numere între 0 și 1, fiecare corespunzând unei bucăți din imagine și având semnficația: între 0 și 0,5 – imaginea colorată nu provine din imaginea alb-negru; între 0.5 și 1 – imaginea colorată provine din imaginea alb-negru.

Generatorul a fost implementat ca un autoencoder, fiind foarte asemănător cu cel prezentat la abordarea anterioară. Utilizând aceeași convenție ca la autoencoder-ul de la Abordarea 1, structura generatorului poate fi scrisă astfel (blocuri operațiilor de convoluție și deconvoluție rămân aceleași):

`C(64,3)-C(128,3)-C(128,1)-C(256,3)-C(256,1)-C(512,3)-C(512,4)-`
`-D(512,1)-D(256,3)-D(256,1)-D(128,3)-D(128,1)-D(64,3)-D(3,3).`

Stratul scris subliniat reprezintă *bottleneck*-ul autoencoder-ului, iar ultimul strat este folosit din același motiv ca la Abordarea 1 (acela de a da ca output un tensor corespunzător unei imagini pe 3 canale).

Precizare: pentru ca rețeaua să poată dispune de mai multe straturi (creștere a complexității și a valorii rezultatelor), dar numărul său de parametri să nu crească cu mult, s-a folosit și mărimea 1 a *kernel*-ului.

Generatorul are, de asemenea, ca parametri, modul de upsampling și skip-connections (cu valori booleane), deci rețeaua GAN va dispune de 4 tipuri de generator.

În Figura 1 din paginile ce urmează este ilustrată arhitectura generatorului care prezintă skip-connections (autoencoder-ul este de tip U-Net) și metoda de upsampling este realizată prin intermediul stratului Conv2DTranspose.

Discriminatorul a fost implementat în 2 moduri.

- Primul mod este asemănător cu cel descris în [2]. Acest tip de discriminator, PatchGAN, este format din blocuri convoluționale, un bloc conținând straturile:

- Conv2D
- BatchNormalization (opțional)
- LeakyReLU,

cu parametri recomandați în [2]. Utilizând convenția folosită și până acum, arhitectura discriminatorului este:

`C(64,3)-C(128,1)-C(256,3)-C(512,1)-C(512,3)-C(1,1)-A.`

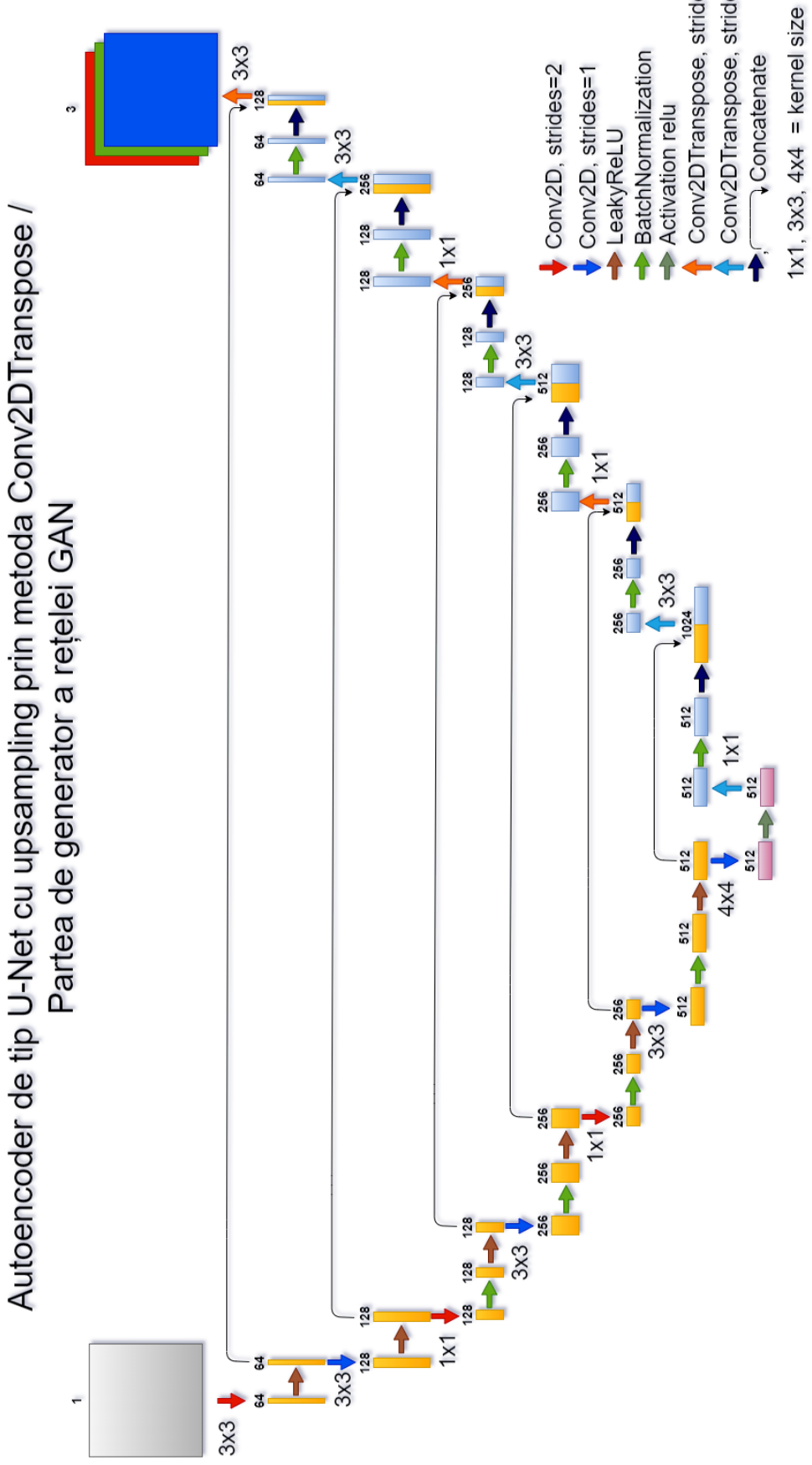


Figura 1 – Arhitectură generator cu skip-connections și Conv2DTranspose

Penultimul strat are mărimea kernel-ului egală cu 1 pentru că rezultatul discriminatorului este o matrice (de valori corespunzătoare verdictului acestuia) și nu trebuie să dispună de dimensiuni în plus. Ultimul strat este un strat de activare sigmoidală, pentru a aduce valorile în intervalul [0, 1].

Arhitectura acestui tip de discriminator este ilustrată în Figura 2 de mai jos.

- Al doilea mod în care a fost implementat discriminatorul provine dintr-o idee proprie, aceea de a convoluționa câte o bucată (*patch*) de imagine în mod separat. Intuiția din spate este că, în acest fel, bucățile de imagine nu se mai suprapun (cum se întâmplă la prima implementare) și astfel se acordă o atenție mai mare fiecărui patch, acestea fiind, acum, și mai mici. Rezultatul ar fi o imagine de o claritate sporită.

Numărul de patch-uri este dat ca parametru (rețeaua a fost antrenată cu valorile 4 și 8).

Peste fiecare patch au fost aplicate straturile:

- Conv2D,
- LeakyReLU, structura fiind:

`C(64,3)-C(256,3)-C(256,3)-C(512,2)-C(1,1).`

În final s-a realizat concatenarea patch-urilor rezultate *image-wise* și s-a aplicat un strat de activare sigmoidală.

În total 3 tipuri de discriminator au fost încorporate în rețeaua GAN: discriminatorul PatchGAN și discriminatorul cu patch-uri nesuprapuse, cu 4x4 și cu 8x8 patch-uri.

3 tipuri de generator și 4 tipuri de discriminator au rezultat în 12 variante de GAN.

Unul dintre parametrii rețelei GAN este *loss_weights*, care reprezintă o listă de 2 valori: una corespunzătoare generatorului și cealaltă corespunzătoare discriminatorului, având semnificația de însemnătate acordată unui loss. De exemplu, dacă loss_weights = [20, 80], atunci se va acorda o însemnătate semnificativ mai mare valorii de *loss* a discriminatorului decât loss-ului generatorului, adică se va încerca ca discriminatorul să dea verdicte mai corecte în detrimentul colorării mai bune a generatorului. Totuși, generatorul este inflențat indirect de activitatea discriminatorului, deci cu cât discriminatorul se descurcă mai bine, cu atât și generatorul va avea performanțe crescute.

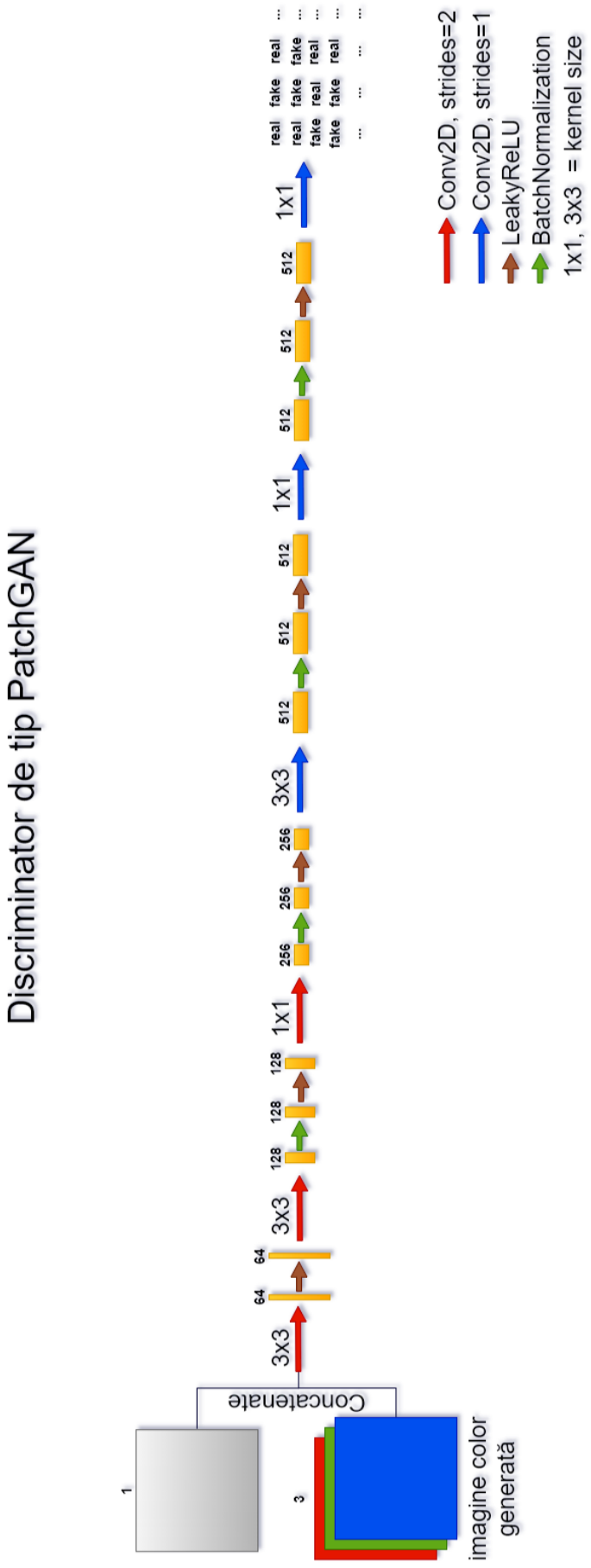


Figura 2 – Arhitectură discriminator de tip PatchGAN

După ce au fost antrenate cele 12 modele de GAN pe setul de date cu peisaje, având loss_weights=[20, 80], a fost selectat modelul cu rezultatele cele mai bune, care a fost antrenat din nou, de 3 ori, cu parametrul loss_weights diferit de la un antrenament la altul. Aceste 3 modele au fost rulate doar pe portrete, întrucât rezultatele nu au fost atât de bune încât să fie antrenate și pe peisaje.

În total au fost antrenate 12+3=15 configurații de rețea GAN și 4 configurații de autoencoder, fiind 19 modele în total.
Pentru un număr de epoci de 100, fiecare dintre antrenamentele rețelelor de tip autoencoder, a durat, în medie (în funcție de configurația fiecăruia, complexitatea rețelei era ușor variabilă), în jur de 15 ore pe portrete. 70 de epoci au durat aproximativ 21 de ore pe un set de date de aceeași mărime de peisaje (timpii diferă deoarece dimensiunile imaginilor de tip peisaj sunt mai mari decât cele ale portretelor). Pe setul de date extins de peisaje, antrenarea timp de 30 de epoci a autoencoder-ului a durat aproximativ 16 de ore.
Un GAN, însă, fiind o rețea mai laborioasă, a avut ca timp de antrenare între 22 și 37 de ore pe setul de portrete (în funcție de configurație). Din motive de instabilitate a conexiunii la placa video de pe Google Colaboratory, nu toate antrenamentele au fost rulate timp de 100 de epoci (pe setul de date cu portrete, cel mai mic număr de epoci a fost 44). Antrenamentul de 100 de epoci ar fi durat aproximativ 100 de ore pe setul de peisaje restrâns și aproximativ 250 de ore pe setul de peisaje extins. Din aceleași motive de conexiune, pe setul dintâi menționat s-au rulat 30 epoci, iar pe cel de-al doilea, 15 epoci. Rețelele nu au fost toate antrenate de 2 ori (o dată pentru peisaje și o dată pentru portrete). Ce s-a făcut, în schimb, a fost ca întâi să fie antrenate toate pentru imagini de tip portret, ca mai apoi să se aleagă 2 modele care au dat rezultatele cele mai bune pentru a fi antrenate ulterior și pe imagini de tip peisaj.

Postporcesarea imaginilor

În funcție de rezultatele obținute în urma antrenării modelelor neuronale, unele imagini (cele de tip peisaj) au fost procesate ulterior, în mod automat, pentru a dispune de culori mai aprinse.

III.4. Rezultate

Rezultatele din urma antrenamentelor au fost uneori așteptate, alteori surprinzătoare. Mai jos urmează o prezentare a câtorva dintre cele mai bune, cât și a unora corespunzătoare unor configurații de rețele care s-au dovedit a nu fi tocmai câștigătoare.

În Figura 3 de mai jos sunt prezentate unele dintre cele mai bune rezultate obținute. Peisajele prezentate au trecut, în plus față de cele de tip portret, printr-o etapă de postprocesare.

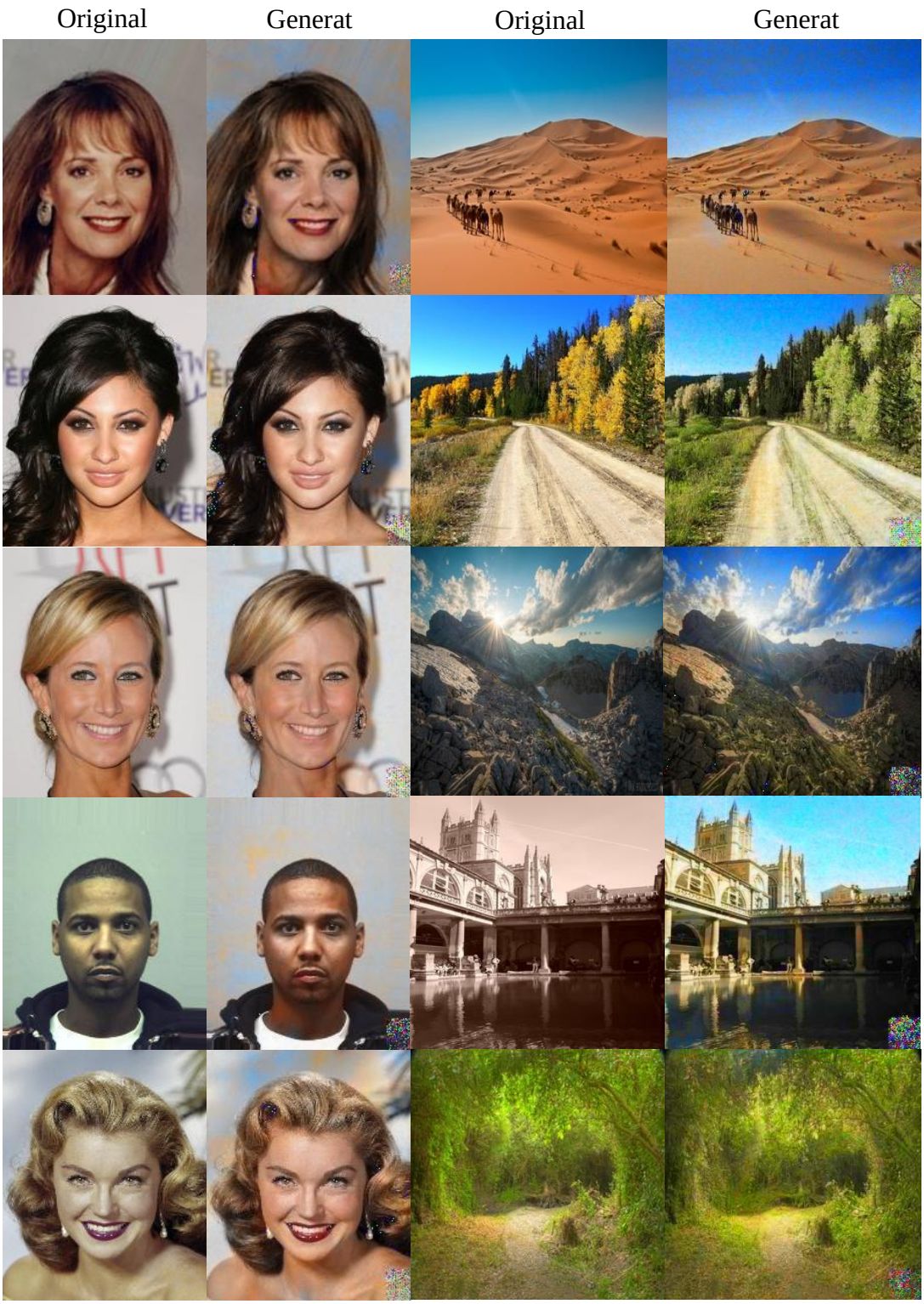


Figura 3

În continuare se va face analiza mai multor rezultate și se vor compara configurațiile modelelor care le-au generat. Rezultatele vor fi evaluate atât manual, anume din punct de vedere vizual, dar și conform unei metrici (diferența medie în modul în medie între imaginea generată și cea originală). Concluziile se vor trage ponderând mai mult factorul uman decât rezultatele metricii. Motivul este că unele imagini generate au, conform opiniei personale, o gamă de culori mai naturală decât imaginile originale, astfel diferența între ele fiind mare. În mod intuitiv, o diferență mare între o imagine generată și originalul său ar corespunde unui rezultat mai slab și nu unui rezultat mai bun, cum se întâmplă de fapt aici, în unele cazuri. Astfel, în această situație, un rezultat care conduce la o diferență mare nu ar trebui penalizat, din punctul propriu de vedere, ci dimpotrivă. Totodată, o diferență mică între generat și original nu însemnă neapărat că modelul neuronal s-a descurcat bine (de exemplu, dacă imaginea originală are culori fade iar imaginea generată la fel). În plus, task-ul de colorare constă nu în încercarea de a reproduce imaginea originală, ci în încercarea de a produce o imagine cu culori plauzibile, care să poată fi echivalentul imaginii de la care s-a plecat. Aceste considerente fac ca diferența dintre imaginile generată și originală să nu fie un aspect ce poate exprima cu acuratețe calitatea rezultatelor și, prin urmare, se va lua în calcul preponderent factorul uman, asumându-se subiectivitatea acestuia și timpul crescut de evaluare.

Autoencoder - Portrete

Pe setul de portrete s-au antrenat timp de 100 de epoci 4 modele de autoencoder: cu skip connections sau fără, cu metoda de upsampling Conv2DTranspose sau UpSampling2D. Rezultatele sunt prezentate mai jos, în Figura 4.

La o privire realizată îndeaproape, se poate observa că imaginile provenite în urma antrenamentelor de modele fară skip connections au o calitate mai slabă decât cele provenite din modelele cu skip connections, prin faptul că au o lipsă de claritate (cel mai ușor se observă în zona imaginii unde se află dinții). De asemenea, dacă se compară cele două imagini "etichetate" cu UpSampling2D cu celelalte două etichetate cu Conv2DTranspose, se observă o lipsă de strălucire a primelor două față de ultimele două (exemplu de zonă unde acest lucru este observabil: obrazul din stânga).

Această analiză conduce la faptul că autoencoder-ul cu rezultatele cele mai bune este cel cu skip connections și care are ca metodă de upsampling stratul Conv2DTranspose.

De asemenea, se poate observa că imaginea originală are o tentă roșiatică, iar rezultatele date de oricare dintre modele sunt formate din culori mai naturale, ceea ce face ca autoencoder-ul să fie un model puternic.

Figura 4

Analiza acestor aspecte este de dorit a fi un proces obiectiv însă, pentru că "frumusețea este în ochii privitorului", modul de a percepe o fotografie nu este universal. Deși se încearcă o abordare cât mai detașată cu putință, este posibil ca în unele cazuri opiniile susținute să conțină o anumită doză de subiectivitate.

Rezultate mai puțin satisfăcătoare

Din rezultatele prezentate în Figura 5 de mai jos, se poate observa că uneori, imaginea rămâne cu câteva zone necolorate. De asemenea, se observă că accesoriile și hainele colorate strident sau părul vopsit într-o culoare mai puțin comună nu sunt colorate corespunzător. Acestea sunt acoperite de o culoare comună: o nuanță de maro, întrucât este, probabil, culoarea care generează de obicei un loss al rețelei cât mai mic, fiind des întâlnită în imagini (păr șaten, piele închisă la culoare). Acest lucru poate fi privit, deopotrivă, ca un rezultat bun sau ca un rezultat mai puțin bun. Din perspectiva obținerii a unor imagini cu culori plauzibile, rezultatele sunt

bune. Din prisma obținerii unor rezultate cât mai apropiate de imaginea originală, rezultatele lasă de dorit. Imaginile din figura de mai jos au fost trecute în această secțiune, ca rezultate mai puțin bune, privind din a doilea punct de vedere amintit. Mențiune: unele imagini originale conțin "dâre" de culoare (acestea nu au fost obținute ulterior).

Figura 5

Autoencoder - Peisaje

Pe seturile de date cu peisaje (restrâns și extins) s-a antrenat un singur model de tip autoencoder, anume cel care a dat rezultatele cele mai bune pe setul de date cu imagini de tip portret: configurația cu skip connections și Conv2DTranspose. Rezultatele se pot vedea în Figura 6 și în Figura 7.

O primă observație ar fi că rețeaua neuronală colorează în multe din cazuri cerul de un albastru aprins, chiar dacă relitatea este diferită sau dacă restul imaginii este slab saturată. Ambele modele au rezultate asemănătoare, deși antrenamentele diferă prin dimensiunea seturilor de date și a numărului de epoci rulate.

Cu toate acestea, dacă după antrenament se aplică o intensificare a saturației imaginii (a celei reieșite din prima configurație), se obțin următoarele rezultate, redate mai jos în Figura 7.

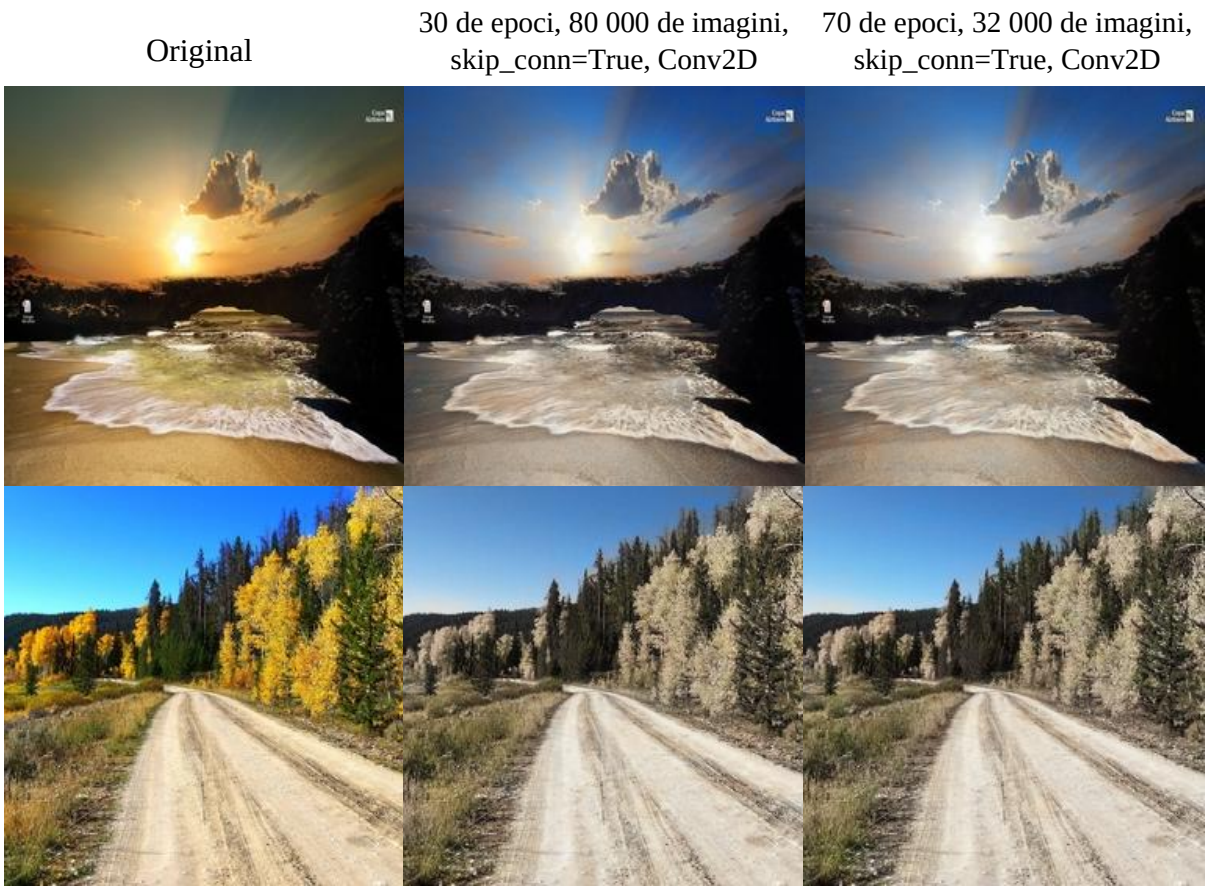


Figura 6

Figura 7

Din rezultatele de după procesări (Figura 7) se poate trage concluzia că în imaginile generate de autoencoder este suficientă informație din punct de vedere al culorii, astfel încât, prin saturarea acesteia, culorile să devină unele naturale.

GAN - Portrete

Pe setul de portrete s-au antrenat 15 modele de rețele GAN, numărul de epoci fiind variabil (între 44 și 100). Se va observa că numărul de epoci nu este un parametru atât de important (de la o valoare suficient de mare încolo), întrucât există modele antrenate mai puțin care au dat rezultate mulțumitoare. 12 din cele 15 modele au fiecare parametrul loss_weights=[20, 80] și provin din: 3 discriminatori (de tip PatchGAN – varianta originală din [2] și 2 derivați din aceasta, cu output 4x4, respectiv 8x8) și 4 generatori (asemănători cu cele 4 autoencodere amintite mai sus). Până aici sunt 3x4=12 GAN-uri. Apoi s-a luat cel mai bun model din acestea și s-a mai antrenat de 3 ori, cu parametrul loss_weights luând valorile [1, 99], [5, 95] și [35, 65]. În Figura 8 de mai jos sunt redate rezultate de la 10 dintre cele 15 modele, celelalte 5 neavând rezultate care să fi meritat să fie amintite. Totuși, este de reținut configurațiile acestora, pentru a trage o concluzie, dacă este posibil, în legătură cu motivele pentru care aceste modele nu au dat rezultate favorabile. Imaginea originală a fost așezată pe fiecare rând de imagini pentru a realiza comparații mai ușor. Se vor face următoarele convenții de notare pentru parametrii modelelor:

- sc1 = generator cu skip connections; sc0 = generator fără skip connections;
- c2D = generator cu metodă de upsampling Conv2DTranspose;
- u2D = generator cu metodă de upsampling UpSampling2D;
- dO = discriminator de tip PatchGAN, varianta originală din [2];
- d4 = discriminatpr de tip PatchGAN cu convoluționare pe fiecare patch în parte, cu output 4x4;
- d8 = discriminatpr de tip PatchGAN cu convoluționare pe fiecare patch în parte, cu output 8x8.
- lw = loss_weights

Original | sc1, c2D, dO, lw=[20, 80] | sc1, c2D, d4, lw=[20, 80]

Original | sc0, u2D, dO, lw=[20, 80] | sc1, c2D, d8, lw=[20, 80]

Original | sc1, c2D, d8, lw=[1, 99] | sc1, c2D, d8, lw=[35, 65]

Original | sc1, u2D, d4, lw=[20, 80] | sc1, u2D, d8, lw=[20, 80]

Original sc0, c2D, d4, lw=[20, 80] sc0, c2D, d8, lw=[20, 80]

Figura 8

GAN-urile care au în componență discriminatorul de tip PatchGAN în varianta originală dau rezultate slabe, imaginile rezultat părând a fi formate din blocuri mari alăturate. GAN-urile care au în componență generatorul fără skip connections produc imagini pixelate. Existența skip connections ajută rețeaua prin faptul că producerea rezultatului final este influențată de structura imaginii originale mai mult decât în lipsa acestor conexiuni suplimentare.

Configurațiile celor 5 modele ale căror rezultate nu au fost prezentate în Figura 8 sunt: (sc1, u2D, dO, lw=[20, 80]), (sc0, c2D, dO, lw=[20, 80]), (sc0, u2D, d4, lw=[20, 80]), (sc0, u2D, d8, lw=[20, 80]), (sc1, c2D, d8, lw=[5, 95]). Rezultatul ultimei dintre acestea nu a fost trecut deoarece este asemănător cu cele ale configurațiilor (sc1, c2D, d8, lw=[1, 99]) și (sc1, c2D, d8, lw=[35, 65]) (doar valorile loss_weights diferă). Primele două modele menționate în lista celor cu rezultate mai slabe au în componența lor discriminatorul de tip PatchGAN varianta originală și de aceea, imaginile colorate obținute sunt foarte neclare. Următoarele două modele nu au în componența lor un generator cu skip connections și, de aceea, rezultatele lor sunt pixelate. Pe imaginea originală din Figura 8, configurația (sc1, u2D, d4, lw=[20, 80]) dă rezultate bune, însă acestea au o tendință generală de a fi oarecum desaturate. Exemple în acest sens pot fi vizualizate în Figura 9 de mai jos. Un motiv pentru care această metodă obține rezultate puțin mai slabe decât Conv2DTranspose ar putea fi acela că UpSampling2D mărește imaginea prin metoda clasică *nearest neighbours*, spre deosebire Conv2DTranspose care face acest lucru cu ajutorul unui kernel învățat.

sc1, u2D, d4, lw=[20, 80] sc1, c2D, d8, lw=[20, 80]

Figura 9

Cele mai bune rezultate, din punct de vedere personal, rămân cele generate de GAN-uri care au generator cu skip connections, metodă de upsampling Conv2DTranspose și discriminator de tip PatchGAN varianta convoluționată pe patch-uri de imagine, atât cu output-ul 4x4, cât și cu output-ul 8x8. Comparându-le pe acestea două, se poate observa că modelul cu discriminator d8 dă rezultate ce conțin câteva pete portocalii și albastre, însă chipul persoanei are culori mai aprinse. Numărul de patch-uri convoluționate fiind mai mare, dimensiunea fiecăruia este mai mică, ceea ce, în mod intuitiv, ar conduce la concluzia că "atenția" la detalii este mai mare. Saturația mai mare a culorilor ar corespunde faptului că modelul "riscă" ma mult, de aici provenind și zonele colorate în portocaliu și albastru puse aparent la întâmplare. Din punct de vedere personal, varianta de GAN câștigătoare este cea cu configurația (sc1, c2D, d8, lw=[20, 80]).

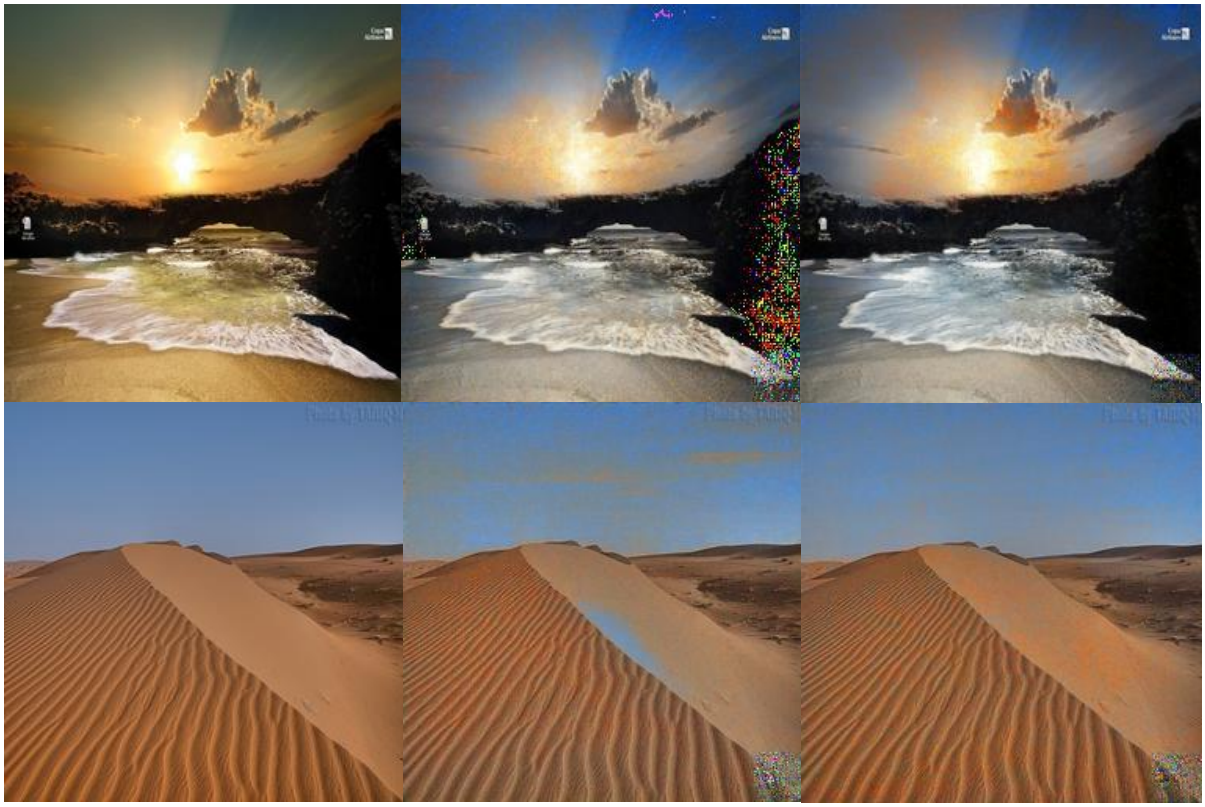
Valorile 20 și 80 ca ponderi ale loss-urilor generatorului și discriminatorului s-au dovedit a da rezultate bune, ele având semnificația că este de 4 ori mai important a minimiza loss-ul discriminatorului decât a-l minimiza pe cel al generatorului. Pentru a face teste și cu alte valori pentru loss_weights s-a luat modelul cel mai bun menționat mai sus și s-au efectat încă 3

antrenamente, cu valorile pentru loss_weights [1, 99] (valoarea indicată în [2]), [5, 95] și [35, 65]. Rezultatele provenite din configurațiile cu loss-urile [1, 99] și [35, 65] prezentate în Figura 8 sunt, după cum se poate observa, apropiate de cel dat de configurația considerată câștigătoare (sc1, c2D, d8, lw=[20, 80]), însă zona de imagine corespunzătoare chipului persoanei este puțin mai desaturată față de cea din rezultatele date de (sc1, c2D, d8, lw=[20, 80]).

GAN – Peisaje

Pe setul de date cu imagini de tip peisaj a fost testat un singur model GAN, anume varianta care s-a descurcat cel mai bine pe imagini de tip portret (configurația (sc1, c2D, d8, lw=[20, 80])). Au fost efectuate 2 antrenamente: unul pe setul de date restrâns (~32 000 de imagini), unul pe setul de date extins (~82 000 de imagini). Primul antrenament a fost rulat 30 de epoci, în timp ce al doilea 15, numărul mic fiind din cauza duratei mari în care se desfășoară antrenamentul (15 epoci în aproximativ 21 de ore), care este incompatibilă cu conexiunea instabilă a plăcii video în mediul de luru. Rezultatele pot fi vizualizate în Figura 10 de mai jos.

Original | 32 000 imagini, 30 epoci | 82 000 imagini, 15 epoci



/

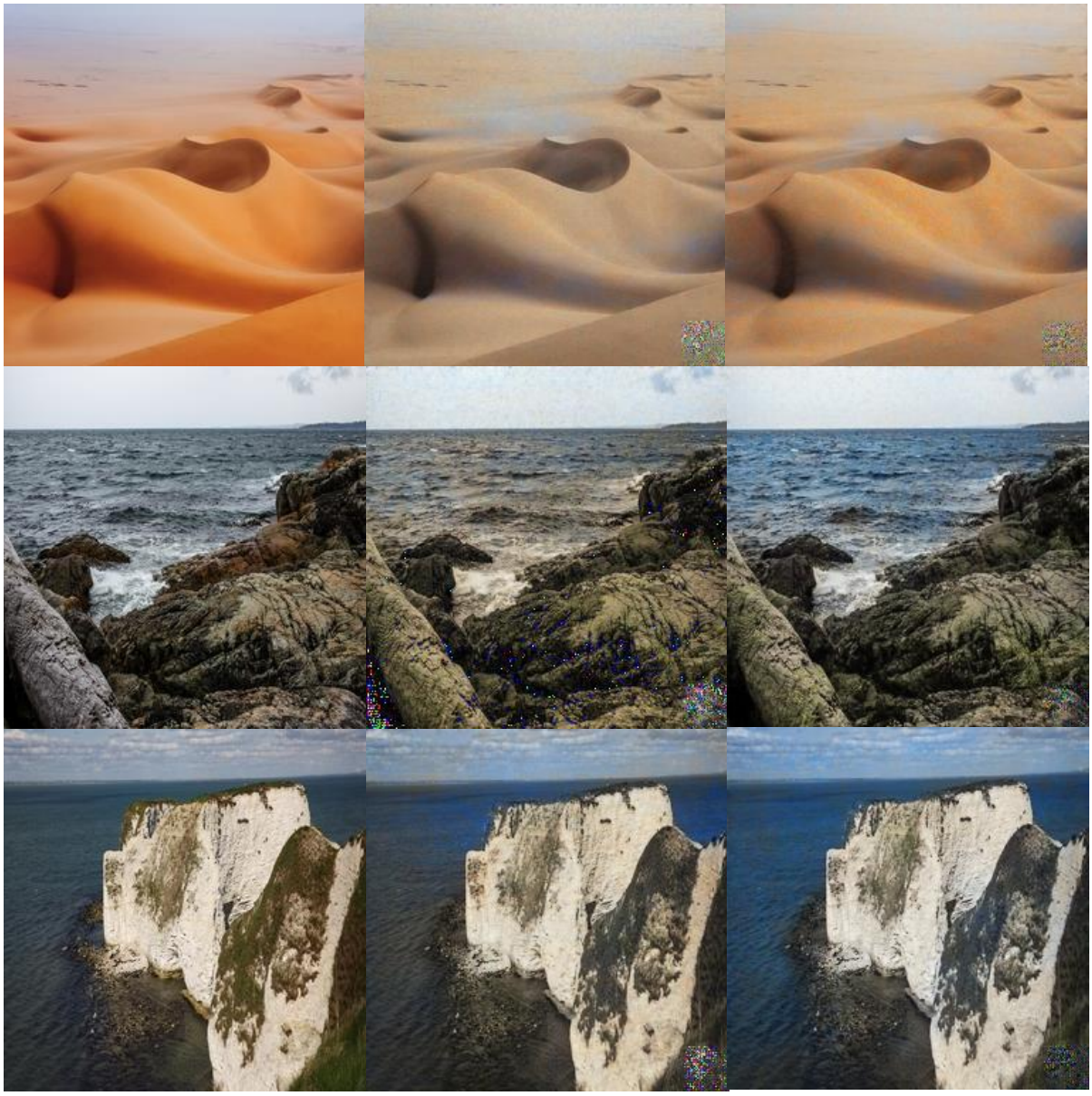


Figura 10

Se observă că al doilea GAN, deși antrenat timp de mai puține epoci, are rezultate mai bune decât primul. Acest fapt s-ar datora setului mai mare de date pe care a fost antrenat, acesta fiind singurul parametru schimbat din configurațiile lor.

Autoencoder versus GAN - Portrete

Pentru o comparație a celor 2 arhitecturi se vor lua rezultatele cele mai bune de la amândouă, pentru setul de date cu portrete și setul de date cu peisaje.

Metrica folosită este diferența absolută medie în medie între valorile pixelilor imaginilor generate și valorile pixelilor imaginilor originale (diferență per pixel).

Prin urmare, după antrenarea celei mai bune versiuni de model pe 32 768 de imagini, autoencoder-ul și GAN-ul s-au descurcat, pe același set de date de test (ce conține 3072 de imagini), astfel (imaginile nesuferind o postprocesare ulterioară):

	Autoencoder	GAN
Diferența medie minimă	0,009	0,011
Diferența medie în medie	0,037	0,042
Diferența medie maximă	0,161	0,173
Număr de epoci	100	60
Set de date	32768 imagini antrenament, 3072 imagini test	

Metrica este scrisă în coloana din tabel colorată cu galben. Minimul posibil este 0, iar maximul posibil este 1 (valorile pixelilor sunt din intervalul [0, 1]). Cu cât valoarea metricii este mai mică, cu atât modelul a dat rezultate mai bune, relativ la aceasta. S-ar putea realiza o corelație între numărul de epoci și diferența medie, însă nu ar fi neapărat expresivă. Din comparația acestor valori, autoencoder-ul iese ca model câștigător.

Figurile 11 și 12 de mai jos conțin graficele loss-urilor (și a *learning rate*-ului) pentru cele 2 modele analizate mai sus.

Scăderea bruscă a learning rate-ului autoencoder-ului provine din *callback*-ul din Keras ReduceOnPlateau. Acesta a acționat când graficul loss-urilor a devenit unul de tip platou, fără succes însă, deoarece loss-urile nu au scăzut mai mult de atât. Totuși, loss-urile erau deja suficient de mici și ating convergența.

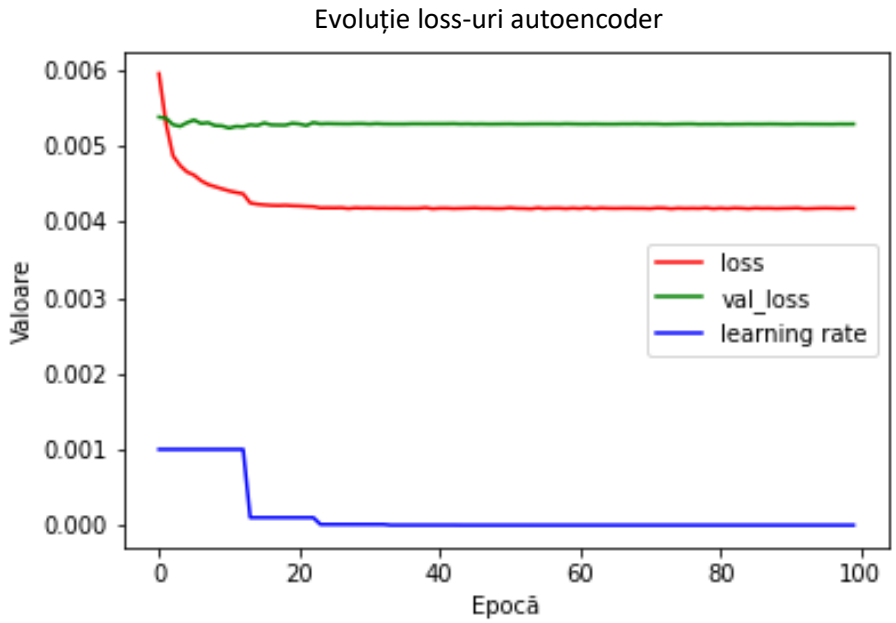


Figura 11

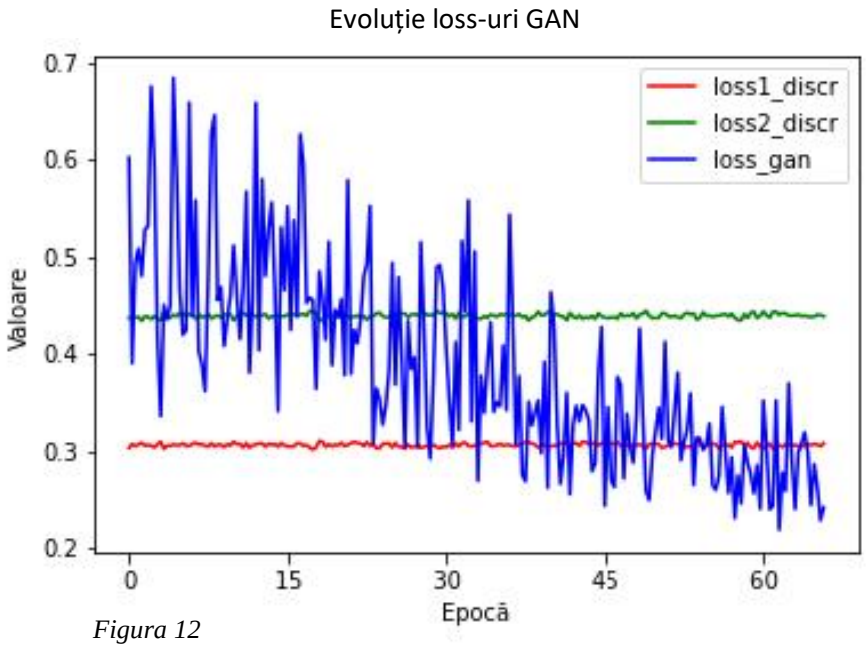


Figura 12

Cele 2 loss-uri ale discriminatorului (primul provenind din antrenarea cu imagini reale și al doilea din antrenarea cu imagini generate) sunt relativ constante, însă loss-ul GAN-ului scade cu fluctuații, dar continuu, per ansamblu. Loss-ul mai mare al GAN-ului față de cel al autoencoder-ului susține, la fel ca metrica folosită, faptul că rezultatele autoencoder-ului sunt mai bune decât cele ale GAN-ului.

În urma acestei analize concluzia evidentă ar fi ca autoencoder-ul este superior GAN-ului din punct de vedere al rezultatelor, dacă acestea se evaluează prin formule. Dacă vorbim însă despre metrica factorului uman, opinia proprie este că GAN-ul e rețeaua care s-a descurcat mai bine, deoarece culorile chipului uman sunt mai aprinse și per total, imaginea este mai expresivă. Este adevărat, totuși, că, de multe ori, culorile fundalului din imaginea obținută diferă semnificativ de culorile din imaginea originală (ceea ce probabil și contribuie mult la diferența mare obținută). Personal, cred că cea mai importantă opinie este cea a privitorului, deoarece el face de fapt parte din publicul pentru care acest proiect se realizează.

În Figura 13 de mai jos sunt prezentate câteva imagini care susțin acest argument.

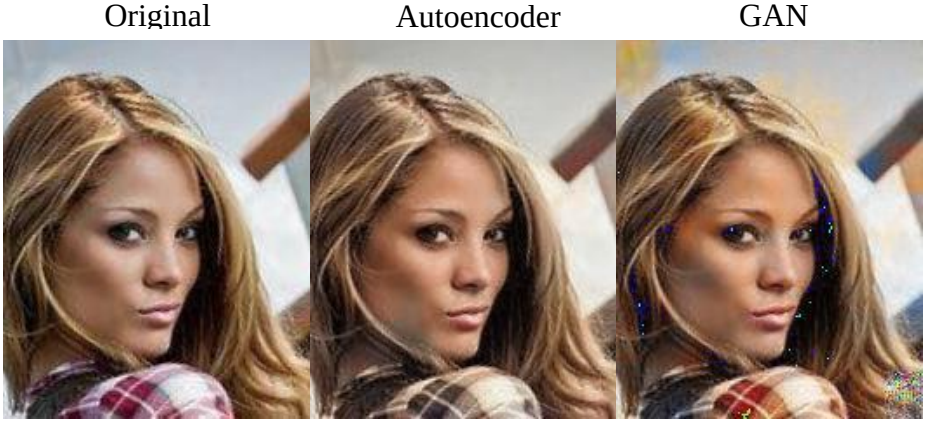


Figura 13

Autoencoder versus GAN – Peisaje

Pentru comparația celor 2 arhitecturi pentru imaginile de tip peisaj, se vor expune rezultatele metricii utilizate și în comparația pentru portrete de mai sus. Setul de date extins conține 81920 de imagini, cu 8192 de imagini în setul de test, iar setul restrâns conține 32768 de imagini, cu 4096 de imagini în setul de test.

	AE - set restrâns	G - set restrâns	AE - set extins	G – set extins
Diferența medie minimă	0,021	0,029	0,019	0,024
Diferență medie în medie	0,086	0,087	0,083	0,159
Diferența medie maximă	0,443	0,415	0,459	0,456
Număr epoci	70	30	30	15
Set de date	32768 antrenament, 4096 test		81920 antrenament, 8192 test	

Conform metricii diferenței medii în medie, autoencoder-ul a dat rezultate mai bune pe peisaje decât GAN-ul și de această dată.

De asemenea, comparația cu valorile metricii pe portrete denotă că modelele s-au descurcat, în general, mai bine pe antrenamentul cu portrete. Acest lucru poate fi cauzat de faptul că imaginile de tip portret sunt mai „standard” decât cele de peisaje: toate portretele au un lucru important în comun, o zonă mare centrată de culoare crem sau maro. În schimb, peisajele pot avea albastru sau crem/maro în partea de jos, sau pot avea culoare verde ca predominantă în tot cadrul.

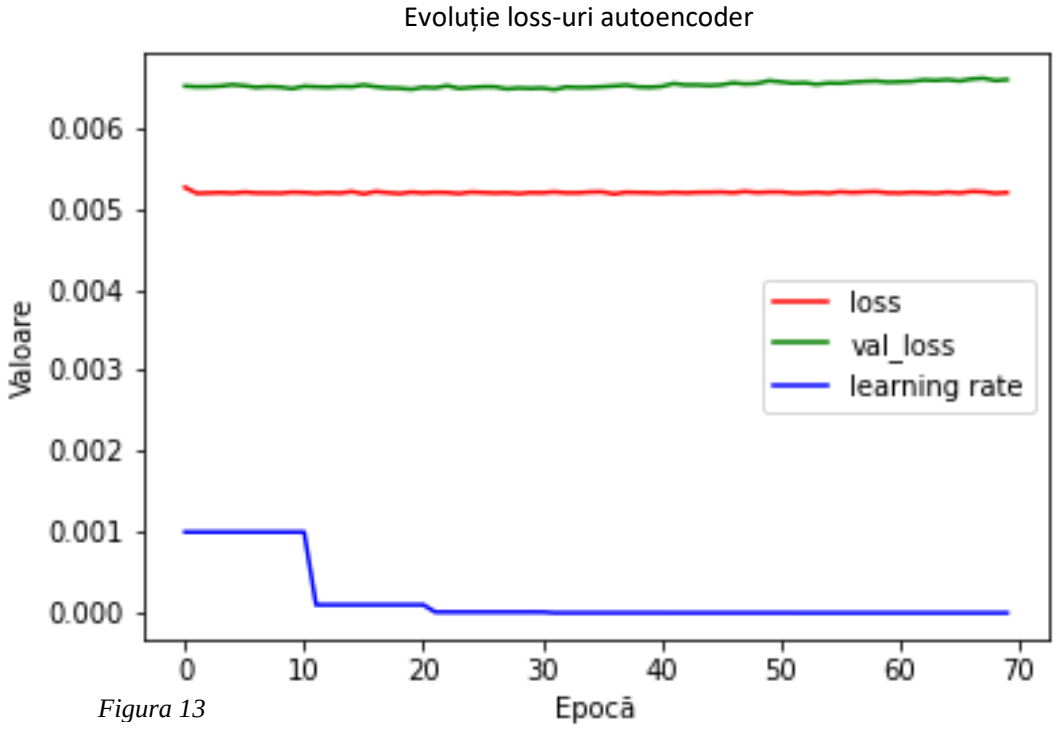


Figura 13

În Figurile 13 și 14 sunt prezentate graficele loss-urilor pentru cele două rețele antrenate pe setul de peisaje restrâns. Modelele au avut aproximativ aceleași comportamente ca pe setul de imagini portret, cu mici excepții: aici loss-ul autoencoder-ului este aproape constant, iar loss-ul GAN-ului este mare la început, însă scade brusc după doar câteva epoci, iar apoi continuă să scadă lin și constant.

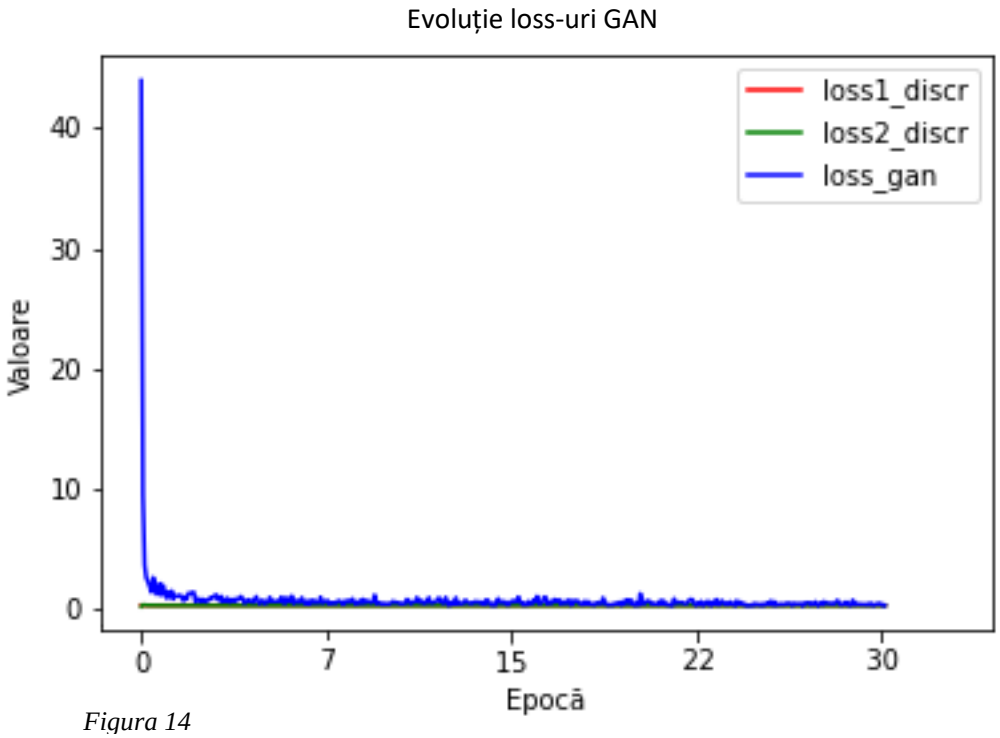


Figura 14

Mai jos, în Figura 15 se găsesc rezultate de la modelele antrenate pe setul mic de peisaje care, tind totuși să contrazică verdictul dat de valoarea metricii utilizate.

Original Autoencoder GAN

Figura 15

Verdict final

Având în vedere analiza modelelor și comparația de mai sus dintre rezultate, în toate cazurile, autoencoder-ul reiese a fi un model mai performant decât GAN-ul, relativ la metrica utilizată. Din punct de vedere vizual, ambele modele dau ca rezultat imagini plauzibile coloristic în general. Personificând modelele, se poate spune că autoencoder-ul are un comportament mai rezervat, alegând să coloreze în culori cu tente maronii. Acesta preferă mai degrabă să nu "iasă din tipar" cu niște culori de o intensitate sau culoare extremă. La polul opus, GAN-ul dă ca rezultate imagini cu culori mai profunde și intense deoarece riscă mai mult, însă acest lucru atrage după el și existența unor pete de o culoare țipătoare sau nepotrivită cu contextul. Avantaje și dezavantaje sunt de fiecare parte însă, personal, consider rețeaua GAN ca fiind mai potrivită pentru acest task și cu un potențial mai mare de exploatare, în vederea unor viitoare rezultate mai bune.

Concluzii și direcții viitoare

Colorare a imaginilor este proces relativ complex și poate fi realizat în multe moduri. Încercând câteva dintre ele, am reușit să înțeleg ce fel de particularități ale unui model l-a făcut mai puternic sau mai slab și de ce. Acesta poate constitui un punct de plecare pentru o eventuală viitoare cercetare în domeniul colorării imaginilor sau a clasei de probleme Image-to-Image Translation în general. Acestă posibilă cercetare ar putea începe cu continuarea tratării problemei din această lucrare. Câteva idei în acest sens ar fi:

- implementarea unei rețele GAN cu clase pentru fiecare tip de imagine, pentru a nu rula câte 2 antrenamente (câte au fost rulate în această lucrare) sau mai multe pentru fiecare astfel de tip;
- implementarea loss-ului Wasserstein pentru rețeaua GAN. Acesta urmărește o mai bună delimitare între scorurile pentru imaginile reale și pentru imaginile generate;
- antrenarea unui model de 2 ori: o dată cu imagini alb-negru pentru a obține imagini colorate, apoi cu imagini colorate de către model pentru a obține altele noi, și mai frumos și corect colorate;
- implementarea unui CycleGAN, rețea mai puternică decât o rețea GAN simplă;
- implementarea unei rețele de tip ProGAN, pentru a putea antrena pe seturi de date cu imagini de dimensiuni mari;
- găsirea unei plăci video ce oferă o conexiune stabilă;
- pasul următor: colorarea de videoclipuri.

Bibliografie

[1] Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S. Lin, Tianhe Yu, Alexei A. Efros (2017), Real-Time User-Guided Image Colorization with Learned Deep Priors

[2] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros (2018), Image-to-Image Translation with Conditional Adversarial Networks

[3] https://github.com/jantic/DeOldify/blob/master/README.md DeOldify Project

[4] Richard Zhang, Phillip Isola, Alexei A. Efros (2016), Colourful Image Colorization

[5]https://blog.data.gov.sg/bringing-black-and-white-photos-to-life-using-colourise-sg-435ae5cc5036 Colourful.sg Project

[6] Satoshu Iizuka, Edgar Simo-Serra, Hiroshi Ishikawa (2016), Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification

[7] Keiron O'Shea, Ryan Nash (2015), An Introduction to Convolutional Neural Networks

[8] Jurgen Schmidhuber, The Swiss AI Lab IDSIA, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, University of Lugano & SUPSI (2014), Deep Learning in Neural Networks: An Overview

[9] Olaf Ronneberger, Philipp Fischer, Thomas Brox (2015), U-Net: Convolutional Networks for Biomedical Image Segmentation

[10] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio (2014), Generative Adversarial Networks

[11] Mehdi Mirza, Simon Osindero (2014), Conditional Generative Adversarial Nets

[12] https://www.kaggle.com/jessicali9530/celeba-dataset Kaggle, CelebA Dataset

[13] http://places.csail.mit.edu/ MIT, Places Dataset

Anexa 1 – Tehnologii folosite

Pentru a prezenta modelele neuronale folosite, voi descrie crearea lor de la conceptele cele mai de bază până la conceptele cele mai complexe.

Rețele convoluționale [7]

Straturile cele mai importante folosite la toate arhitecturile realizate sunt straturi convoluționale. Rețelele convoluționale sunt un algoritm de deep learning care preia o imagine dată ca input și acordă importanță (prin weight-uri și bias-uri) diferitelor caracteristici (feature-uri) din aceasta, fiind capabil să facă diferența între ele (să învețe). Ce învață de fapt o rețea convoluțională este o matrice (kernel) care se aplică în mod repetat peste imagine, facându-se operații de înmulțire și adunare între acest kernel și porțiuni din imagine. Valorile kernel-ului sunt învățate prin minimizarea unei funcții de cost, care face diferența dintre output-ul rețelei și target-ul/label-ul imaginii. Label-ul imaginii este setat în funcție de ce se dorește ca rețeaua să învețe (de exemplu poate fi unul sau mai multe numere sau tot o imagine).

Rețele convoluționale - Motivație

O imagine este, în fapt, o matrice de valori ale pixelilor, așa că în loc de o rețea convoluțională ar putea fi folosită, teoretic, o rețea neuronală de tip Multilayer Perceptron (căreia îi este dat ca input vectorul obținut prin înșiruirea liniilor din matricea corespunzătoare imaginii). Totuși, pixelii dintr-o imagine sunt puternic interconectați cu cei din cel puțin imediata lor apropiere, aceste corelații reușind să fie identificate doar din structura de matrice, nu și de vector. Așadar, optarea pentru rețele convoluționale în loc de cele mai simple de tip MLP este motivată prin faptul că, spre deosebire de MLP, rețelele convoluționale reușesc să capteze dependențele spațiale și temporale din date.

Arhitecturi folosite: Arhitectura 1 - Autoencoder

1.1. Autoencoder simplu [8]

Prima abordare de rezolvare a problemei este cu un autoencoder. Autoencoderele sunt rețele neuronale convoluționale care codifică input-ul într-o reprezentare redusă ca dimensiuni și apoi îl reconstruiesc, din codificarea obținută. De-a lungul straturilor, imaginea se micșorează treptat prin straturi Conv2D sau de MaxPooling (straturi de encoding), până ajunge la dimensiunea "de mijloc" corespunzătoare codificării (la stratul bottleneck), iar apoi își mărește treptat dimensiunile prin straturi Conv2DTranspose sau UpSampling2D (straturi de decoding), până

ajunge sa aibă mărimea imaginii inițiale. Ce învață o rețea de tip autoencoder este acea codificare comprimată, iar pentru că output-ul rețelei trebuie să semene cât mai mult cu input-ul, ce se reține de fapt în acea codificare sunt caracteristicile cele mai relevante ale input-ului. Această metodă de învățare este nesupervizată sau selfsupervised deoarece target-ul reprezintă chiar imaginea input. Pentru problema tratată în lucrare, autoencoder-ul primește ca input o serie de imagini alb-negru care au ca label echivalentul lor color. Așadar, autoencoder-ul folosit a fost tratat ca un algoritm de învățare supervizată, astfel încât output-ul acestuia să nu fie reproducerea imaginii input, ci imaginea input colorată.

Autoencoder simplu - Motivație

Autoencoder-ul a fost utilizat deoarece reprezintă o metodă prin care se învață la nivel profund detaliile semnificative ale imaginii. Un alt motiv pentru alegerea acestei arhitecturi este că, prin reducerea dimensiunilor imaginii și, implicit, a dimensiunilor straturilor rețelei, antrenamentul are nevoie de resurse de timp și spațiu considerabil mai puține față de cât ar avea nevoie o rețea convoluțională cu același număr de straturi dar care nu redimensionează imaginea de-a lungul straturilor. Autoencoder-ul dispune, cel puțin pentru problema de față, de un raport calitate-resurse folosite suficient de bun.

1.2. Autoencoder de tip U-Net [9]

O rețea de tip U-Net nu este altceva decât un autoencoder la care se adaugă niște legături suplimentare între straturi. Litera “U” din nume provine din faptul că rețeaua, poate fi dispusă/imaginată având forma acestei litere, o caracteristică importantă fiind simetria straturilor. La autoencoder fiecare strat are legături doar cu stratul următor, însă U-Net vine cu o modificare care pe cât de ușor este de realizat, pe atât de semnificative sunt diferențele în rezultatele pe care le produce. Modificarea este că, în straturile conținute de la bottleneck până la final nu intră doar weight-urile stratului anterior corespunzător fiecăruia, ci și stratul ce se află în simetrie cu acesta (ultimul strat e conectat la penultimul și la primul, penultimul strat este conectat la antepenultimul și al doilea etc.). Acest tip de conectare se numește skip-connection deoarece nu se realizează de la un strat la stratul imediat următor.

Autoencoder de tip U-Net - Motivație

Autoencoder-ul folosit inițial a fost transformat într-o rețea U-Net deoarece aceasta din urmă aduce schimbări pozitive semnificative în rezultate. Straturile de decoding nu sunt influențate

doar de stratul anterior corespunzător, care poate rezulta într-o versiune deformată a imaginii, ci și de straturile simetrice lor (straturile de encoding), care păstrează structura originală a imaginii. Astfel, imaginea "decodată" (output-ul) va păstra structura imaginii originale.

GAN [10]

Rețelele de tip GAN (Generative Adversarial Networks) sunt modele arhitecturale cu un grad de complexitate mai ridicat decât al rețelelor descrise mai sus, deoarece sunt formate din două rețele: un generator și un discriminator. Aceste două rețele concurează una împotriva alteia: generatorul primește ca input un vector de random noise și din acesta încearcă să producă ca output ceva ce are sens pentru ochiul uman. Discriminatorul primește ca input imaginea dată de generator ca output și încearcă să dea un verdict, anume dacă este reală sau este generată. Practic, prin antrenare, generatorul încearcă să prezică ceva cât mai verosimil, pentru a-l face pe discriminator să "creadă" că este real, iar discriminatorul încearcă să dea verdicte cât mai aproape de adevăr, pentru a distinge cât mai bine între ce este real și ce este generat.

GAN - Motivație

Încă de la apariția rețelelor de tip GAN, acestea s-au dovedit a fi un instrument extrem de puternic în Computer Graphics, obținând rezultate uimitoare pe o gama largă de task-uri. Acestea au devenit din ce în ce mai performante datorită dezvoltării constante de noi variații ale arhitecturii rețelelor componente (combinații de straturi, funcții de loss diferite, mai mulți generatori sau mai mulți discriminatori etc.), motiv pentru care am ales o astfel de arhitectură pentru task-ul de colorare de imagini.

Arhitectura 2 - Conditional GAN (cGAN) [11]

Conditional GAN este un tip de rețea GAN în care generatorul este condiționat ca input. Acest fapt face ca cGAN-urile să aibă aplicabilitate pentru task-ul de Image-to-Image Translation. Generatorul primește ca input, pe lângă vectorul de random noise, și imaginea care se dorește a fi transformată, aceasta fiind diferența între un cGAN și un GAN. Astfel imaginea generată păstrează fidel structura imaginii de la care se pleacă.

cGAN - Motivație

Am ales să implementez o arhitectura de tip cGAN deoarece este exclus ca imaginile să își schimbe structura sau să se deformeze pe parcursul procesului de colorare. De fapt, pentru a avea siguranța că imaginea își păstrează structura originală, am eliminat vectorul de random noise, eliminând totodată, oarecum, aspectul generativ al rețelei. Partea de "generator" este, în fapt, un autoencoder, cu exact aceleași caracteristici descrise mai sus. Rețeaua cGAN rezultată poate fi privită ca un autoencoder care este penalizat pentru greșeli de o altă rețea (discriminatorul) care învață să dea verdicte din ce în ce mai aproape de adevăr. Discriminatorul este o rețea convoluțională, având ca output o matrice cu valori care reprezintă verdicte (real sau generat).

Arhitectura de cGAN este implementată în lucrare în două moduri: generatorul este același în ambele arhitecturi (cel descris mai sus), ce diferă este însă discriminatorul. Acesta din urmă este construit în cele două moduri după aceeași idee de bază. Voi prezenta această idee și motivația din spatele ei, iar apoi voi puncta diferențele dintre cele două implementări.

Discriminator PatchGAN [2]

Discriminatorul dă verdict la nivel de patch-uri (bucăți de imagine), și nu la toată imaginea. Acest tip de discriminator este de tip PatchGAN [1]. La discriminatorul PatchGAN, fiecare verdict (număr) din matricea rezultată este calculat astfel încât să corespundă unui patch de dimensiunea 70x70 din imaginea input. Această dimensiune se numește receptive field (numărul de pixeli din imaginea input mapați la un pixel/număr din imaginea/matricea output). Acest tip de discriminator are avantajul că poate fi aplicat indiferent de dimensiunea imaginii input.

Motivație discriminator de tip PatchGAN

Luat un pixel dintr-o imagine input, se dorește a-i prezice valoarea în imaginea output. Loss-urile folosite de obicei, L1 sau L2, se încearcă să fie minimizate în medie. Dacă pixelul în cauză aparține de o structura tip muchie, acesta ar trebui să aibă o valoare prezisă foarte mare sau una foarte mică (corespunzând unei culori foarte deschise sau foarte închise). Însă pentru că se dorește ca diferența între pixelul target și pixelul prezis să fie cât mai mică, rețeaua nu-și va asuma să acorde pixelului prezis o valoare foarte mare sau foarte mică, deoarece diferența amintită ar avea astfel, o valoare foarte mare. Prin urmare, valoarea pixelului prezis va fi una medie, ceea ce are ca efect ca muchiile, colțurile, zonele cu schimbări bruște de culoare din imagine să fie blurate. Loss-urile L1 sau L2 obțin structura de bază a imaginii, însă detaliile nu sunt redate la claritate mulțumitoare.

Discriminatorul PatchGAN se “uită” pe rând la câte o subimagine din imaginea mare, corectând activitatea generatorului, prin verdicte de “fals”/“generat” pentru ceea ce găsește a fi blurat (ca efect al utilizării L1 sau L2). Așadar, imaginea output va avea claritate bună, chiar dacă loss-ul a rămas în continuare L1 sau L2.

2.1. Discriminatorul este convoluționat pe imaginea în întregime

Acest discriminator este cel PatchGAN, cu excepția faptului că receptive field-ul diferă și că este adăugat un strat în plus. Convoluțiile sunt aplicate pe toată imaginea, iar patch-urile se suprapun.

2.2. Discriminatorul este convoluționat pe fiecare patch separat

Se împarte imaginea în numărul de patch-uri dorit (acest proces realizându-se prin intermediul unui strat special din biblioteca Keras: Lambda), patch-urile nesuprapunându-se, iar apoi discriminatorul este aplicat pe fiecare dintre aceste patch-uri (subimagini). Rezultatul se obține prin concatenarea image-wise “subrezultatelor” obținute de la fiecare subimagine.

Acest tip de discriminator a fost implementat pentru ca acesta să aibă un domeniu de aplicabilitate mai restrâns și a acorda o importanță deosebită pentru bucăți mici din imagine, bineînțeles, având ca scop final o claritate mai bună a imaginii.