

UNIVERSITATEA DIN BUCUREŞTI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

LUCRARE DE LICENȚĂ

COORDONATOR ȘTIINȚIFIC
Conf. Dr. Marius-Nicolae Popescu

STUDENT
Petrișor-Ionuț Calofir

BUCUREŞTI
2018

UNIVERSITATEA DIN BUCUREŞTI
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

LUCRARE DE LICENȚĂ

Transferul Stilului Artistic

COORDONATOR ȘTIINȚIFIC
Conf. Dr. Marius-Nicolae Popescu

STUDENT
Petrișor-Ionuț Calofir

BUCUREŞTI
2018

Cuprins

1 Transferul stilului artistic	4
1.1 Introducere	4
1.2 Prezentarea lucrării	4
1.2.1 Un algoritm neural al stilului artistic	7
1.2.2 Transferul stilului artistic fotografic	7
1.2.3 Transferul stilului artistic în timp real	7
1.2.4 Compararea metodelor	8
2 Un algoritm neural al stilului artistic	9
2.1 Introducere	9
2.1.1 Generarea conținutului	10
2.1.2 Generarea stilului	11
2.2 Metodă	15
2.3 Detalii de implementare	17
2.3.1 Python	20
2.3.2 TensorFlow	20
2.3.3 ImageNet	20
2.3.4 Inițializarea Xavier	21
2.3.5 ReLU	21
2.3.6 Coborâre pe gradient	21
2.3.7 Backpropagation	22
2.3.8 Optimizatorul Adam	22
2.3.9 Rata de învățare	23
2.4 Rezultate și comparații	24
3 Transferul stilului artistic fotografic	29
3.1 Introducere	29

3.1.1	Etichetări semantice	30
3.2	Metodă	30
3.3	Detalii de implementare	31
3.4	Rezultate și comparații	33
4	Transferul stilului artistic în timp real	36
4.1	Introducere	36
4.2	Metodă	36
4.3	Detalii de implementare	37
4.3.1	Normalizarea datelor	38
4.3.2	Normalizarea batch-ului	39
4.3.3	Tangenta hiperbolică	40
4.3.4	Layer transpus de conoluție	40
4.3.5	Rețele reziduale	41
4.3.6	Microsoft COCO	41
4.4	Rezultate și comparații	42
5	Compararea metodelor	44
6	Concluzii	46
6.1	Dezvoltări ulterioare	46

Capitolul 1

Transferul stilului artistic

1.1 Introducere



De-a lungul timpului, oamenii au reușit să altereză mediul înconjurător combinând diferite obiecte cu anumite stiluri, fie ele abstracte sau mai apropiate de realitate, obținând astfel opere de artă. Deși această acțiune este ușor de realizat de către oameni, fiind făcută chiar și de copii, este dificil să se găsească un algoritm care să simuleze acest proces deoarece creierul uman este complex, modul în care funcționează nefiind înțeles în totalitate de către oameni.

1.2 Prezentarea lucrării

În cele ce urmează o să prezint o modalitate pentru rezolvarea problemei din introducere, aceea de a combina conținutul unei poze cu stilul altor poze, obținând astfel poze artistice, asemănătoare cu cele create de om [5]. Apoi, la această metodă voi adăuga noi constrângeri pentru a putea crea poze cât mai fotografice, de a putea

schimba o poză din zi în noapte, de a schimba anotimpul din poză, etc [11]. În final, o să prezint o metodă pentru ca poza dorită să fie creată în timp real deoarece pentru a obține o poză cu metodele anterioare este necesar un timp mai mare de rulare [9].

Metodele de mai sus sunt posibile cu ajutorul învățării automate, termen care a fost inventat de către Arthur Samuel în 1959 și care a spus despre învățarea automată: "Un domeniu de studiu care oferă calculatorului abilitatea de a învăța fără a fi explicit programat.", mai exact cu ajutorul rețelelor neurale convoluționale (CNN) [1.2]. Având o poză inițializată cu un zgomot aleator sau chiar cu poza în care se află conținutul și o funcție de cost definită pe baza celor trei poze, poza inițială, poza cu conținut și poza cu stil, se vor modifica pixelii din poza inițială astfel încât costul să fie cât mai mic și astfel se va obține o poză artistică având conținutul pozei cu conținut și stilul pozei cu stil.

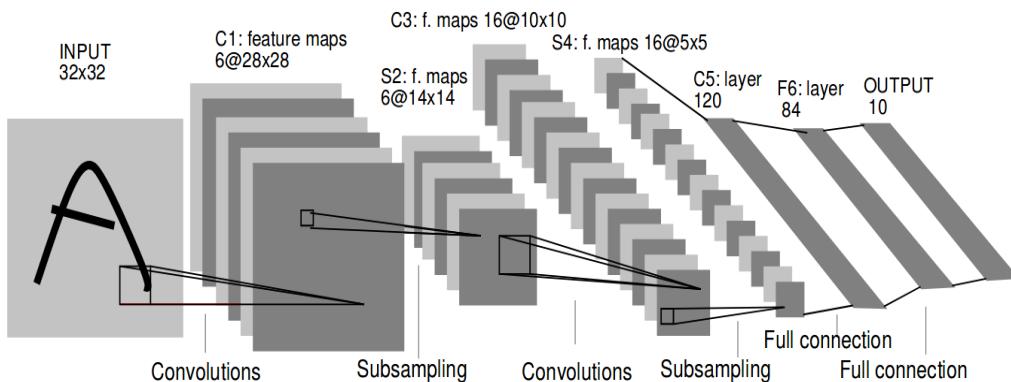


Figura 1.2: Rețea neurală convoluțională [1]

În figura [1.2] este prezentă arhitectura unei rețele neurale convoluționale. Acest tip de rețele au adus o îmbunătățire semnificativă în rezolvarea problemelor cu imagini.

Modul în care această rețea merge este unul simplu. La fiecare layer, numit layer convoluțional, sunt definite aşa numitele filtre sau kernele de o anumită dimensiune, această dimensiune reprezintă cât de mare este porțiunea din imagine pe care o vede un anumit filtru. Acest filtru, este plimbăt pe imagine, de-a lungul tuturor canalelor imaginii, din n în n pixeli și fiecare valoare din filtru este înmulțită cu pixelul corespunzător din porțiunea din imagine, iar apoi aceste valori sunt adunate pentru a

obține un singur număr, activarea neuronului. Numerele respective sunt puse într-o matrice ce poartă denumirea de matricea activărilor. Descrierea anterioară se poate vedea mai bine în figura [1.3].

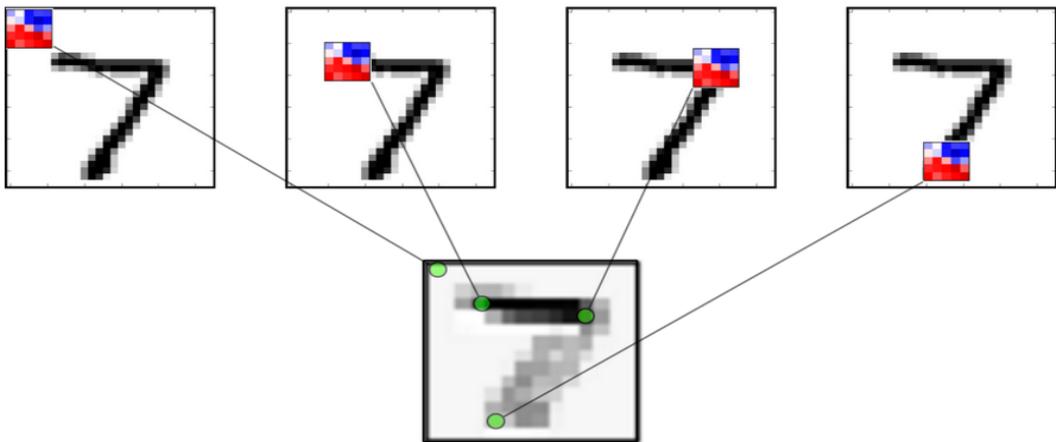


Figura 1.3: [16]

La un anumit layer există mai multe filtre, fiecare având anumite activări, iar aceste activări sunt așezate ca într-o stivă, precum se poate observa și în figura [1.2]. Peste aceste activări este aplicată o funcție neliniară, de obicei ReLU [2.3.5] și apoi sunt date ca intrare la următorul layer. Pentru ca rețeaua să învețe caracteristici mai complexe este nevoie ca imaginea inițială să fie micșorată pe parcursul rețelei. Pentru a micșora imaginea sunt posibile mai multe variante, o variantă este de a folosi un tip de layer numit pooling layer. Acesta se bazează pe plimbarea unei ferestre de o anumită dimensiune pe imagine, iar din toți pixelii aflați în fereastră se obține un singur număr reprezentând valoarea maximă a pixelilor sau media tuturor pixelilor [1.4].

O altă variantă de micșorare a pozei este aceea ca atunci când plimbăm filtrul pe o poză, să alegem un număr mai mare de pixeli dintre glisări, astfel încât să obținem o poză mai mică. În figura [1.2] mai apar și câteva layere care conțin valorile activărilor puse într-un vector, aceste layere nu păstrează spațialitatea pozei și sunt folosite după aplicarea layerelor convoluționale.

Ponderile filtrelor și al layerelor vectorizate sunt optimizate folosind algoritmul coborârii pe gradient [2.3.6] pe baza unei funcții de cost care se dorește a fi minimizată.

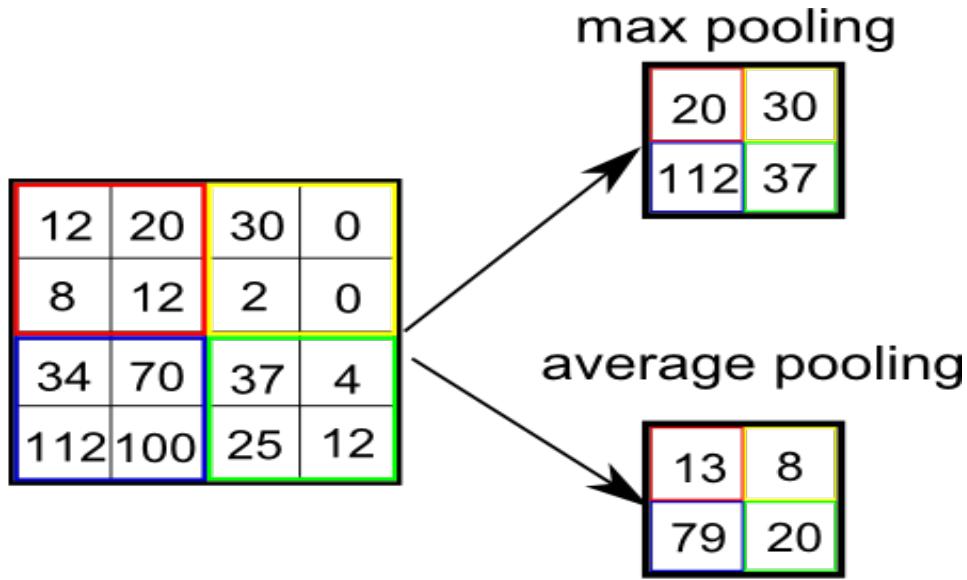


Figura 1.4: [20]

1.2.1 Un algoritm neural al stilului artistic

În acest capitol voi prezenta articolul lui Leon A. Gatys, A Neural Algorithm of Artistic Style [5], apărut în anul 2015. Acesta arată pentru prima dată faptul că stilul și conținutul unei poze sunt separabile în cadrul unei rețele neurale convoluționale și astfel, se pot obține poze artistice combinând două poze aleatoare.

1.2.2 Transferul stilului artistic fotografic

În acest capitol voi prezenta articolul lui Fujun Luan, Deep Photo Style Transfer [11], apărut în anul 2017. Acesta adaugă noi constrângeri metodei lui Leon A. Gatys pentru a obține poze cât mai fotografice, de a schimba o poză din zi în noapte, de a schimba anotimpul din poză, etc.

1.2.3 Transferul stilului artistic în timp real

În acest capitol voi prezenta articolul lui Justin Johnson, Perceptual Losses for Real-Time Style Transfer and Super-Resolution [9], apărut în anul 2016. Acesta vine cu o soluție la timpul mare necesar creării unei poze artistice, reducându-l semnificativ antrenând o rețea neurală convoluțională care învață să aplique un anumit stil pe o

poză.

1.2.4 Compararea metodelor

În acest capitol voi compara rezultatele obținute de către cele trei metode de mai sus văzând avantajele și dezavantajele pentru fiecare metodă și în ce situații pot fi folosite acestea pentru a obține rezultate satisfăcătoare.

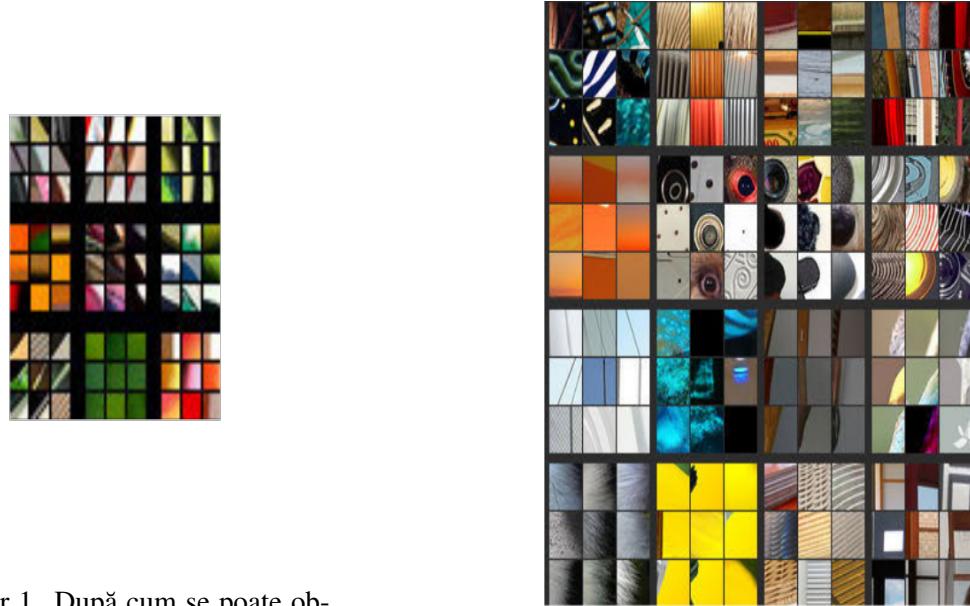
Capitolul 2

Un algoritm neural al stilului artistic

2.1 Introducere

În anii recenți rețelele conoluționale [1.2] s-au dovedit foarte utile în rezolvarea problemelor care conțin imagini. Spre deosebire de rețelele neurale simple, acestea păstrează spațialitatea imaginii și de aceea se comportă mai bine. În anumite sarcini, precum recunoașterea de obiecte dintr-o poza, rețele neurale conoluționale aproape au atins performanța omului în distingerea obiectelor. Pentru o mai bună înțelegere despre cum funcționează aceste rețele voi prezenta un exemplu. [14][3] Alegem un neuron dintr-un anumit layer al rețelei și vedem cum arată porțiunea din imagine pentru care activarea acestui neuron este maximă. În imaginile [2.1a, 2.1b, 2.2, 2.3a, 2.3b], care au fost luate din articolul lui Matthew D. Zeiler, Visualizing and Understanding Convolutional Networks apărut în anul 2013 [3], pentru fiecare layer în parte, au fost aleși diferiți neuroni din layer-ul respectiv, iar pentru fiecare neuron, au fost alese noua imagini care maximizează activarea neuronului și afișată porțiunea pe care neuronul respectiv o vede din poza întreagă.

O explicație pentru care caracteristicile la care se activează neuronii din layerele mai adânci sunt mai complexe este aceea că de-a lungul rețelei neurale, poza este micșorată și astfel neuronii văd o porțiune mai mare din imagine. De asemenea, neuronii din fiecare layer se uită la activările neuronilor din layerul precedent, și astfel combinând caracteristici simple se obțin caracteristici mai complexe.



(a) Layer 1. După cum se poate observa în această poză, neuronii din primul layer tend să se activeze la caracteristici precum linii sau anumite nuanțe de culoare. De exemplu, în colțul din stânga sus neuronul respectiv a avut o activare mare la portiunile din imagine care conțin linii oblice. Pe când, la zona din mijloc de pe ultimul rând, un alt neuron s-a activat când în portiunea pe care acesta o vede erau diferite nuanțe de verde.

(b) Layer 2. Neuronii din acest layer s-au activat la caracteristici mai complexe decât cele la care s-au activat neuronii din primul layer. Se poate observa de exemplu în poza a doua de pe primul rând că neuronul respectiv s-a activat când a văzut un fel de textură alcătuită din linii. În ultima poză de pe primul rând, neuronul respectiv se uită la acele portiuni din imagine care conțin o linie verticală în centru.

Figura 2.1: Layer 1 și layer 2

2.1.1 Generarea conținutului

Din exemplul de mai sus, se observă că atunci când o rețea neurală convoluțională este antrenată să recunoască diferite obiecte dintr-o imagine, aceasta învață să se uite la caracteristici complexe din imagine și astfel este posibil ca având o poză inițializată cu zgomot aleator, o poză oarecare și o funcție de cost bazată pe cele două poze și pe activările neuronilor dintr-un anumit layer, să obținem din poza cu zgomot o poză asemănătoare cu cea pe care o dorim să o recreăm. [4] Astfel, bazându-ne pe această idee putem genera o poză cu conținut asemănător cu cel al unei poze aleatoare alegând un singur layer.

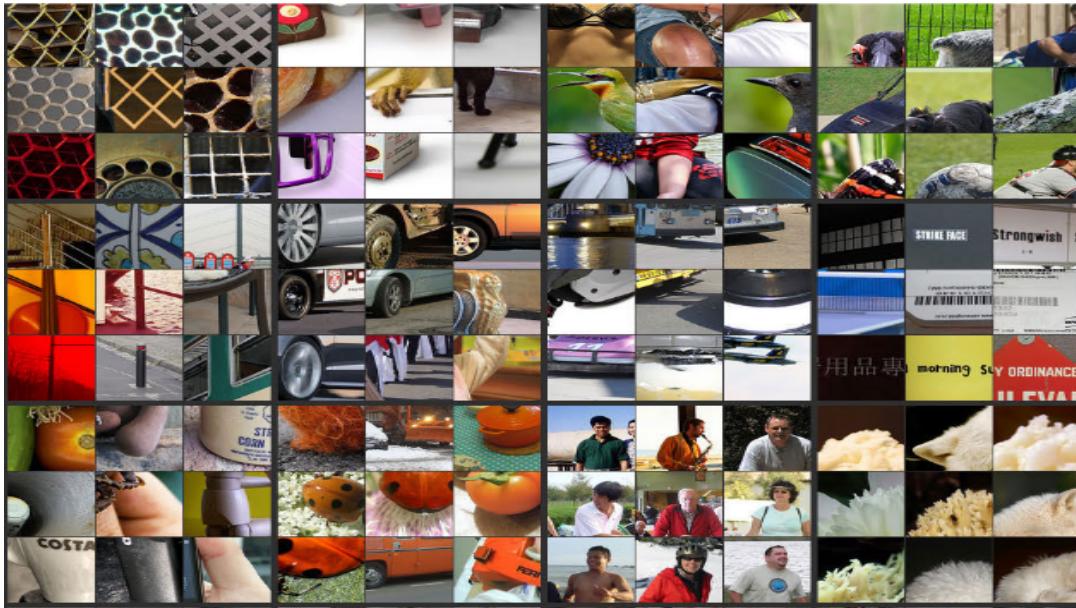
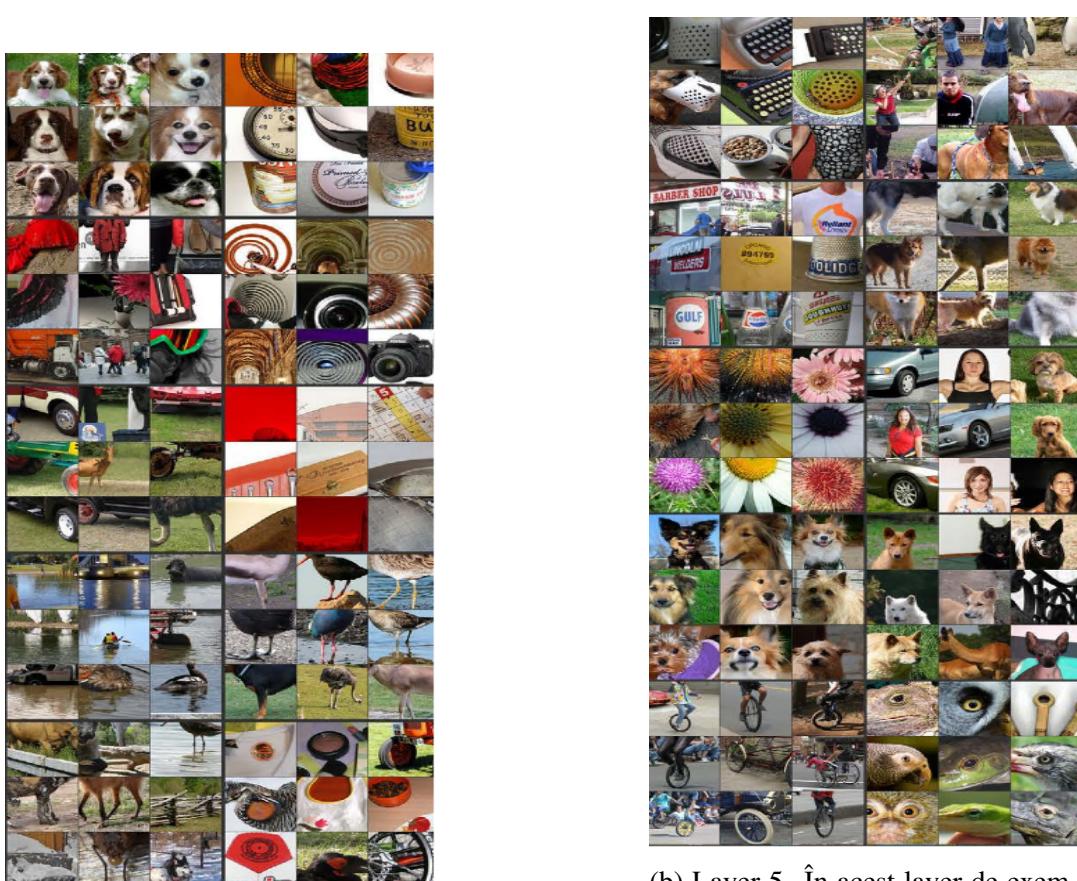


Figura 2.2: Layer 3. În acest layer de exemplu, în poza din colțul stânga sus, neuronul respectiv detectează un anumit fel de textură. În poza a treia de pe ultimul rând se observă că acel neuron se activează la oameni care stau într-o anumită poziție.

În figura [2.4] se poate observa mai bine ce am spus mai sus. Pe prima coloană se află poza cu conținut pe care am dorit să o recreez, pe a doua coloană se află imaginea reconstruită, pornind de la o imagine inițializată cu zgomot, iar pe a treia coloană se află o parte din imaginea respectivă pentru a se vedea mai bine detaliile. Pentru poza de la A am folosit activările din layerul *conv1_2*, pentru poza de la B am ales layerul *conv2_2*, pentru poza de la C am ales layerul *conv3_2*, pentru poza de la D am ales layerul *conv4_2*, iar pentru poza de la E am ales layerul *conv5_2*, peste toate aceste activări am aplicat ReLU. Se observă că dacă se generează poza folosind primele layere, atunci pixelii acesteia au aproape aceleași valori ca și pixelii din poza inițială, pe când dacă se generează poza folosind layere aflate mai departe în rețea se păstrează conținutul, dar pixelii sunt diferenți.

2.1.2 Generarea stilului

După cum am spus, rețelele neurale convoluționale păstrează spațialitatea imaginii și o modeleză cu ajutorul filtrelor aplicate asupra acesteia. Dacă atunci când optimizăm o poză cu zgomot, definim o funcție de cost care ține cont de corelația dintre



(a) Layer 4. În acest layer de exemplu, în poza din colțul stânga sus se observă că neuronul respectiv a învățat să detecteze o anumită rasă de câini. Pe când neuronul din poza din dreapta de pe al patrulea rând se uită la picioare de animale.

(b) Layer 5. În acest layer de exemplu, în poza din stânga de pe ultimul rând se observă că neuronul respectiv a avut activarea maximă când în poză sunt roți subțiri. Pe când, tot pe acest rând, dar la poza din dreapta, neuronul respectiv se uită la ochi de animale.

Figura 2.3: Layer 4 și layer 5

filtrele pozei din unul sau mai multe layere, atunci se obține o textură alcătuită din elemente ale pozei de intrare, precum culoare, linii, diferite forme, etc. [6]

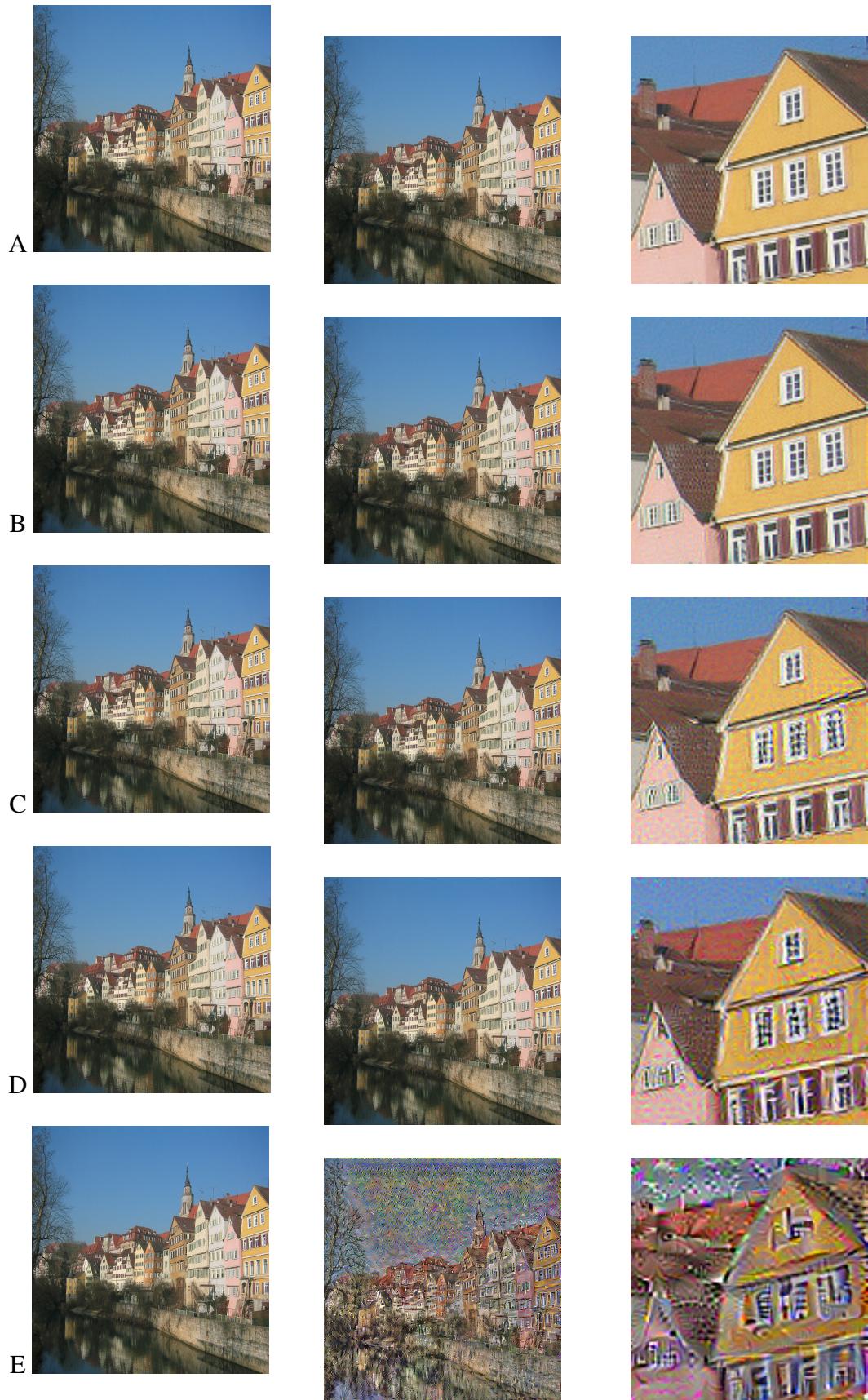


Figura 2.4: Generarea conținutului

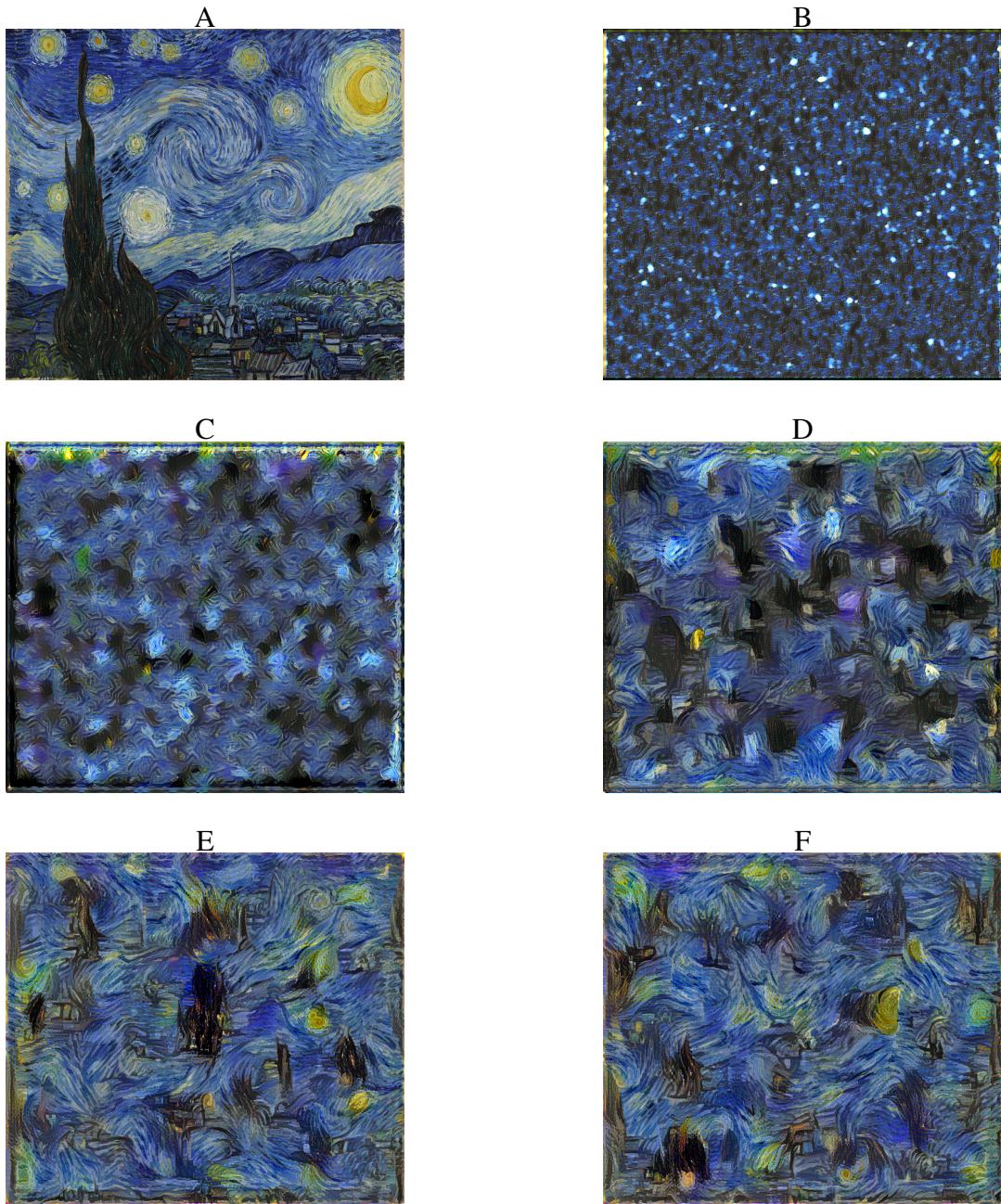


Figura 2.5: Generarea stilului. Figura A este poza pentru care vreau să generez stilul, iar celelalte sunt pozele generate plecând de la o poză inițializată cu zgomot aleator. Pentru poza B am folosit activările din layerul *conv1_1*, pentru poza C am folosit layerele *conv1_1* și *conv2_1*, pentru poza D am folosit layerele *conv1_1*, *conv2_1* și *conv3_1*, pentru poza E am folosit layerele *conv1_1*, *conv2_1*, *conv3_1* și *conv4_1*, iar pentru poza F am folosit layerele *conv1_1*, *conv2_1*, *conv3_1*, *conv4_1* și *conv5_1*. Peste activările acestor layere am aplicat ReLU. Se poate observa că pe măsură ce folosim layere aflate mai departe în rețea, obținem o textură a pozei inițiale, părți din obiectele acesteia apărând în textura generată.

2.2 Metodă

În această secțiune voi defini o funcție de cost, propusă de Leon A. Gatys în articolul sau, A Neural Algorithm of Artistic Style [5], care ne va ajuta în generarea unei imagini care combină conținutul unei poze cu stilul alteia.

Această metodă presupune ca având ca date de intrare două poze, o poză în care se află conținutul pe care ni-l dorim în poza finală și o poză care conține stilul dorit să obținem o nouă poză artistică ce combină conținutul cu stilul pozelor de intrare.

Fie \vec{x} o imagine inițializată cu zgomot aleator, \vec{c} imaginea care conține conținutul pe care ni-l dorim și \vec{s} imaginea care conține stilul, atunci H^l este lungimea pozei la layerul l , W^l este lățimea pozei la layerul l și C^l este numărul de canale/filtre ale pozei la layerul l . F_{ijk}^l este activarea neuronului de la poziția ijk din layerul l . Pentru pozele \vec{x} și \vec{c} considerăm matricele activărilor X_{ijk}^l , respectiv C_{ijk}^l . Atunci, funcția de cost pentru conținut, de care am vorbit în [2.1.1] este definită astfel:

$$\mathcal{L}_{continut}(\vec{c}, \vec{x}) = \frac{1}{2H^l W^l C^l} \sum_{i,j,k} (C_{ijk}^l - X_{ijk}^l)^2 \quad (2.1)$$

La notațiile pentru ecuația [2.1] mai adăugam câteva notații. Fie N^l numărul de filtre pentru o poza din layerul l și $M^l = H^l * W^l$. Atunci, matricea $FV^l \in \mathcal{R}^{N^l \times M^l}$ conține pe fiecare linie vectorul filtrului respectiv din layerul l , unde FV_{ij}^l este activarea neuronului din filtrul i la poziția j în layerul l .

Pentru a putea construi funcția pentru stil, vom folosi matricele gram calculate pentru poza cu zgomot \vec{x} , respectiv poza care conține stilul \vec{s} . O matrice gram este definită astfel, fie $\vec{v}_1, \vec{v}_2 \dots \vec{v}_n$ vectori, atunci matricea gram este egală cu produsul scalar dintre fiecare doi vectori $Gram_{ij} = \langle \vec{v}_i, \vec{v}_j \rangle$ [18]

Pentru pozele \vec{x} și \vec{s} considerăm matricile $XV^l \in \mathcal{R}^{N^l \times M^l}$, respectiv $SV^l \in \mathcal{R}^{N^l \times M^l}$ și matricile gram $XG_{ij}^l = \sum_k XV_{ik}^l XV_{jk}^l$, respectiv $SG_{ij}^l = \sum_k SV_{ik}^l SV_{jk}^l$.

Pentru o mai bună înțelegere a matricelor gram, se pot vedea pozele [2.6, 2.7], luate din cursul lui Andrew Ng [13].

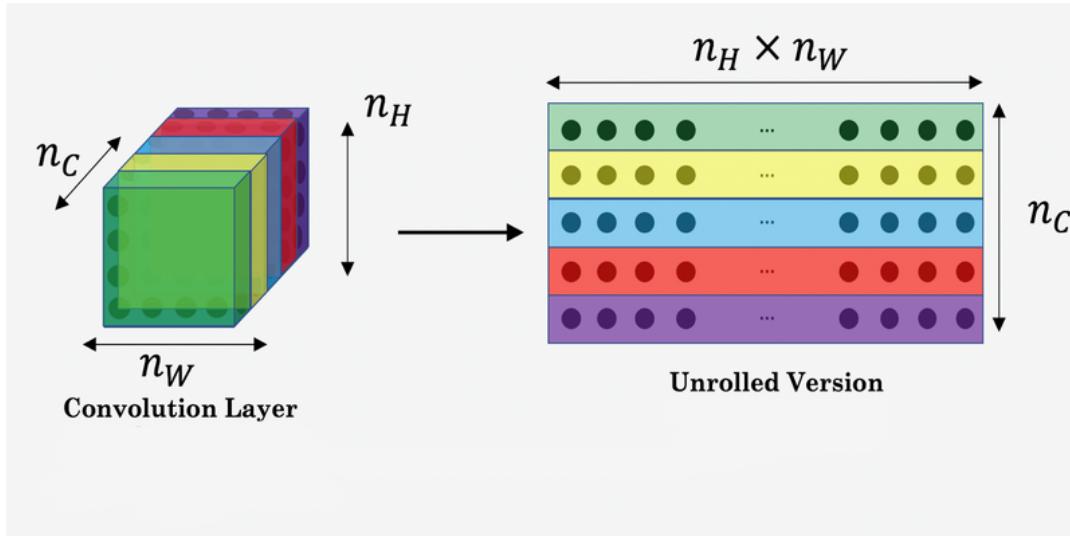


Figura 2.6: Filtrele salvate ca vectori. $N_H = H^l, N_W = W^l, N_C = C^l$. Aici se poate vedea o exemplificare a matricei $FV^l \in \mathcal{R}^{N^l \times M^l}$

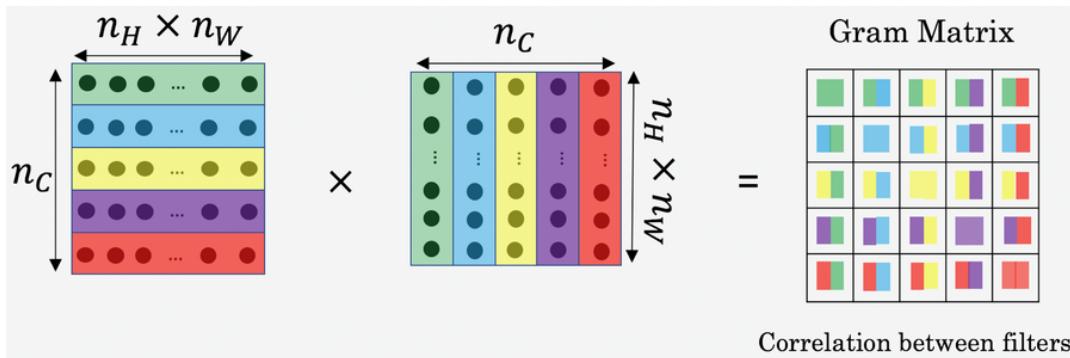


Figura 2.7: Matricea Gram. $N_H = H^l, N_W = W^l, N_C = C^l$. Aici se poate vedea o exemplificare a matricei gram $Gram_{ij} = \langle \vec{v}_i, \vec{v}_j \rangle$

Așadar, matricea gram calculează cât de mult apar 2 filtre împreună într-o poza, fiind produsul dintre activările a doi neuroni care văd aceeași portiune din imagine. Cu cât valoarea din matricea gram este mai mare, cu atât înseamnă că cele două filtre sunt mai corelate.

Având matricele gram calculate, costul dintr-un anumit layer poate fi construit astfel:

$$E_l = \frac{1}{4(H^l)^2(W^l)^2(C^l)^2} \sum_{i,j} (SG_{ij}^l - XG_{ij}^l)^2 \quad (2.2)$$

Atunci, funcția de cost pentru stil, de care am vorbit în [2.1.2] este definită astfel:

$$\mathcal{L}_{stil}(\vec{s}, \vec{x}) = \sum_{l=1}^L w_l E_l \quad (2.3)$$

unde L este numărul de layere luate în considerare pentru generarea stilului, iar w_l este un parametru care controlează importanța layerului l .

Pentru ca pozele generate să fie cât mai calitative, să nu prezinte zgomot, o să mai definesc o funcție de cost care calculează cât de mult zgomot este în imagine [28]. Aceasta are următoarea formulă:

$$\mathcal{L}_{zgomot}(\vec{x}) = \sum_{i,j} |x_{i+1,j} - x_{i,j}| + |x_{i,j+1} - x_{i,j}| \quad (2.4)$$

Având definite costurile pentru zgomot, conținut și stil, costul total se definește astfel:

$$\mathcal{L}_{total}(\vec{c}, \vec{s}, \vec{x}) = \alpha \mathcal{L}_{continut}(\vec{c}, \vec{x}) + \beta \mathcal{L}_{stil}(\vec{s}, \vec{x}) + \gamma \mathcal{L}_{zgomot}(\vec{x}) \quad (2.5)$$

unde parametrii α , β și γ controlează compromisul dintre conținut, stil și respectiv zgomot.

Pe baza funcției totale de cost vom optimiza pixelii unei poze inițializate cu zgomot aleator cu ajutorul algoritmului gradientului de coborâre astfel încât costul total să fie cât mai mic.

2.3 Detalii de implementare

Pentru a implementa metoda descrisă mai sus am folosit Python [2.3.1] și TensorFlow [2.3.2]. După cum am observat, rețelele convoluționale învață anumite reprezentări ale obiectelor dintr-o poză atunci când sunt antrenate pentru diferite sarcini. Astfel, pentru a putea calcula costurile definite în subcapitolul precedent ne vom folosi de activările din layerele unei rețele convoluționale care a fost antrenată să recunoască obiectele dintr-o poză. Antrenarea a avut loc pe baza a aproximativ 1,2 milioane de poze, obiectele din poze fiind clasificate în 1000 de categorii. În figura [2.8] se pot observa câteva poze din setul de antrenare. Această retea a fost folo-

sită la competiția ImageNet din 2014 [2.3.3], unde a obținut locul 2 la problema de clasificare, obținând o eroare egală cu 0.07 și se numește VGG19 [2.9], iar parametrii pe care i-a învățat rețeaua în timpul antrenării sunt făcuți publici de către autorii acesteia pe site-ul lor. [12] Din această rețea am folosit doar layerele convolutionale, fără cele conectate complet, iar în loc de *maxpool* am folosit *avgpool*, precum Leon A. Gatys, deoarece acesta a observat că această modificare produce rezultate mai satisfăcătoare.



Figura 2.8: Imagini din setul de antrenare ImageNet

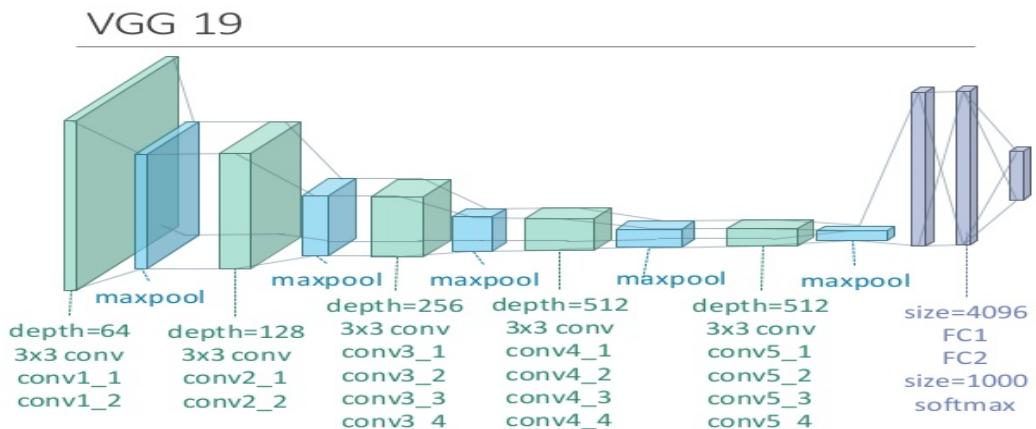


Figura 2.9: Arhitectura rețelei neurale convolutionale VGG19 [17]

Algoritmul primește ca date de intrare două poze în spațiul de culori RGB, una cu conținutul dorit și alta cu stilul dorit. Având rețeaua de mai sus și cele două poze, am inițializat o poză de aceeași dimensiune cu cea a pozei de conținut folosind inițializarea Xavier [2.3.4], astfel, poza inițială prezintă zgomot aleator, pixelii acesteia având o distribuție uniformă, cu valori între $[-\text{limita}, \text{limita}]$, unde

$limita = \sqrt{\frac{6}{H*W*C}}$, iar H este lungimea pozei, W este lățimea pozei și C este numărul de canale ale pozei.

Apoi am normalizat cele trei poze pentru a avea aceeași distribuție ca cea pe care a fost antrenată rețeaua, și anume, pentru fiecare canal al pozei am scăzut valorile [123.68, 116.779, 103.939], valori care reprezintă media canalelor RGB pentru pozele folosite la antrenare de rețeaua VGG19 și le-am trecut pe rând prin rețea și pentru fiecare am salvat ieșirile din layerul *conv4_2* pentru conținut și ieșirile din layerele *conv1_1*, *conv2_1*, *conv3_1*, *conv4_1*, *conv5_1* pentru stil. Acestea sunt numele layerelor din rețeaua VGG19. Peste aceste valori pe care le-am salvat am aplicat ReLU [2.3.5] și am calculat costul cu ajutorul funcțiilor de cost definite, iar cu algoritmul gradientului de coborâre [2.3.6] și a metodei backpropagation [sb:backpropagation] de calculare a gradienților am optimizat pixelii pozei inițiale astfel încât costul total să fie cât mai mic.

În imaginea [2.10] se poate observa procesul pe care l-am descris mai sus.

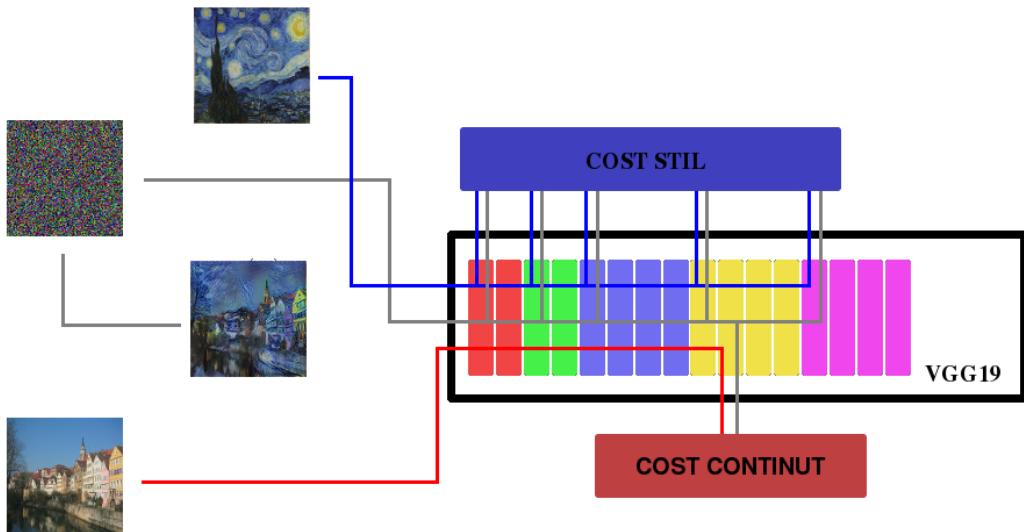


Figura 2.10: Workflow-ul algoritmului

Pentru toate rezultatele din acest capitol am repetat proceful de mai sus timp de 2000 de iterații cu rata de învățare [2.3.9] egală cu 2 pentru a ajunge la rezultate satisfăcătoare. De asemenea, am ales $\alpha = 100$, $\beta = 1$, $\gamma = 0.03$ și $w_l = 0.2$ pentru toate layerele. Acești parametri i-am ales după mai multe încercări, în care am analizat pozele generate, după perceptia mea, deoarece pentru acest timp de problemă nu există

o metrică pentru a spune cât de bune sunt pozele generate. Pozele au fost redimensionate, păstrând raportul dintre lungime și lățime, la $\max(\text{lungime}, \text{lățime}) = 700$ de pixeli.

2.3.1 Python

Python [24] este un limbaj de programare, creat de Guido van Rossum în anul 1991. Este suportat de mai multe sisteme de operare fiind un limbaj de programare ușor de folosit și care oferă posibilitatea de a scrie programe complexe, într-o maniera elegantă. Python este folosit de majoritatea celor care doresc să creeze programe ce folosesc învățarea automată.

2.3.2 TensorFlow

TensorFlow [27] a fost creat de către cercetătorii din echipa Google Brain. Această librărie, prin funcțiile pe care le deține, ca de exemplu calcularea gradienților sau crearea de layere într-o rețea, oferă posibilitatea de a construi ușor programe ce folosesc învățarea automată. Tensorflow este folosit activ de cei de la Google și de asemenea este și open-source, fapt ce constituie un plus, deoarece datorită numărului mare de contribuitori este îmbunătățit continuu.

2.3.3 ImageNet

ImageNet [21] este o bază de date ce conține milioane de poze adnotate de către oameni, imagini destinate pentru diferite probleme legate de învățarea automată. Începând cu anul 2010, în fiecare an se organizează un concurs numit ImageNet Large Scale Visual Recognition Challenge (ILSVRC) în care diferiți cercetători încearcă să obțină rezultate cât mai bune la probleme de genul: clasificarea obiectelor dintr-o poză sau localizarea obiectelor.

2.3.4 Inițializarea Xavier

[23] O problemă cu care se confruntă rețelele convolutionale foarte adânci este modul în care sunt inițializate filtrele. Dacă acestea sunt inițializate cu valori prea mici, atunci de-a lungul rețelei, este posibil să devină foarte aproape de zero, astfel rețeaua nemaiputând învăța. Pe de altă parte, dacă sunt prea mari, atunci acestea tind să devină prea mari de-a lungul rețelei ca să mai fie folositoare. În articolul său [2], Xavier Glorot propune o formulă de inițializare a filtrelor care s-a dovedit a fi fiabilă în practică. Această inițializare are rolul de a ține valorile dintr-un anumit filtru într-un anumit interval, valorile având o distribuție uniformă sau normală. Formula pentru variație în cazul când dorim ca valorile să aibă o distribuție uniformă este:

$$\text{variație} = \sqrt{\frac{6}{\text{intrari} + \text{iesiri}}} \quad (2.6)$$

Unde *intrari* reprezintă câte valori intră în layerul respectiv, iar *iesiri* câte valori ieș din layerul respectiv.

2.3.5 ReLU

Rectified linear unit (ReLU) [25] este o funcție de activare neliniară. Aceasta se aplică pe valorile de ieșire ale layerelor. Este inspirată din biologie și din cum se crede că funcționează transmiterea de informație în creier. Atunci când un neuron are o valoare negativă, după aplicarea funcției ReLU acesta are valoarea 0, însemnând că este inactiv, iar când valoarea unui neuron este pozitivă, activarea acestuia este fix valoarea lui.

$$f(x) = \max(0, x) \quad (2.7)$$

2.3.6 Coborâre pe gradient

Coborârea pe gradient [23] este un algoritm de minimizare a unei funcții care depinde de mai multe variabile. Să presupunem că funcția de cost este următoarea

$\mathcal{L}(x_1, x_2 \dots x_n)$. Atunci, pentru a minimiza această funcție vom aplica următorul algoritm pentru modificarea variabilelor de care depinde.

$$x_i = x_i - \alpha * \frac{\partial}{\partial x_i} \mathcal{L} \quad (2.8)$$

Unde $\frac{\partial}{\partial x_i} \mathcal{L}$ este derivata parțială a funcției \mathcal{L} în raport cu variabila x_i . Derivata reprezintă panta tangentei la graficul funcției, și indică direcția funcției, dacă aceasta crește sau descrește. Iar α este rata de învățare.

2.3.7 Backpropagation

[23] Așa cum am văzut la algoritmul coborârii pe gradient, pentru a minimiza o funcție de cost $\mathcal{L}(x_1, x_2 \dots x_n)$, este nevoie să calculăm derivatele parțiale $\frac{\partial}{\partial x_i} \mathcal{L}$. Plecând de la rezultatul funcției de cost, cu ajutorul acestei metode se propagă înapoi în rețea eroarea până la variabilele de care depinde această funcție care sunt optimizate după formula [2.8].

2.3.8 Optimizatorul Adam

Optimizatorul Adam [23] este o variantă îmbunătățită a algoritmului coborârii pe gradient făcând ca învățarea să meargă mai repede. O problemă a coborârii pe gradient este aceea că face mulți pași până a ajunge să conveargă către minim. Optimizatorul Adam se folosește de valorile anterioare ale gradienților, și intuitiv, vede un trend pe care îl urmează gradienții, putând la un anumit moment să facă un pas mai mare decât ar face algoritmul coborârii pe gradient. Formula acestuia la pasul t este următoarea:

$$V_t = \beta_1 * V_{t-1} + (1 - \beta_1) * \frac{\partial}{\partial x_i} \mathcal{L} \quad (2.9)$$

$$S_t = \beta_2 * S_{t-1} + (1 - \beta_2) * \left(\frac{\partial}{\partial x_i} \mathcal{L} \right)^2 \quad (2.10)$$

$$x_i = x_i - \alpha * \frac{V_t}{\sqrt{S_t} + \epsilon} \quad (2.11)$$

Unde, de obicei, $\beta_1 = 0.9$, asta însemnând aproximativ că V_t calculează media ultimilor 10 gradienți, $\beta_2 = 0.999$, $\epsilon = 1e-08$, acesta este folosit în cazul în care

$S_t = 0$ și să se evite împărțirea la 0, iar α este rata de învățare.

2.3.9 Rata de învățare

Rata de învățare [23] este folosită în formula algoritmilor de optimizare. Aceasta controlează cat de mult să fie modificată o variabilă pentru a optimiza funcția ţintă. O valoare mica a acestei rate, înseamnă că variabila respectivă se va modifica foarte puțin, fiind necesar un timp mai mare pentru ca funcția să conveargă către minim. Pe de altă parte, dacă rata de învățare este foarte mare, atunci este posibil ca funcția să nu mai ajungă să conveargă, deoarece variabila este modificată foarte mult.

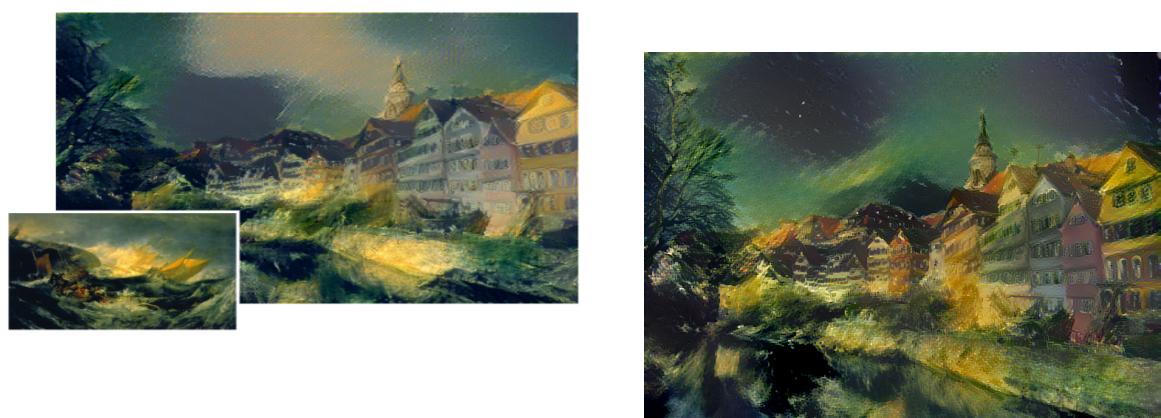
2.4 Rezultate și comparații



(a) Neckarfront, Tübingen, Germania, autor Andreas Praefcke.



(b) The Scream, Edvard Munch, 1893.



(c) The Shipwreck of the Minotaur, J.M.W. Turner, 1805

Figura 2.11: Pe coloana din stânga sunt rezultatele luate din articolul lui Leon A. Gatys, iar pe coloana din dreapta sunt rezultatele obținute de mine. Pozele diferă deoarece conțează dimensiunea imaginilor între care s-a realizat transferul de stil sau modul în care a fost implementat algoritmul. Fiindcă nu există un mod obiectiv pentru a decide care imagini sunt mai artistice, ambele variante sunt valabile.



(a) The Starry Night, Vincent van Gogh, 1889



(b) Femme nue assise, Pablo Picasso, 1910



(c) Composition VII, Wassily Kandinsky, 1913



(d) The Muse, Pablo Picasso, 1935



(e) The Great Wave off Kanagawa, Hokusai, 1829-1832



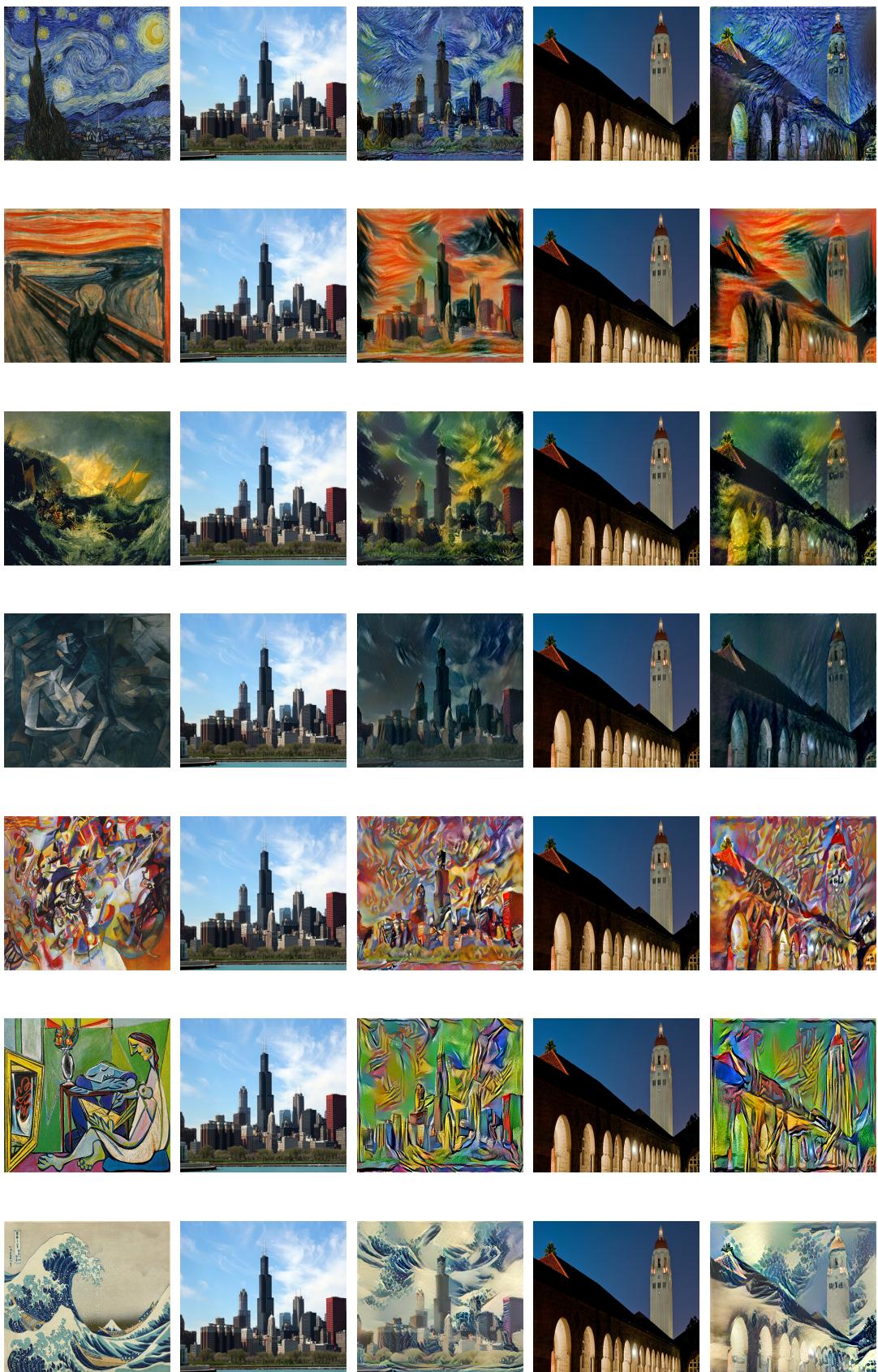


Figura 2.13



Figura 2.14: În această figură se observă impactul pe care îl are poza cu stil la diferite dimensiuni. Poza (b) a avut dimensiunea $\max(\text{lungime}, \text{latime}) = 700$ de pixeli. Pentru generarea pozei (c), poza (a) a avut dimensiunea $\max(\text{lungime}, \text{latime}) = 700$ de pixeli. Pentru generarea pozei (d), poza (a) a avut dimensiunea $\max(\text{lungime}, \text{latime}) = 400$ de pixeli. Pentru generarea pozei (e), poza (a) a avut dimensiunea $\max(\text{lungime}, \text{latime}) = 1400$ de pixeli.

Pozele cu conținut din figura [2.13] au fost luate de pe github-ul lui Justin Johnson [15], respectiv [19].

În poza [2.15] se poate observa compromisul dintre conținut și stil. Pentru stil, pentru pozele de pe linia A am folosit activările din layerul *conv1_1*, pentru pozele de pe linia B am folosit layerele *conv1_1* și *conv2_1*, pentru pozele de pe linia C am folosit layerele *conv1_1*, *conv2_1* și *conv3_1*, pentru pozele de pe linia D am folosit layerele *conv1_1*, *conv2_1*, *conv3_1* și *conv4_1*, iar pentru pozele de pe linia E am folosit layerele *conv1_1*, *conv2_1*, *conv3_1*, *conv4_1* și *conv5_1*. Iar pentru conținut, pentru toate pozele am folosit activările din layerul *conv4_2*.

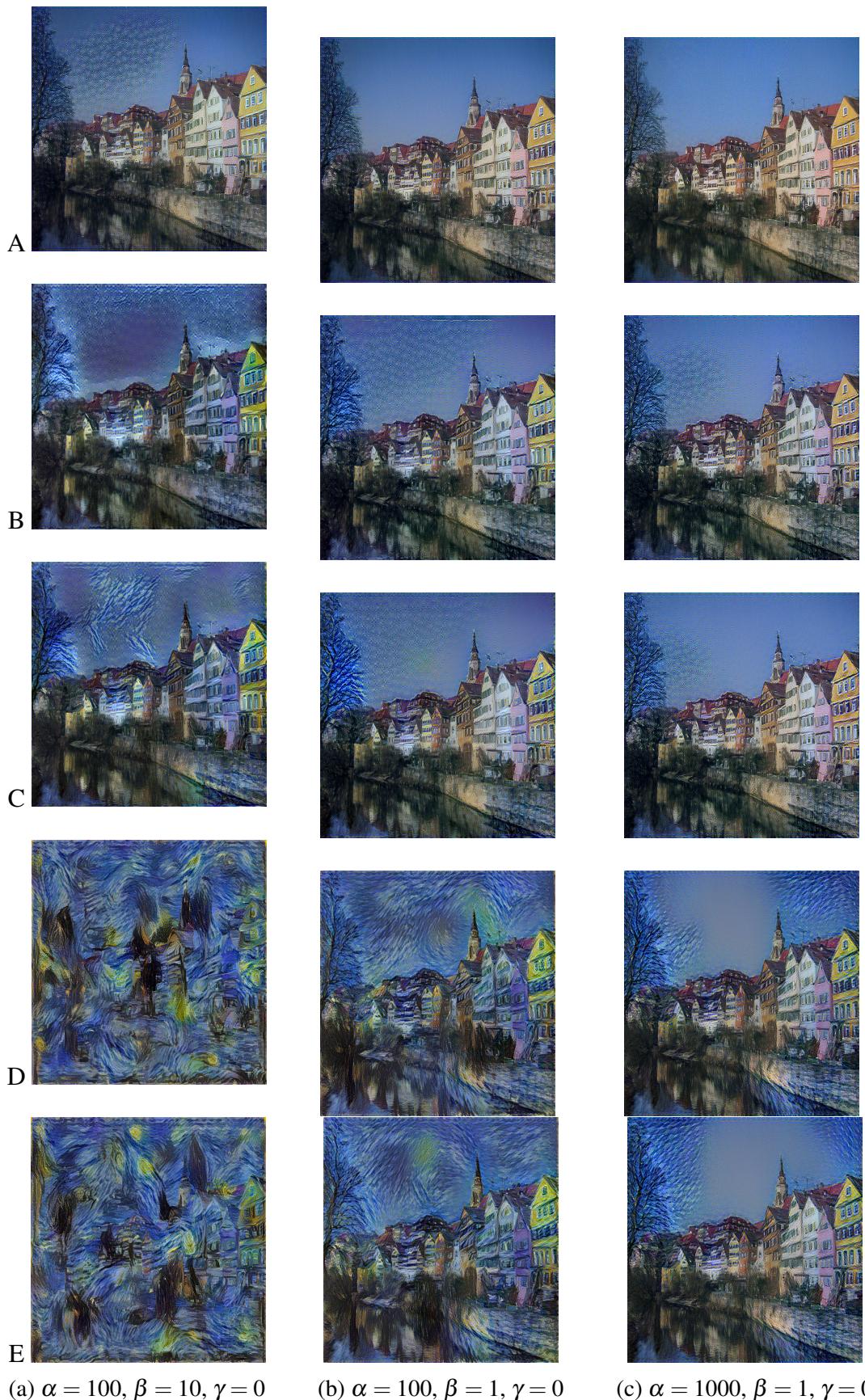


Figura 2.15

Capitolul 3

Transferul stilului artistic fotografic

3.1 Introducere

După cum am văzut în capitolul Un algoritm neural al stilului artistic [2] este posibil ca stilul și conținutul unei poze să fie separabile, și astfel combinând două poze oarecare să se obțină o poză nouă care combină conținutul uneia dintre poze cu stilul celeilalte.

Deși această metodă oferă rezultate satisfăcătoare, se pune problema că poza obținută să fie cât mai fotografică. Chiar dacă folosim metoda lui Leon A. Gatys pe două poze care sunt fotografice, de exemplu două poze făcute unor blocuri dintr-un oraș, atunci tot apar nereguli în poza generată, lucru pe care nu îl-am dori, am dori că poza generată să fie tot una fotografică. Am vrea să se păstreze conținutul semantic în timpul combinării conținutului cu stilul. De exemplu dacă în poza cu stil se vede cerul, atunci este posibil că în poza generată, părți ale cerului să apară pe unele obiecte din poza cu conținut deoarece, formula [2.3] nu ține cont de unde apare respectivul stil în imagine, ci doar că acesta trebuie să fie prezent în locuri aleatoare. Pentru a rezolva această problemă este nevoie de o constrângere pentru formula [2.3] pentru a forța algoritmul să respecte conținutul semantic. În articolul său [11] Fujun Luan propune introducerea de măști pentru poza de conținut și poza de stil. Aceste măști etichetează obiectele cu același înțeles semantic din cele două poze, atribuindu-le o etichetă.

În cele ce urmează o să dovedesc că această îmbunătățire adusă de Fujun Luan algoritmului lui Leon A. Gatys este fiabilă în generarea de poze fotografice venind cu mai multe exemple. De asemenea, o altă aplicabilitate pe care o aduc măștile pentru

pozele de conținut și stil este aceea că de exemplu, putem modifica culoarea unor obiecte din aceeași poză între ele.

3.1.1 Etichetări semantice

Pentru cazul descris mai sus, cel în care ne dorim să păstrăm conținutul semantic atunci când combinăm conținutul cu stilul a două poze, voi prezenta soluția lui Fujun Luan, aceea de a introduce măști cu ajutorul cărora putem eticheta obiectele cu același înțeles semantic. Pe lângă faptul că aceste măști impun algoritmului să aducă modificări doar obiectelor asemănătoare, mai oferă și posibilitatea utilizatorului de a transfera stilul doar acelor obiecte de care este interesat și astfel obținând noi poze aparte.

3.2 Metodă

În această secțiune voi defini o funcție de cost, asemănătoare cu cea din capitolul [2] [2.5], propusă de Fujun Luan în articolul său.

Pentru notații, le vom folosi tot pe cele definite anterior, la care o să mai adăugam unele noi.

Funcția de cost pentru conținut este aceeași cu cea definită la [2.1].

Reamintesc că funcția de cost pentru stil este definită la [2.3]. Pentru a rezolva problema enunțată în [3.1.1] vom modifica această funcție de cost pentru a constrânge algoritmul să aplique un anumit stil doar pe porțiunile cu același înțeles semantic din poza de conținut și poza de stil cu ajutorul măștilor. Fie $M_{l,c}$ masca pentru o poză din layerul l și c canalul măștii respective pentru acel layer. Pentru pozele \vec{x} și \vec{s} considerăm matricele $XM_{l,c}$, respectiv $SM_{l,c}$. Atunci înmulțind matricele $XV^l \in \mathcal{R}^{N^l \times M^l}$ și $SV^l \in \mathcal{R}^{N^l \times M^l}$ cu matricele măștilor obținem $XV^{l,c} = XV^l XM_{l,c}$, respectiv $SV^{l,c} = SV^l SM_{l,c}$. Astfel, matricele gram devin $XG_{ij}^{l,c} = \sum_k XV_{ik}^{l,c} XV_{jk}^{l,c}$,

respectiv $SG_{ij}^{l,c} = \sum_k SV_{ik}^{l,c} SV_{jk}^{l,c}$ și costul dintr-un anumit layer devine:

$$E_l = \sum_{c=1}^C \frac{1}{4(H^l)^2(W^l)^2(C^l)^2} \sum_{i,j} (SG_{ij}^{l,c} - XG_{ij}^{l,c})^2 \quad (3.1)$$

unde C este numărul total de canale din mască.

Atunci, funcția de cost pentru stil este definită astfel:

$$\mathcal{L}_{stil+}(\vec{s}, \vec{x}) = \sum_{l=1}^L w_l E_l \quad (3.2)$$

Astfel, având definite toate costurile intermediare, putem defini costul total:

$$\mathcal{L}_{total}(\vec{c}, \vec{s}, \vec{x}) = \alpha \mathcal{L}_{continut}(\vec{c}, \vec{x}) + \beta \mathcal{L}_{stil}(\vec{s}, \vec{x}) + \gamma \mathcal{L}_{zgomot}(\vec{x}) \quad (3.3)$$

unde parametrii α , β și γ controlează compromisul dintre conținut, stil și zgomot, iar $\mathcal{L}_{zgomot}(\vec{x})$ este funcția de cost definită în [2.4].

3.3 Detalii de implementare

Pentru a implementa metoda de mai sus am folosit același procedeu descris în capitolul Un algoritm neural al stilului artistic [2] cu o mică adăugare. Și anume, după ce am trecut imaginile prin rețeaua VGG19, am adăugat măștile peste activările din layerele respective și am fost atent ca atunci când am aplicat masca pe activările din layerele pentru stil, aceasta să fie de aceeași dimensiune cu matricea activărilor, deoarece imaginea este micșorată de-a lungul rețelei.

Și pentru această metodă am folosit aceeași parametri ca cei menționați în capitolul precedent.

Măștile pentru poze le-am creat manual în programul de editare de imagini numit GIMP. Această abordare de creare manuală a măștilor oferă o arie mai largă de generare de poze artistice, cel care folosește acest algoritm, fiind liber să aleagă între ce obiecte din poză să fie făcut transferul de stil. Culorile prezente în măștile pe care le-am folosit pentru acest algoritm sunt alb, negru, roșu, verde sau albastru, numărul

acestora fiind suficient pentru păstrarea înțelesului semantic în pozele pe care le-am întâlnit. De asemenea, se mai pot adăuga oricâte culori dacă pozele sunt complexe și există multe obiecte diferite. În cazul în care nu se dorește transferul stilului pentru un anumit obiect, atunci în mască, pentru acel obiect trebuie să fie o culoare care nu este prezentă în cele definite, eu am folosit culoarea gri pentru acest caz. Un lucru la care a trebuit să mai fiu atent când am creat măștile este acela să nu am obiecte marcate în masca pentru poza de conținut, dar care nu sunt prezente în masca pentru poza de stil sau invers, deoarece algoritmul nu ar mai funcționa cum trebuie, neavând între ce obiecte să facă transferul de stil.

3.4 Rezultate și comparații

Toate pozele cu stil și cu conținut sunt luate din articolul lui Fujun Luan, Deep Photo Style Transfer [11].



(a) Poza cu conținut

(b) Poza cu stil



(c) Poză luată din articolul lui Fujun Luan

(d) Poză generată de mine

Figura 3.1: Cele două poze sunt diferite, deoarece în poza din stânga, autorul mai aplică unele optimizări în plus. De asemenea, mai intră în calcul și dimensiunea pozelor cu conținut și stil care au fost folosite la generarea celei de-a treia poze.

În continuare, voi mai prezenta și alte rezultate pentru a exemplifica posibilitățile pe care le oferă această metoda, cum ar fi schimbarea anotimpului dintr-o poză sau modificarea doar a anumitor obiecte din poză folosind măștile.



Figura 3.2

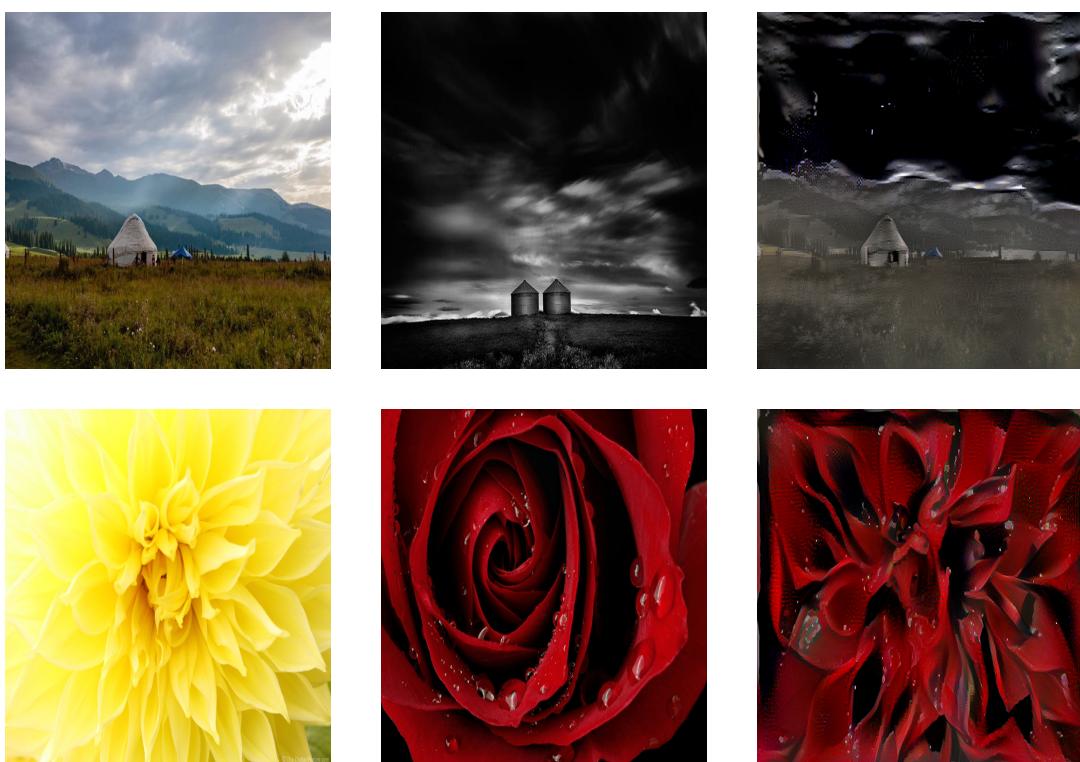


Figura 3.3: Poze mai puțin reușite din cauza complexității și din cauza nepotrivirii întocmai a conținutului cu stilul.

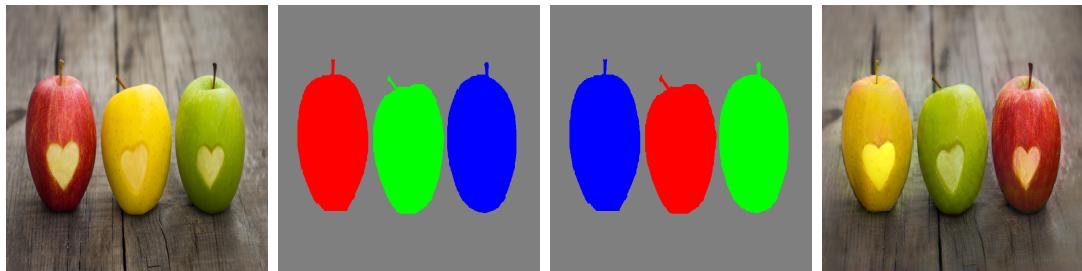


Figura 3.4: În această figură se poate vedea avantajul măștii. Poza din stânga este tratată și ca poză de conținut și ca poză de stil. Poza a doua reprezintă masca pentru poza care este generată, iar a treia poză reprezintă masca pentru poza cu stil. Ultima poză este poza generată de către algoritm.



Figura 3.5: În această figură, pentru generarea pozei a treia am pornit de la o poză inițializată cu zgomot, iar pentru generarea ultimei poze, am pornit de la o poză inițializată cu poza de conținut peste care am adăugat zgomot aleator. Se poate observa că folosind ultima metodă algoritmul învăță mai ușor reprezentarea conținutului, iar stilul are un impact mai scăzut.

(a) $\gamma = 0.001$ (b) $\gamma = 0.01$ (c) $\gamma = 0.1$ (d) $\gamma = 1$

Figura 3.6: În această figură se poate vedea impactul pe care îl are γ , parametrul care controlează importanța funcției de cost pentru zgomot.

Capitolul 4

Transferul stilului artistic în timp real

4.1 Introducere

Deși metodele prezentate în capituloare [2] și [3] demonstrează faptul că se pot obține poze artistice combinând conținutul și stilul a două poze, timpul necesar pentru generarea unei astfel de poze este unul destul de mare. În acest capitol voi prezenta o soluție la problema de mai sus descrisă de Justin Johnson în articolul său [9]. După cum știm, se poate antrena o rețea neurală convoluțională pentru a rezolva diferite probleme care conțin imagini, precum recunoașterea de obiecte sau segmentarea obiectelor dintr-o poză. Plecând de la această idee și folosind metoda lui Leon A. Gatys de separare a stilului și conținutului, Justin Johnson propune antrenarea unei rețele neurale convoluționale pentru a învăța să aplique un anumit stil pe o poză.

În cele ce urmează voi arăta că soluția lui Justin Johnson este una fiabilă, obținând rezultate asemănătoare cu cele date de algoritmul lui Leon A. Gatys. Pe lângă faptul acesta, timpul de generare a unei poze artistice este redus semnificativ, deoarece este nevoie doar de o parcurgere a pozei initiale prin rețea, singura parte care necesită un timp mai mare fiind aceea de antrenare a rețelei.

4.2 Metodă

Pentru a putea antrena o rețea neurală convoluțională care să învețe să aplique un anumit stil pe poze este nevoie să ne definim o funcție de cost. Ne vom folosi de funcțiile de cost prezentate în capitolul Un algoritm neural al stilului artistic [2]. Și anume, funcția de cost pentru conținut este cea definită la [2.1], funcția de cost pentru stil este cea definită la [2.3], funcția de zgomot pentru poza care urmează să

fie generată este cea definită la [2.4] și funcția de cost finală este cea de la [2.5].

4.3 Detalii de implementare

Deoarece această metodă este diferită de cele două prezentate anterior, procedeul pe care l-am urmat pentru a implementa-o este diferit. În primul rând este necesar să antrenez o rețea neurală convezională care să învețe să aplice un anumit stil. Așadar am folosit rețeaua propusă de Justin Johnson în articolul său, pe care a numit-o Transform Net. Arhitectura acestei rețele se poate observa în imaginea [4.2a]. În această imagine, $C \times H \times W$ conv este interpretat astfel: C este numărul de filtre din layer, H este lungimea filtrului, iar W este lățimea filtrului. $Stride n$ este numărul de pixeli pe care să îl sără filtrul atunci când este plimbăt pe imagine. De asemenea se observă că apar și 5 blocuri reziduale [4.2b]. După fiecare layer non-rezidual este aplicată o normalizare a batch-ului [4.3.2] urmată de funcția de activare ReLU, mai puțin pentru ultimul layer, unde în loc de ReLU este folosită funcția de activare tanh (tangenta hiperbolică) [4.3.3] pentru că ne dorim ca pixelii pozei întoarse de rețea să fie între $[-1, 1]$. Arhitectura acestei rețele este una interesantă deoarece inițial, rețeaua reduce dimensiunea pozei de intrare până la un anumit punct pentru a putea învăța caracteristici mai complexe, apoi la această dimensiune sunt aplicate o serie de blocuri reziduale, iar mai apoi imaginea este mărită cu ajutorul layerelor ce poartă numele de layere convezionale transpusă ajungând la dimensiunea inițială.

La fel ca și la metoda lui Leon A. Gatys, algoritmul primește ca date de intrare două poze, una cu conținutul dorit și una cu stilul. La acest algoritm nu mai este nevoie să fie inițializată cu zgomot aleator o a treia poză deoarece este folosită rețeaua să învețe aplicarea stilului respectiv. Așadar, poza de conținut am normalizat-o și am trecut-o prin rețeaua Transform Net obținând o imagine nouă. Apoi am trecut cele trei poze, poza obținută de la Transform Net, poza cu conținutul și poza cu stilul prin rețeaua VGG19, și la fel ca la metoda lui Leon A. Gatys, am salvat activările din layerul *conv4_2* pentru conținut și activările din layerele *conv1_1*, *conv2_1*, *conv3_1*, *conv4_1*, *conv5_1* pentru stil pentru a putea calcula funcțiile de cost. Pe baza acestor funcții de cost, cu ajutorul algoritmului de coborâre pe gradient și a metodei back-

propagation am optimizat valorile filtrelor pentru a obține un cost total cât mai mic.

În imaginea 4.1 se poate observa procesul pe care l-am descris mai sus.

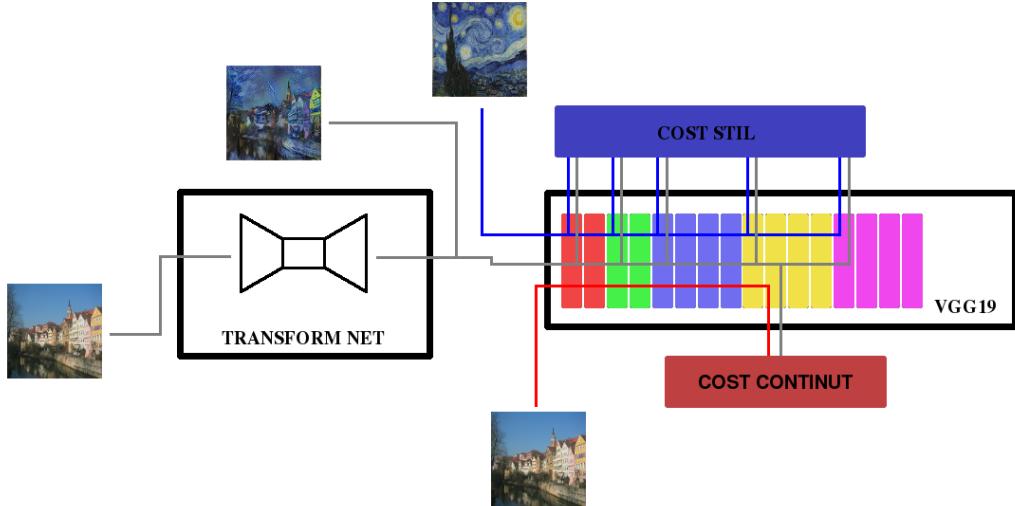


Figura 4.1: Workflow-ul algoritmului

Pentru a învăța aplicarea unui anumit stil, rețeaua Transform Net are nevoie de multe poze cu conținut. De aceea, pentru antrenare, am folosit setul de date Microsoft COCO [4.3.6]. Setul de antrenare conține aproximativ 80000 de poze. Pentru rezultatele din această lucrare am antrenat rețeaua timp de 10000 de iterării, fiecare iterare conținând un batch de 4 poze luate aleator din setul de antrenare pe care le-am redimensionat la dimensiunea de 256×256 pixeli. De asemenea și poza cu conținut și poza cu stil le-am redimensionat la 256×256 de pixeli. Antrenarea a durat aproximativ 7 ore pe Google Cloud, pe o placă video NVIDIA Tesla K80.

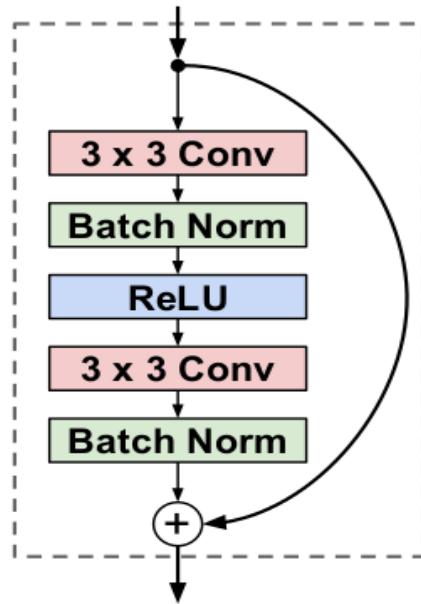
Parametrii folosiți în timpul antrenării au fost $\alpha = 1000$, $\beta = 1$ și rata de învățare egală cu 0.001.

4.3.1 Normalizarea datelor

[23] Pentru antrenarea unei rețele neurale trebuie ca datele de intrare să fie normalize. Această practică are mai multe avantaje. Un avantaj este acela că, dacă de exemplu datele de intrare depind de două caracteristici, prima caracteristică fiind în intervalul $[0, 1]$, iar a două în intervalul $[0, 1000]$ și având o funcție pe care dorim

Layer	Activation size
Input	$3 \times 256 \times 256$
$32 \times 9 \times 9$ conv, stride 1	$32 \times 256 \times 256$
$64 \times 3 \times 3$ conv, stride 2	$64 \times 128 \times 128$
$128 \times 3 \times 3$ conv, stride 2	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 128 \times 128$
$32 \times 3 \times 3$ conv, stride 1/2	$32 \times 256 \times 256$
$3 \times 9 \times 9$ conv, stride 1	$3 \times 256 \times 256$

(a) Arhitectura rețelei [10]



(b) Bloc rezidual [10]

să o minimizăm și care depinde de cele două variabile, atunci dacă afișăm conturul funcției, acesta va avea formă de oval, iar ca algoritmul folosit pentru a minimiza această funcție să conveargă către minim va avea nevoie de un timp mai mare. De aceea, dacă datele sunt normalizeze, toate caracteristicile aflându-se în același interval, atunci antrenarea se va face mult mai repede, deoarece conturul funcției va avea forma mai rotundă. Un alt avantaj este acela că dacă toate caracteristicile se află în același interval, atunci acestea vor contribui în mod egal la costul unei funcții, altfel dacă luăm tot exemplul de mai sus, atunci caracteristica ce se află în intervalul $[0, 1000]$ va avea un impact mai mare asupra valorii funcției și rețeaua va tinde să îi acorde o importanță mai mare.

4.3.2 Normalizarea batch-ului

[23] După cum am spus în subcapitolul anterior, normalizarea datelor poate ajuta rețeaua să învețe mai repede și mai bine. De aceea, și la normalizarea batch-ului vom folosi aceeași idee, de a păstra datele într-un anumit interval. Normalizarea batch-ului înseamnă de fapt normalizarea activărilor înainte ca acestea să fie date ca intrare la următorul layer. Așadar, pentru normalizare, vom folosi următoarea formulă:

$$x = \frac{\gamma(x - \mu)}{\sigma + \varepsilon} + \beta \quad (4.1)$$

Unde x este valoarea unei activări, μ este media tuturor activărilor, σ este varianța, ε este un număr foarte mic pentru a se evita împărțirea la 0, iar γ și β sunt parametri pe care îi învață rețeaua. Ce este interesant, este faptul că dacă $\gamma = \sigma$ și $\beta = \mu$ atunci practic nu se aplică nicio normalizare, deci rețeaua poate învăța și lucrul acesta.

4.3.3 Tangenta hiperbolică

Tangenta hiperbolică [26] este tot o funcție neliniară de activare, precum ReLU, aceasta având valori între $[-1, 1]$. Formula acesteia este:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (4.2)$$

4.3.4 Layer transpus de conoluție

Layerul transpus de conoluție merge pe același principiu cu cel al layerului de conoluție, doar că în cazul astă de la o imagine de o dimensiune oarecare, se dorește obținerea unei imagini de dimensiune mai mare. În figura [4.3] se poate observa ideea acestui layer.

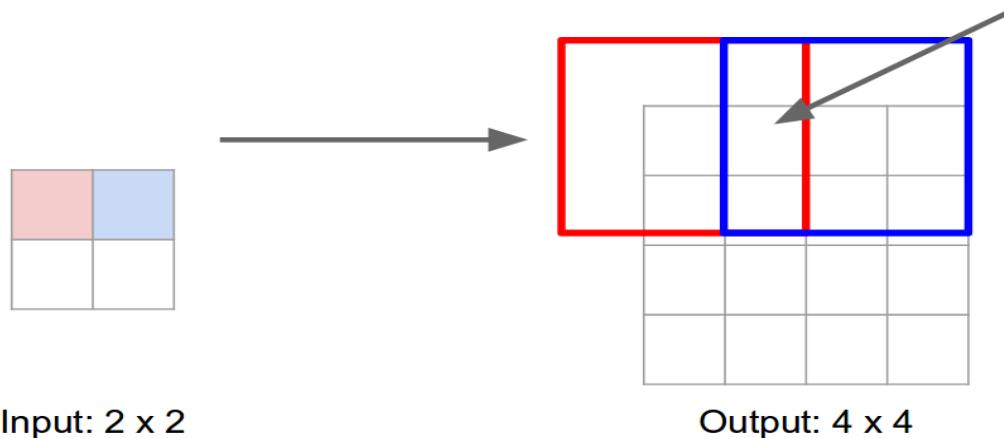


Figura 4.3: Conoluție transpusă [22]

După cum se observă și în poză, pe rând, pixelii din poza inițială se vor înmulți cu valorile filtrului care se plimbă pe poza mărită, iar aceste valori se vor copia în locurile respective. În locurile unde mai multe filtre se suprapun, se vor aduna valorile. De asemenea se poate decide câți pixeli să sară un filtru atunci când este plimbat pe poza mărită, în cazul de față, filtrul sare peste 2 pixeli.

4.3.5 Rețele reziduale

Rețelele neurale reziduale [23] au fost introduse pentru a fi posibilă antrenarea unor rețele foarte adânci. Dacă în teorie, mărirea numărului de layere ar însemna ca rețea să învețe mai bine pe setul de antrenare, în practică s-a dovedit contrariul, o cauză fiind numărul foarte mare de parametri pe care rețea trebuie să îi optimizeze. Așadar, introducerea de blocuri reziduale permite crearea de rețele adânci care să obțină performanțe cel puțin la fel de bune ca o rețea nu la fel de adâncă.

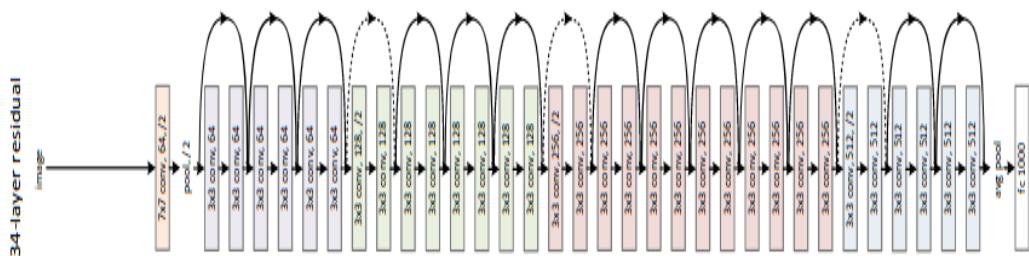


Figura 4.4: Arhitectura unei rețele reziduale [7]

După cum se poate vedea în figura [4.4], rețelele reziduale, introduc conexiuni care săr peste anumite layere, numite blocuri reziduale. Așadar introducerea acestora nu afectează cu nimic rău rețeaua, aceasta putând învăța să aplique funcția identitate în interiorul blocurilor pentru a le ignora, ci doar să îmbunătățească performanța rețelei.

4.3.6 Microsoft COCO

Microsoft Common Objects in COntext (COCO) [8] este tot un set de date asemănător cu ImageNet ce conține un număr impresionant de imagini annoteate, care pot fi folosite în diferite probleme, precum segmentarea imaginii sau recunoașterea de obiecte dintr-o imagine. În figura [4.5] se pot vedea câteva exemple.



Figura 4.5: Imagini din setul de date Microsoft COCO din 2014

4.4 Rezultate și comparații

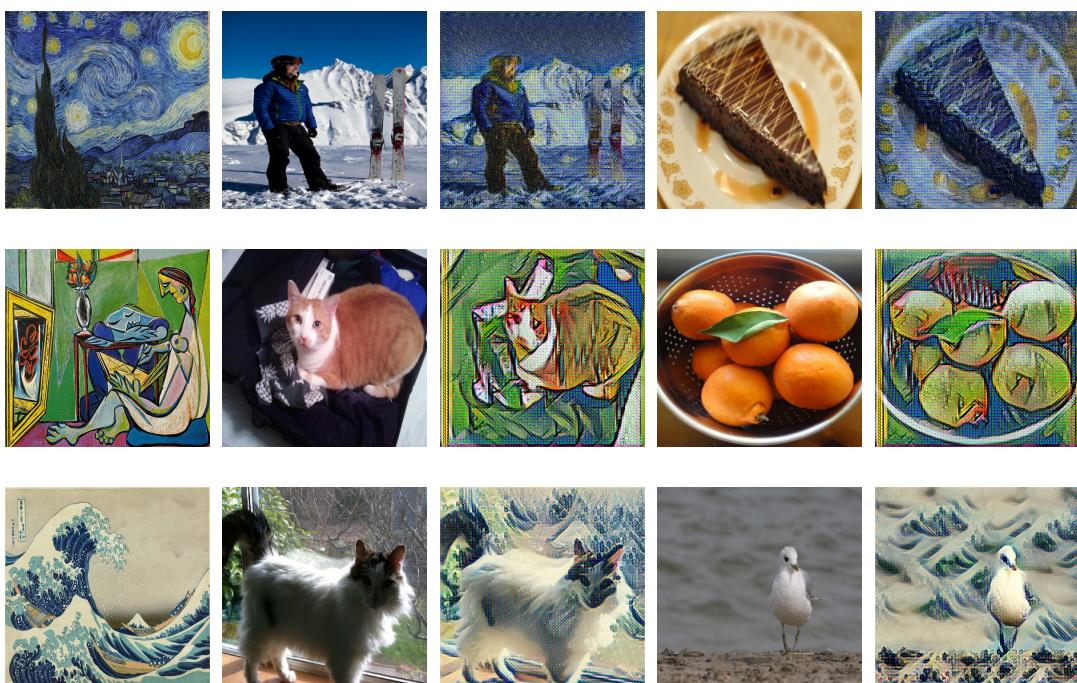


Figura 4.6: Imagini de dimensiune 256 x 256 pixeli



Figura 4.7: Deși rețeaua a fost antrenată pe poze de dimensiune 256×256 de pixeli se poate vedea în figura de mai sus că rețeaua se comportă bine și pe poze de dimensiune mai mare, imaginile din figură având dimensiunea de 512×512 de pixeli.

Capitolul 5

Compararea metodelor

Deși aceste metode rezolvă aceeași problema, de a combina conținutul unei poze cu stilul alteia pentru a obține o nouă poză artistică ele pot fi aplicate în funcție de necesitatea utilizatorului. De exemplu, dacă utilizatorul dorește să combine picturi celebre atunci metoda lui Leon A. Gatys este cea mai potrivită. În schimb dacă se dorește doar combinarea unor anumite obiecte pentru obținerea unor noi rezultate interesante, atunci se poate aplica metoda lui Fujun Luan. Pentru o generare rapidă a pozelor artistice se poate apela la metoda lui Justin Johnson, metodă cu care se pot obține rezultate asemănătoare cu metoda lui Leon A. Gatys, dar care are un timp de rulare foarte mic.

Mai jos o să arăt câteva exemple pentru a evidenția îmbunătățirile aduse de ultimele două metode în comparație cu prima.

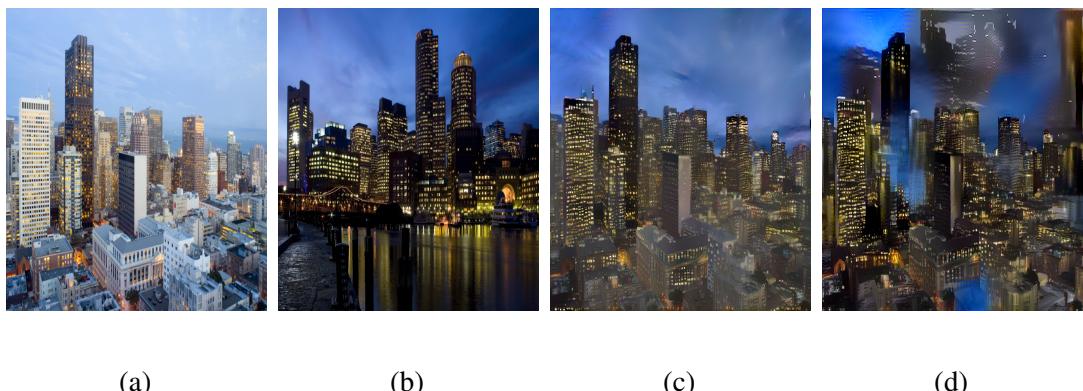


Figura 5.1: În această figură poza (a) este poza cu conținutul pe care dorim să îl obținem, iar poza (b) este poza cu stilul dorit. Poza (c) a fost generată cu algoritmul din capitolul [3] pe când poza (d) a fost generată cu algoritmul din capitolul [2]. Se observă că dacă nu sunt folosite măști pentru a controla unde să se aplique stilul, atunci înțelesul semantic nu este păstrat, după cum se poate vedea de exemplu că părți ale cerului au fost generate pe anumite clădiri din imagine, fapt pe care nu ni-l dorim în cazul pozelor fotografice.

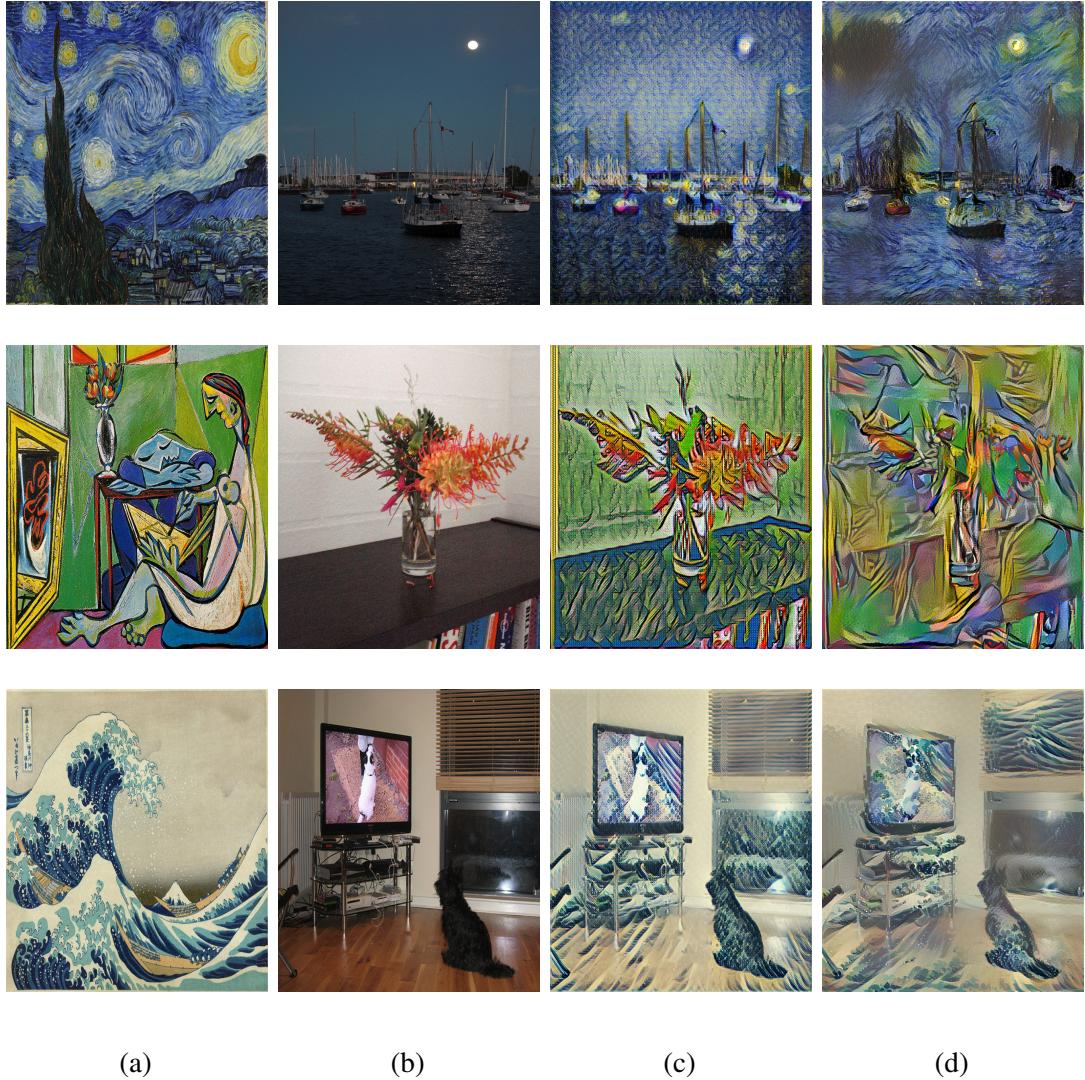


Figura 5.2: În această figură poza (a) este poza cu conținutul pe care dorim să îl obținem, iar poza (b) este poza cu stilul dorit. Pozele de pe coloana (c) a fost generate cu algoritm din capitolul [4] pe când pozele de pe coloana (d) au fost generate cu algoritm din capitolul [2]. Se observă ca pozele generate cu cele două metode sunt aproximativ la fel, cele de pe coloana (c) punând mai mult accent pe conținut, pe când cele de pe coloana (d) pun accent mai mult pe stil, însă este vorba doar de preferință între cele două metode. O diferență majoră însă între cele două este timpul necesar generării unei astfel de poze. Am testat pe laptopul meu care are o placă video NVIDIA GeForce 840M și rezultatele sunt următoarele: timpul necesar pentru generarea unei poze de pe coloana (c) este de aproximativ 4secunde, pe când timpul necesar generării unei poze de pe coloana (d) pentru a obține rezultate satisfăcătoare este de aproximativ 100deminute, menționez că dimensiunea pozelor este de 512 x 512 de pixeli.

Capitolul 6

Concluzii

În această lucrare am arătat, prin exemplificarea a trei metode care pot fi aplicate în funcție de tipul problemei, că având două poze este posibil să combinăm conținutul primei poze cu stilul celeilalte pentru a obține o nouă poză artistică.

6.1 Dezvoltări ulterioare

Desigur că aceste metodele descrise în această lucrare mai pot fi îmbunătățite și de aceea pe viitor mi-am propus să încerc următoarele: să schimb rețeaua pe baza căreia calculez costurile pentru conținut și stil și să vad ce rezultate obțin, de asemenea o să încerc și alte seturi de date pentru antrenarea rețelei din capitolul [4] sau să o antrenez un număr mai mare de iterații. Și nu în ultimul rând, aş dori să încerc o metodă pentru pozele fotografice pentru ca figurile geometrice din poza inițială cu conținut să nu prezinte distorsiuni în imaginea generată.

Bibliografie

- [1] Y. LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [2] Glorot Xavier and Bengio Yoshua. “Understanding the difficulty of training deep feedforward neural networks”. In: (2010).
- [3] Zeiler Matthew D. and Fergus Rob. “Visualizing and Understanding Convolutional Networks”. In: *arXiv preprint arXiv:1311.2901v3* (2013).
- [4] Mahendran Aravindh and Vedaldi Andrea. “Understanding Deep Image Representations by Inverting Them”. In: *arXiv preprint arXiv:1412.0035v1* (2014).
- [5] Gatys Leon A., Ecker Alexander S., and Matthias Bethge. “A Neural Algorithm of Artistic Style”. In: *arXiv preprint arXiv:1508.06576v2* (2015).
- [6] Gatys Leon A., Ecker Alexander S., and Matthias Bethge. “Texture Synthesis Using Convolutional Neural Networks”. In: *arXiv preprint arXiv:1505.07376v3* (2015).
- [7] He Kaiming et al. “Deep Residual Learning for Image Recognition”. In: *arXiv preprint arXiv:1512.03385v1* (2015).
- [8] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *arXiv preprint arXiv:1405.0312v3* (2015).
- [9] Johnson Justin, Alahi Alexandre, and Fei-Fei Li. “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”. In: *arXiv preprint arXiv:1603.08155v1* (2016).
- [10] Johnson Justin, Alahi Alexandre, and Fei-Fei Li. “Perceptual Losses for Real-Time Style Transfer and Super-Resolution: Supplementary Material”. In: (2016).
- [11] Luan Fujun et al. “Deep Photo Style Transfer”. In: *arXiv preprint arXiv:1703.07511v3* (2017).

- [12] Simonyan Karen and Zisserman Andrew. *Very Deep Convolutional Networks for Large-Scale Visual Recognition*. 2018. URL: http://www.robots.ox.ac.uk/~vgg/research/very_deep/.
 - [13] Andrew Ng. *Art generation with Neural Style Transfer*. 2018. URL: <https://www.coursera.org/learn/convolutional-neural-networks/notebook/lEthw/art-generation-with-neural-style-transfer>.
 - [14] Andrew Ng. *What are deep ConvNets learning?* 2018. URL: <https://www.coursera.org/learn/convolutional-neural-networks/lecture/GboGx/what-are-deep-convnets-learning>.
 - [15] *Chicago*. URL: <https://raw.githubusercontent.com/jcjohnson/fast-neural-style/master/images/content/chicago.jpg>.
 - [16] *Convolutional Neural Network*. URL: https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/02_Convolutional_Neural_Network.ipynb.
 - [17] *Dog Breed Identification - Team 4*. URL: <https://telecombcn-dl.github.io/2017-dlai-team4/>.
 - [18] *Gramian Matrix*. URL: https://en.wikipedia.org/wiki/Gramian_matrix.
 - [19] *Hoover Tower*. URL: <https://raw.githubusercontent.com/jcjohnson/neural-style/master/examples/inputs/hoovertowernight.jpg>.
 - [20] *https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117*. URL: <https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117>.
 - [21] *ImageNet*. URL: www.image-net.org/.
 - [22] *Lecture 11 — Detection and Segmentation*. URL: <https://www.youtube.com/watch?v=nDPWywWRIRo>.
 - [23] Andrew Ng. *Deep Learning*. URL: <https://www.coursera.org/specializations/deep-learning>.
-

- [24] *Python*. URL: [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)).
- [25] *ReLU*. URL: [https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)).
- [26] *Tanh*. URL: https://en.wikipedia.org/wiki/Hyperbolic_function.
- [27] *TensorFlow*. URL: <https://en.wikipedia.org/wiki/TensorFlow>.
- [28] *Total variation denoising*. URL: https://en.wikipedia.org/wiki/Total_variation_denoising.