



UNIVERSITATEA DIN
BUCUREŞTI



FACULTATEA DE
MATEMATICĂ ȘI
INFORMATICA

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

**COMBINAREA RETELELOR GENERATIVE
ADVERSARIALE ȘI MODELELOR DE
SEGMENTARE PENTRU TRANSFERUL
STILULUI IMAGINILOR CU PERSOANE**

Absolvent
Năstase Matei-Dorian

Coordonator științific
Prof. Dr. Ionescu Radu Tudor

București, iunie 2021

Rezumat

Anime este un stil de animație specific culturii japoneze care se bucură de foarte mult succes la nivel global. Fie că este vorba de romane vizuale (manga) sau de ecranizările acestora (seriale anime), există un număr mare de oameni care consumă acest tip de conținut. Cu toate acestea, stilul anime reprezintă la rândul său o artă, și ca orice tip de animație, atât crearea de seriale noi, cât și de manga, necesită un număr foarte mare de artiști. Lucrarea de față propune o metodă de antrenare, bazată pe studii relevante în domeniu, a unui ansamblu de rețele de ultimă generație (un CycleGAN și un SegFormer) capabil de a transfera stilul specific anime în poze cu persoane reale. Transferul de stil se va face localizat la nivelul feței persoanelor, iar modelul de învățare automată este însotit de o interfață grafică ce facilitează utilizarea sa. Credem că un astfel de algoritm ar putea reprezenta o sursă de inspirație care să asiste în procesul de creație al artiștilor ce folosesc acest stil. Mai mult, efectul inedit de colaj creat deschide porțile și către alte aplicații, de exemplu filtre pentru rețelele de socializare și jocuri ce folosesc realitate augmentată.

Abstract

Anime is an animation style deeply rooted in Japanese culture that has gained great success worldwide. Whether we are talking about graphic novels (manga) or their screenings (anime series), there is a huge fanbase that loves this type of content. Despite that, anime is also an art form, and there are lots of artists involved in the making of any manga or anime series. This paper introduces a new model ensemble, based on several studies relevant to this field, that combines state-of-the-art network architectures (a CycleGAN and a SegFormer) capable of transferring the anime style to pictures with real people. The style transfer effect will be localized only in the face area, and the model is integrated into a graphical interface that will facilitate the use of this machine learning algorithm. We believe that this algorithm can be used as a starting point and aid the creation process of the artists that use this style. Moreover, the unique collage effect that is created in the generated pictures opens the gate to other types of applications, such as social media filters and games that use virtual reality.

Cuprins

1 Introducere	5
1.1 Detalierea temei	5
1.2 Motivația	5
2 Concepte teoretice	7
2.1 Ce este învățarea automată	7
2.1.1 Etapele dezvoltării unui program de învățare automată	8
2.1.2 Tipuri de învățare automată	9
2.1.3 Transferul de cunoștințe	9
2.1.4 Ansamblul de modele	10
2.2 Rețele neurale și Deep Learning	10
2.2.1 Modul de lucru al unei rețele neurale	10
2.2.2 Perceptronul	11
2.2.3 Funcții de activare	11
2.2.4 Antrenarea unei rețele neurale	12
2.2.5 Underfit vs Overfit	13
2.3 Generarea de imagini	14
2.3.1 Generarea adversarială	14
2.3.2 CycleGAN și transferul de stil	15
2.3.3 Arhitectura unui CycleGAN	17
2.3.4 Avantaje și dezavantaje	19
2.3.5 Evaluarea imaginilor generate	19
2.4 Segmentarea imaginilor	21
2.4.1 Transformeri	21
2.4.2 Atenția	22
2.4.3 Transformerul vizual - ViT	23
2.4.4 SegFormerul	24
3 Abordări recente	26
3.1 Transferul stilului unor pictori celebri	26
3.2 Transferul stilului anime	27

3.2.1	Arhitectura clasică	27
3.2.2	Arhitecturi noi	28
4	Tehnologii folosite	30
5	Abordarea propusă	32
5.1	Transferul stilui anime	32
5.1.1	Crearea unui set de date	32
5.1.2	Preprocesarea imaginilor	34
5.1.3	Arhitectura modelelor	34
5.1.4	Modul de antrenare	36
5.1.5	Evaluarea modelului	37
5.2	Segmentarea semantică	38
5.2.1	Alegerea setului de date	38
5.2.2	Rafinarea modelului pre-antrenat	39
6	Experimente și rezultate	41
6.1	Antrenarea CycleGAN-ului	41
6.2	Antrenarea SegFormerului	47
7	Prezentarea aplicației	50
7.1	Funcționalități	50
7.2	Structura aplicației	50
7.3	Galerie de rezultate	53
8	Concluzii și dezvoltări ulterioare	54
Bibliografie		56

Capitolul 1

Introducere

1.1 Detalierea temei

Încă de la apariția sa în jurul anilor 1950, domeniul inteligenței artificiale a stârnit interesul cercetătorilor deoarece pleca de la o premisă inovatoare și poate puțin controversată și anume, crearea unor mașini capabile să imite inteligența umană. Astăzi, inteligența artificială a devenit o unealtă puternică și accesibilă, iar companii precum IBM estimează că până la 7.800 de locuri de muncă vor fi puternic influențate sau poate chiar înlocuite complet de programe cu inteligență artificială în următorii ani [8]. Cu toate acestea, se crede că unele aspecte ale gândirii umane precum curiozitatea, raționamentul moral, creativitatea și emoțiile nu pot fi redate printr-un algoritm.

Lucrarea de față își propune crearea și antrenarea unor rețele neuronale de ultimă generație (state-of-the-art) capabile de a transforma poze cu oameni reali în poze cu personaje din desene animate în stil anime. Anime-ul este un tip de animație specific culturii japoneze care se distinge prin stilul și expresivitatea personajelor, fiind considerat și o formă de artă. Acesta poate fi întâlnit atât în benzi desenate sau romane vizuale (manga), cât și în adaptări ale acestor cărți în filme sau seriale, reprezentând o parte importantă a culturii japoneze și având o popularitate mare la nivel internațional. Obiectivul acestei arhitecturi este de a învăța stilul recognoscibil al artiștilor de animație japonezi și de a-l reproduce prin transformarea persoanelor din poze în varianta lor anime, simulând astfel într-o anumită măsură, creativitatea umană. Mai mult, acest efect de transfer de stil va fi aplicat localizat, doar la nivelul feței, păstrând restul imaginii neschimbă și creând astfel un efect de colaj, un mix între realitate și conținut generat automat.

1.2 Motivația

Învățarea automată a devenit o pasiune pe plan academic încă de la primul contact cu acest domeniu, în cadrul cursului de Inteligență Artificială. Ce m-a atras spre acest do-

meniu a fost felul în care conceptele din informatică se îmbină perfect cu cele matematice, dar și comunitatea de cercetare ce s-a format în jurul acestei arii, care este foarte mare și primitoare. În același timp, pe plan personal, am avut mereu o înclinație spre artă, în special pentru desen și crearea de materiale vizuale, aşa că am încercat să găsesc o temă de cercetare care să combine ambele pasiuni. Pe lângă interesul personal asupra temei propuse, acest studiu are și o aplicație practică importantă, deoarece există un număr foarte mare de fani și multe evenimente dedicate anime-urilor, organizate la nivel global. Cu toate acestea, crearea de noi seriale de acest gen este un proces lent și foarte costisitor, întrucât este nevoie de un număr foarte mare de artiști care să deseneze fiecare cadru în parte, ecranizarea unui manga putând dura chiar și câțiva ani. Așadar, un algoritm care ar putea să genereze astfel de conținut automat ar reprezenta o sursă de inspirație sau o unealtă ce ar facilita procesul de creație pentru artiștii din acest domeniu.

Capitolul 2

Concepte teoretice

2.1 Ce este învățarea automată

Învățarea automată (Machine learning) este o ramură a inteligenței artificiale care se ocupă cu crearea de programe capabile de a face predicții sau de a lua decizii singure, fără a fi programate în mod explicit. Spre deosebire de programarea imperativă, în care programul este scris într-un mod secvențial și deterministic, urmând un set de pași bine stabiliți, programele ce folosesc învățare automată se bazează pe diverse concepte matematice pentru a-și îmbunătăți performanța, învățând din experiență.

Traditional Programming

Developers write rules (program) that produce an output.



Machine Learning

Developers write a training algorithm, that finds rules, which produce the desired output.

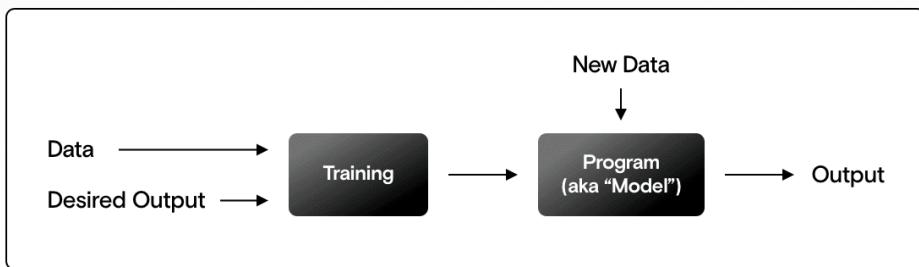


Figura 2.1: Diferențele dintre un program tradițional și un program cu învățare automată [25]

Practic, programatorul specifică doar algoritmul de învățare, iar programul învață propria sa reprezentare (numită și ”modelul”) a datelor cu care a fost antrenat. Astfel, putem vorbi despre un program care se scrie pe sine însuși, deoarece programul este capabil să ajusteze propriul său model intern, în maniera specificată de programator.

2.1.1 Etapele dezvoltării unui program de învățare automată

În dezvoltarea unui program ce folosește învățare automată se evidențiază următorii pași esențiali [9] [22] :

1. **Definirea problemei:** este important ca problema pe care încearcă să o rezolve programul să fie bine definită încă de la început; cu cât cerințele problemei sunt mai specifice, cu atât modelul va fi mai puternic, deoarece va putea învăța tiparele problemei respective, în loc să încerce să formeze niște reguli generale care pot introduce ambiguitate. De asemenea, acesta este un punct bun pentru a stabili cum va fi evaluat modelul, ce metrică va fi folosită, care este obiectivul de bază (baseline accuracy);
2. **Crearea unui set de date:** Calitatea și cantitatea datelor influențează direct performanța unui model. Datele trebuie să fie diverse și reprezentative pentru problema aleasă, iar în general, deși un număr de date mai mare rezultă adesea într-o performanță mai bună, trebuie să fie echilibrat volumul de date deoarece un volum mare duce la un timp de antrenare mai lung și la mai multe resurse folosite. Colectarea datelor se poate face prin diferite metode, folosind baze de date, API-uri sau prin web scrapping¹;
3. **Preprocesare:** Datele brute sunt adesea incomplete și inconsistente, putând conține valori lipsă sau erori. Preprocesarea implică curățarea și transformarea datelor brute într-un format potrivit pentru învățare. Tot aici, se pot aplica diverse tehnici de augmentare a datelor pentru a crește volumul de date sau pentru a îmbunătăți calitatea acestora;
4. **Ingineria feature-urilor:** În acest stadiu, datele care pot fi sub formă de fișiere text, imagini, semnale audio, etc. sunt convertite într-o reprezentare numerică care evidențiază proprietățile (features) acestora, cu ajutorul cărora modelul va învăța;
5. **Alegerea modelului/architecturii:** Alegerea modelului depinde de natura problemei și a datelor. Câteva modele comune includ: arbori de decizie (decision trees), mașini cu vectori suport (support vector machines) și rețele neurale (neural networks);

¹Web scraping se referă la procesul de extragere automată a datelor din paginile web. Pentru aceasta se folosesc librării specializate pentru accesarea paginilor web, analizarea structurii lor și extragerea datelor relevante.

6. **Antrenarea modelului:** Modelul ales este antrenat pe datele preprocesate și transformate în features și implică trecerea fiecărei intrări din setul de date prin algoritmul de învățare, urmând ca modelul să-și calibreze parametrii interni la fiecare pas;
7. **Validarea modelului:** Acest proces implică testarea modelului pe date noi (diferite de cele de antrenare) și calcularea unor metrii care pot arăta cât de bine a decurs procesul de antrenare;
8. **Optimizarea hiperparametrilor:** Hiperparametrii pot fi priviți ca niște setări ale modelului. De multe ori procesele de antrenare și validare sunt repetate cu diferite seturi de hiperparametri, deoarece aceștia pot influența în bine sau în rău performanța algoritmului.

2.1.2 Tipuri de învățare automată

Acest domeniu se împarte la rândul lui în trei mari paradigmă de învățare [16] [9]:

- **Învățarea supervizată:** este cel mai întâlnit tip și implică antrenarea modelelor pe seturi de date cu etichete (labeluri). Astfel, algoritmul primește atât datele de intrare, cât și rezultatul dorit (care constă în valoarea etichetei) și este capabil să își ajusteze parametrii în funcție de cât de aproape/departe este față de acest rezultat. Este folosită în general pentru problemele de clasificare și regresie;
- **Învățarea nesupervizată:** reprezintă opusul metodei anterior menționate, antrenarea în acest caz realizându-se cu un set de date neetichetat. Aici, algoritmul învăță similaritățile dintre exemplele din setul de date, acest tip de învățare fiind folosit în general pentru probleme de clustering (gruparea datelor în diverse categorii);
- **Învățare prin recompensă (Reinforcement learning):** aici modelul învăță prin metoda trial and error; modelul este lăsat să interacționeze cu un anumit mediu, acesta fiind "răsplătit" pentru acțiunile bune sau "penalizat" pentru cele nedorite, scopul algoritmului fiind să maximizeze scorul obținut. Folosind această paradigmă, se pot antrena modele care să învețe să joace jocuri video.

În experimentele noastre a fost folosită o combinație între învățarea supervizată și nesupervizată, regăsită adesea sub numele de învățare semi-supervizată în literatura de specialitate.

2.1.3 Transferul de cunoștințe

Antrenarea unui model este costisitoare din punctul de vedere al timpului și al resurselor computaționale, de aceea o practică comună în acest domeniu este transferul de

cunoștințe (knowledge transfer). Una dintre cele mai folosite metode de transfer al cunoștințelor este folosirea de modele pre-antrenate [19] pe o sarcină mai generală, cu un set de date masiv, și rafinarea acestora pentru rezolvarea unor probleme mai specifice. În acest caz, modelele pre-antrenate conțin deja cunoștințe despre extragerea trăsăturilor generice, iar specializarea lor pe o problemă mai restrânsă (fine-tuning) necesită un număr mai mic de date și un timp de antrenare redus.

2.1.4 Ansamblul de modele

Așa cum a fost menționat mai devreme, un model este mai puternic atunci când încearcă să rezolve o problemă (task) mai specifică, bine definită. De aceea, o abordare frecvent întâlnită în rezolvarea unor task-uri mai ample este formarea de ansambluri de modele. Un ansamblu de modele reprezintă o colecție de modele similare sau diverse, care lucrează împreună pentru a obține un rezultat mai bun decât ar putea realiza fiecare model individual. Există mai multe strategii de combinare a modelelor, dar în cazul acestui studiu s-a folosit o strategie de înlănțuire (cunoscută sub numele de chaining model sau cascading model), în care outputul primului model este folosit ca input pentru al doilea.

2.2 Rețele neurale și Deep Learning

Rețelele neurale reprezintă un model de învățare automată bazat pe o reprezentare simplificată a modului de funcționare al creierului uman[12]: mai mulți neuroni sunt legați unii de alții prin sinapse, iar un neuron procesează impulsuri nervoase ca mai apoi să trimită mai departe aceste semnale la neuronii cu care este conectat. Analog, perceptronul, sau neuronul artificial, reprezintă unitatea de bază a unei rețele neurale. Acești perceptri sunt grupați în straturi (layers), iar cu cât o rețea are mai multe straturi, aceasta devine mai adâncă (deep), fiind capabilă să învețe reprezentări ale datelor mult mai complexe, de aici venind și termenul de Deep Learning.

2.2.1 Modul de lucru al unei rețele neurale

În linii mari, scopul unei rețele neurale se reduce la aproximarea unei funcții, unde datele de intrare ale programului fac parte din domeniul funcției, iar datele de ieșire din codomeniu. Spre exemplu, în cazul acestui studiu, am dori să aproximăm o funcție f , capabilă să mapeze multimea pozelor cu oameni reali la multimea pozelor cu personaje în stil anime. Practic, în timpul procesului de antrenare, rețeaua învață parametrii acestei funcții.

În general, o rețea neuronală urmează următoarea structură:

- Stratul de intrare: care are o dimensiune egală cu dimensiunea datelor de intrare;
- Stratul de ieșire: care are o dimensiune egală cu cea a datelor de ieșire;
- Straturile ascunse: reprezintă straturile de perceptri intermediare; numărul de straturi (adâncimea rețelei) și felul în care sunt legați perceptronii depinde de arhitectura rețelei.

Majoritatea arhitecturilor sunt de tip feed-forward, adică datele circulă prin rețea în mod unidirectional prin stratul de intrare, apoi straturile ascunse și la final, stratul de ieșire.

2.2.2 Perceptronul

Perceptronul reprezintă un nod dintr-o rețea neurală și, la rândul său, acesta modelază o funcție [1] de forma $f(x) = w * x + b$:

- x reprezintă intrarea perceptronului - datele de intrare în cazul stratului de intrare sau outputul altor perceptri în cazul celorlalte straturi;
- w este ponderea perceptronului (weight-ul) și reprezintă parametrul ce trebuie învățat și este ajustat în timpul procesului de antrenare;
- b se numește bias, și este un termen constant prin care se poate controla pragul de la care un perceptron se activează sau nu;
- $f(x)$ reprezintă ieșirea perceptronului.

Este important de precizat că în formula de mai sus x nu este întotdeauna un număr, ci mai degrabă un vector (tensor) de dimensiune n . Astfel, pentru un vector x de forma $x_1 x_2 \dots x_n$, vom avea weight-urile corespunzătoare $w_1 w_2 \dots w_n$, putând privi modul de acționare al unui perceptron ca pe o înmulțire între matrice.

2.2.3 Funcții de activare

În cadrul unei rețele neurale, fiecare funcție modelată de un perceptron este folosită pentru a ajunge la un rezultat final. Desigur, funcția $f(x) = w * x + b$ este o funcție liniară, iar rezultatul compunerii de funcții liniare este de asemenea o funcție liniară, aşa că o întrebare firească este cum reușesc rețelele neurale să învețe mapări atât de complexe, care sunt de cele mai multe ori funcții neliniare. Răspunsul la această întrebare sunt funcțiile de activare. Acestea sunt funcții neliniare (precum funcția sigmoidă, tangentă hiperbolică sau ReLU) care fac parte din structura perceptronului. Înainte ca rezultatul $f(x)$ să fie pasat la neuronii din straturile următoare, această valoare este trecută prin funcția de activare, iar dacă rezultatul obținut depășește un anumit prag (threshold), neuronul este

activat, iar valoarea care va fi transmisă mai departe va fi rezultatul obținut în urma aplicării funcției de activare, introducând astfel neliniaritate în modelul nostru.

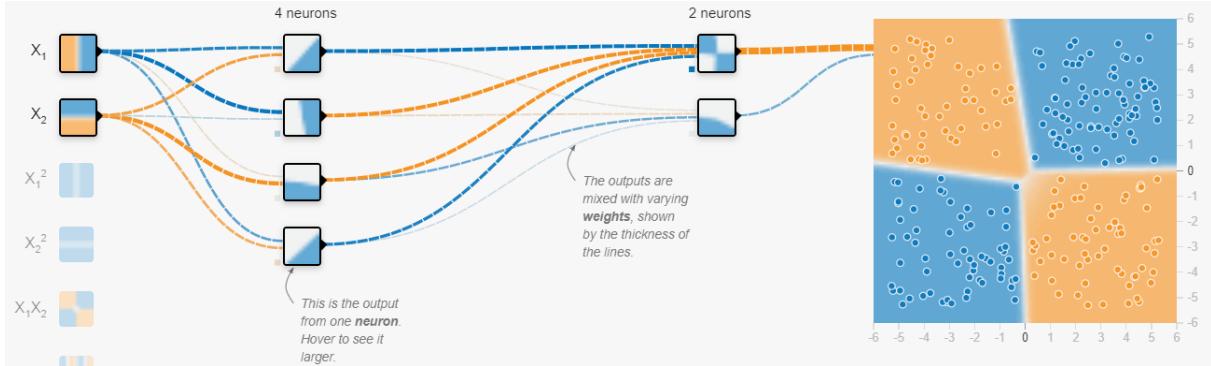


Figura 2.2: Exemplu de rețea neuronală simplă.

Imaginea a fost generată cu ajutorul acestui site: <https://playground.tensorflow.org>

În figura 2.2 s-a evidențiat modul de funcționare al unei rețele neuronale simple, care folosește funcția ReLU ca funcție de activare. Astfel, se poate observa cum mai mulți perceptri grupei în straturi pot funcționa împreună într-o rețea, modelând o funcție neliniară. În teorie, având un număr suficient de mari de perceptri, se poate modela cu o rețea neurală orice funcție continuă, până la un anumit grad de precizie stabilit.

2.2.4 Antrenarea unei rețele neurale

Modul de antrenare al unei rețele este structurat în epoci. Fiecare epochă începe prin trecerea datelor de antrenare prin rețea (pasul forward) și obținerea unui rezultat. Acest rezultat este comparat cu rezultatul așteptat, calculându-se un loss (loss-ul reprezintă o metodă de a quantifica cât de apropiat este rezultatul rețelei de rezultatul real). Scopul rețelei este de a minimiza funcția de loss (evoluția loss-ului pe parcursul epochilor). Acest lucru este realizat prin calcularea gradientilor (derivata de ordin întâi a funcției de loss, în raport cu weight-urile și biasul) în sensul invers al straturilor din rețea, de la stratul de ieșire până la cel de intrare, proces cunoscut și sub numele de backpropagation. Gradientii oferă informații despre cum va afecta ajustarea unui parametru valoarea loss-ului, fiecare weight în parte fiind ajustat printr-un algoritm de optimizare precum Gradient Descent.

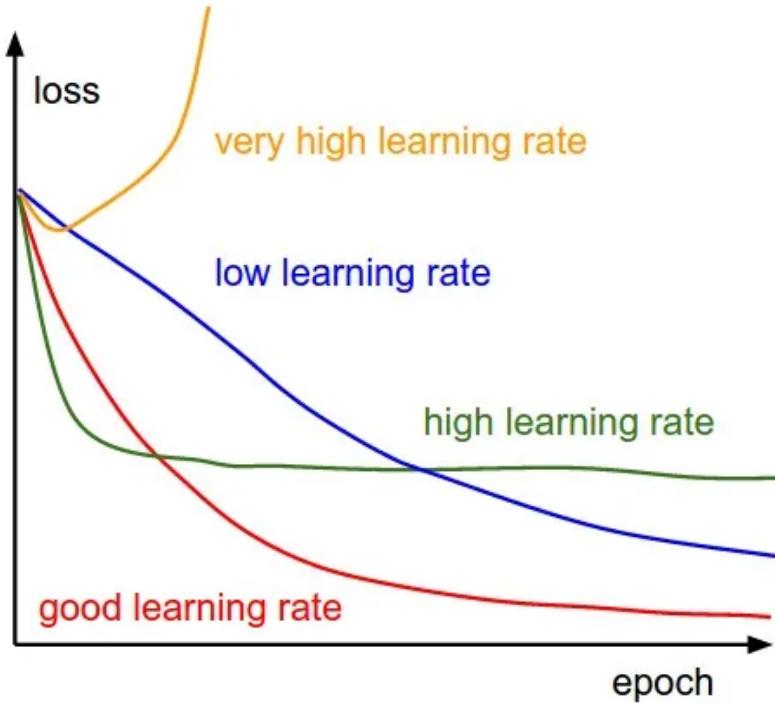


Figura 2.3: Modul în care learning rate-ul influențează procesul de învățare [15]

În figura 2.3 se poate observa importanța learning rate-ului în procesul de antrenare. Learning rate-ul este un parametru important pentru o rețea neurală deoarece acesta controlează proporția prin care vor fi ajustate weight-urile aplicând algoritmul de optimizare. În mod ideal, un grafic de loss ar arăta ca linia roșie din graficul de mai sus, valoarea loss-ului scăzând în mod asymptotic în raport cu axa abscisă. În realitate însă, este normal ca această descreștere să fluctueze puțin, iar valoarea parametrului de learning rate poate determina viteza cu care o rețea converge² (linia roșie vs linia albastră), sau dacă aceasta reușește sau nu să conveargă (linia roșie vs linia galbenă).

2.2.5 Underfit vs Overfit

Antrenarea unei rețele neuronale nu este un procedeu simplu. De multe ori funcția aproximare pe care o obținem nu este întotdeauna cea mai potrivită, conducând la apariția a două dintre cele mai întâlnite probleme din domeniul învățării automate: overfitul și underfitul. Overfitul reprezintă o aproximare prea strânsă pe datele de antrenare. Rețeaua nu reușește în acest caz să generalizeze tiparele și regulile prezente în setul de date și, de multe ori, deși modelul se descurcă bine pe datele de antrenare, acesta nu face față datelor noi. Underfitul este opusul overfitului, atunci când modelul nu reușește să învețe anumite tipare din setul de antrenare, iar funcția de aproximare generată nu se potrivește cu graficul funcției dorite.

²Spunem că o rețea converge atunci când valoarea loss-ului se stabilizează în jurul unei valori apropriate de 0, moment în care procesul de învățare se oprește.

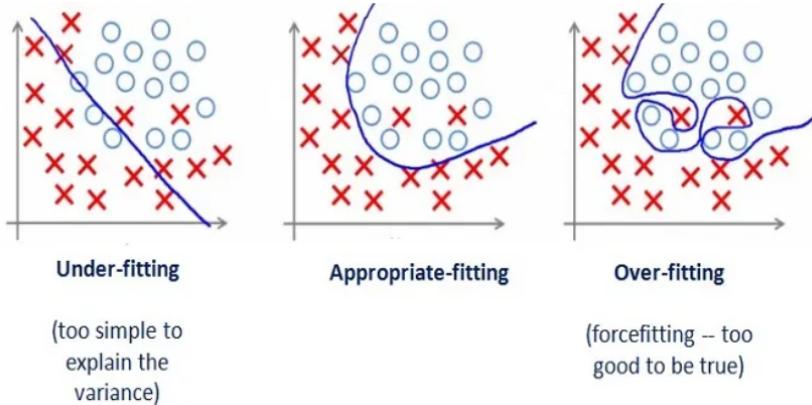


Figura 2.4: Diferențele dintre funcțiile de aproximare ce pot fi generate de o rețea [21]

De multe ori procesul de antrenare este de tip trial and error, în care arhitectura rețelei și valorile hiperparametrilor pot suferi modificări. De aceea este important ca setul de date să fie împărțit într-un set de antrenare și unul de validare, iar evoluția valorii loss-ului în timpul antrenării să fie urmărită.

2.3 Generarea de imagini

Task-ul de generare de imagini implică crearea de imagini noi, originale, pornind de la diverse date de intrare precum vectori de zgomot sau prompturi (o scurtă instrucțiune sau o frază inițială care descrie conținutul imaginii dorite). Primele rețele care au reușit să genereze imagini au fost bazate pe concepte din statistică precum inferența Bayesiană. Ideea principală a acestui tip de rețele era de a învăța o reprezentare latentă, de dimensiuni reduse a datelor [20]. Această reprezentare era folosită pentru a genera noi date din aceeași distribuție din care fac parte și datele de antrenare.

2.3.1 Generarea adversarială

În anul 2014, cercetătorii Ian J. Goodfellow și Jean Pouget-Abadie, propun în lucrarea "Generative Adversarial Networks", o nouă abordare asupra problemei generării de imagini. Noua arhitectură propune o idee inovatoare de antrenare a unei rețele ce nu folosește o abordare statistică. În schimb, arhitectura clasică a unui GAN este compusă dintr-o rețea generator (G) și o rețea discriminator (D). Cele două rețele reprezintă inamici care se antrenează simultan ca într-un joc (de aici venind și denumirea de rețea adversarială).

Rolul generatorului este de a învăța propria sa distribuție p_g peste setul de antrenare x . Acesta primește ca date de intrare un vector de zgomot z (biți aleși aleator dintr-o distribuție p_z) și învață o mapare $G(z, \theta_g)$, unde θ_g reprezintă parametrii modelului. Pe de altă parte, discriminatorul învață și el o funcție, $D(x, \theta_d)$, unde $D(x)$ reprezintă probabilitatea ca x să fi provenit din setul de antrenare, și nu din p_g . Se observă că scopurile

celor două rețele sunt contrare, discriminatorul încearcă să distingă între pozele reale și pozele generate, în timp ce generatorul încearcă să mute biții din vectorul de zgomot inițial astfel încât să formeze imagini ce păcălesc discriminatorul [11]. Antrenând aceste modele împreună, distribuția p_g ar trebui să devină apropiată de distribuția originală a datelor, astfel discriminatorul poate fi îndepărtat la finalul antrenării pentru a folosi doar generatorul la obținerea unor poze noi.

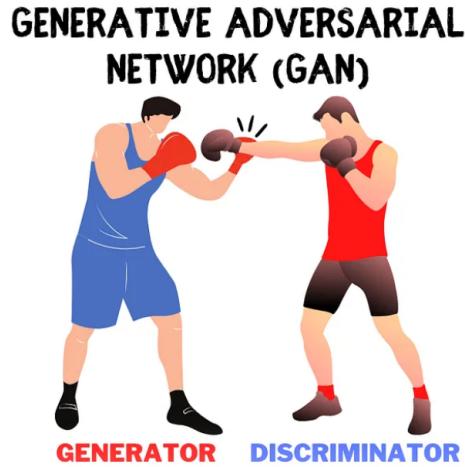


Figura 2.5: Ilustrare a modului adversarial de antrenare a rețelei [5]

Această metodă de antrenare s-a dovedit a fi foarte bună în practică, fapt ce a inspirat mai mulți cercetători să dezvolte diverse variații ale acestei arhitecturi, dând naștere unei întregi familii de rețele adversariale. Mai mult, o parte dintre aceste noi tipuri de GAN-uri au fost specializate pentru a rezolva și task-uri precum generarea de imagini de calitate înaltă (high resolution) și transferul stilului între imagini. Deși produc rezultate foarte bune, rețelele de tip GAN au anumite dezavantaje, una dintre cele mai mari probleme ce poate fi întâlnită în timpul antrenării numindu-se mode collapse. Mode collapse-ul poate avea loc atunci când generatorul reușește să redea anumite trăsături specifice setului de date de antrenare, suficient de convingătoare pentru a păcăli discriminatorul, dar nu suficient de diverse, pozele generate fiind foarte similare, indiferent de valoarea de intrare oferită rețelei.

2.3.2 CycleGAN și transferul de stil

Stilul de transfer între imagini reprezintă o nișă a generării de imagini. Spre deosebire de cazul general, aici algoritmul nu mai primește un vector de zgomot (biți aleatori), ci o poză dintr-o anumită mulțime (domeniu). Se dorește generarea unei imagini noi prin aplicarea unui stil dintr-un alt domeniu (schimbarea de culori sau texturi, sau modificări ușoare în structura imaginii), dar păstrând în mare compoziția imaginii inițiale.

Cercetătorii Jun-Yan Zhu și Taesung Park, introduc în lucrarea "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", o nouă arhitectură de tip GAN, concepută special pentru transferul de stil dintre imagini.

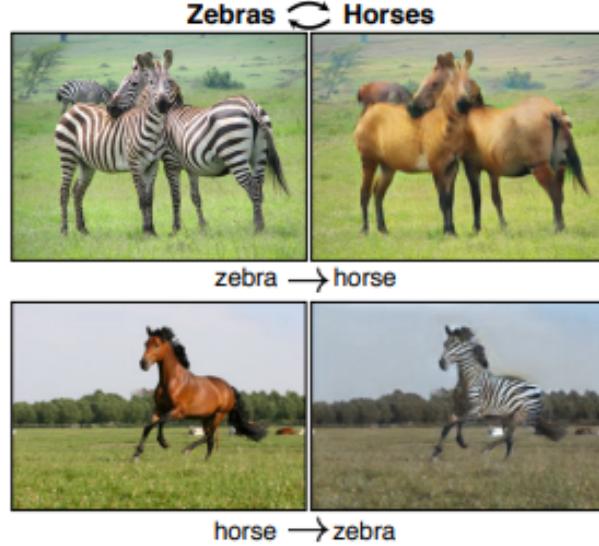


Figura 2.6: Rezultatele unui CycleGAN care transferă stilul dintre multimea A - poze cu cai, și multimea B - poze cu zebre [31]

Pe lângă obiectivul principal al generatorului de a păcăli discriminatorul, autorii lucrării introduc un nou obiectiv numit consistență într-un ciclu (cycle consistency). Ideea de bază provine din observația că funcția care mapează imaginile din domeniul A în domeniul B ar trebui să fie o bijecție, deoarece ne dorim ca fiecarei imagini din domeniul A să ii corespundă o imagine unică în domeniul B, și invers. Dacă funcția $f : A \rightarrow B$, aproximată de generator este bijectivă, atunci ar trebui să se poată construi și inversa funcției, $g : B \rightarrow A$, cu $g(x) = f^{-1}(x)$. Așadar, arhitectura CycleGAN, constă în două rețele de tip GAN ce se antrenează simultan. Primul GAN învață maparea $A \rightarrow B$, iar al doilea pe cea inversă. La final, loss-ul va fi calculat în funcție de cât de similară este imaginea inițială x , cu imaginea obținută prin $g(f(x))$ [31]. Obiectivul rețelei devine:

$$\begin{aligned} \mathcal{L}(F, G, D_A, D_B) &= \mathcal{L}_{GAN}(F, D_B, A, B) + \mathcal{L}_{GAN}(G, D_A, B, A) + \lambda \mathcal{L}_{cyc}(F, G) \\ \mathcal{L}_{GAN}(F, D_B, A, B) &= \mathbb{E}_{b \sim p_B} [\log D_B(b)] + \mathbb{E}_{a \sim p_A} [\log(1 - D_B(F(a)))] \quad (2.1) \\ \mathcal{L}_{cyc}(F, G) &= \mathbb{E}_{a \sim p_A} [\|G(F(a)) - a\|_1] + \mathbb{E}_{b \sim p_B} [\|F(G(b)) - b\|_1] \end{aligned}$$

În formula 2.1 am notat cu A, B cele două domenii între care vrem să facem transferul de stil, F este generatorul care aproximează funcția $f : A \rightarrow B$, iar G este generatorul pentru funcția $g : B \rightarrow A$, pe când D_A și D_B sunt discriminatoarele pentru cele două domenii. \mathcal{L}_{GAN} este funcția de loss tipică unui GAN, \mathcal{L}_{cyc} este loss-ul de consistență într-un ciclu, iar λ este o constantă, unul dintre hiperparametrii rețelei.

2.3.3 Arhitectura unui CycleGAN

La baza arhitecturii acestui model stau straturile de convoluție. Operația de convoluție se bazează pe înmulțirea de matrice și constă într-o fereastră de weight-uri de dimensiune n (kernel) care este glisată de-a lungul inputului. Dimensiunea ferestrei glisante este adesea un număr impar (valori întâlnite frecvent sunt 3 și 5), iar fereastra se mișcă cu un anumit pas de deplasare sau stride; un stride de 1 însemnând mutarea ferestrei cu un pixel la dreapta. În urma acestei operații se obține o hartă de caracteristici (feature map) care evidențiază prezența anumitor tipare în datele de intrare.

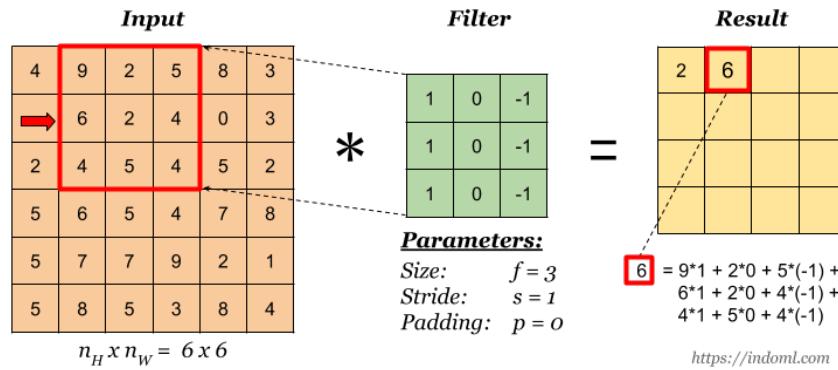


Figura 2.7: Modul de funcționare al operației de convoluție cu un filtru de dimensiune 3 și stride 1 [17]

În discriminator, straturile de convoluție sunt folosite ca într-o rețea de tip CNN³ tradițională, acestea fiind așezate unul după celălalt într-o succesiune crescătoare de dimensiuni. Primele straturi de convoluție învață astfel patternuri simple (low-level features), iar cu cât mergem mai adânc în rețea vom înălți patternuri mai complexe (higher-level features). Acest mecanism este inspirat din felul în care senzațiile vizuale sunt procesate de creier, folosind o ierarhizare a neuronilor: mai multe celule nervoase mai simple sunt legate la un neuron mai complex ce filtrează informații și le transmite mai departe printr-un impuls nervos.

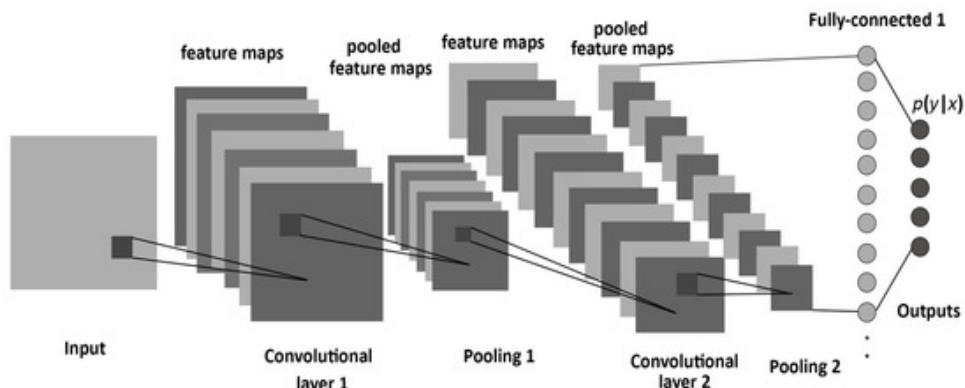


Figura 2.8: Arhitectura unui CNN [24]

³CNN - convolutional neural network sau rețea neurală convoluțională

Arhitectura generatorului folosește operația de conoluție în trei moduri diferite:

- Downsampling: o operație de reducere a dimensionalității imaginii; prin straturi de conoluție cu stride > 1 , rețeaua este capabilă să extragă caracteristici abstrakte din datele de antrenare și să învețe structura și stilul general;
- Straturi reziduale (inspirate de arhitectura ResNet): aici straturile de conoluție sunt conectate prin skip connections, care permit modelului să se concentreze pe învățarea modificărilor specifice stilului, în loc să refacă întreaga imagine de la zero;
- Upsampling: o operație de creștere a rezoluției spațiale a imaginii; prin straturi de deconoluție⁴, rețeaua poate mări feature mapurile și genera o imagine nouă, de dimensiuni egale cu cele ale imaginii de intrare.

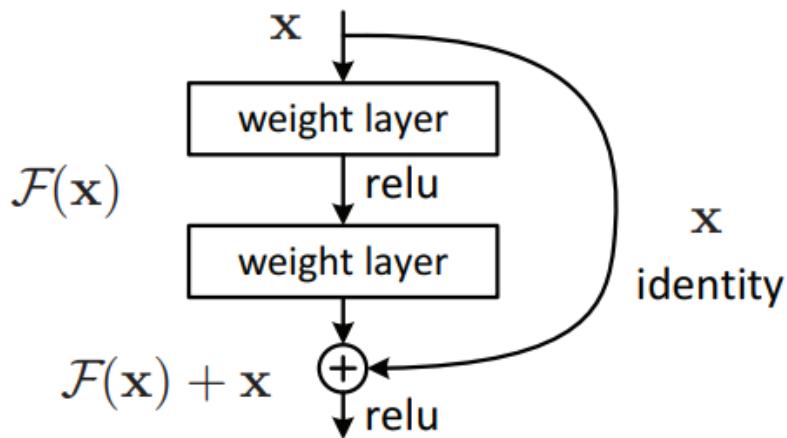


Figura 2.9: Reprezentare a unui bloc rezidual cu skip connections în ResNet [23]

În cazul generatorului, straturile cu parametrii învățabili din figura 2.9 sunt tot straturi de conoluție. Conexiunea de tip skip este reprezentată prin adunarea inputului x , ce intră în stratul rezidual, cu outputul obținut după cele două straturi de conoluție, notat cu $\mathcal{F}(x)$ în figura 2.9.

⁴Deconoluția este operația inversă conoluției și este folosit pentru amplificarea detaliilor și reconstituirea unei rezoluții mai mari a imaginii

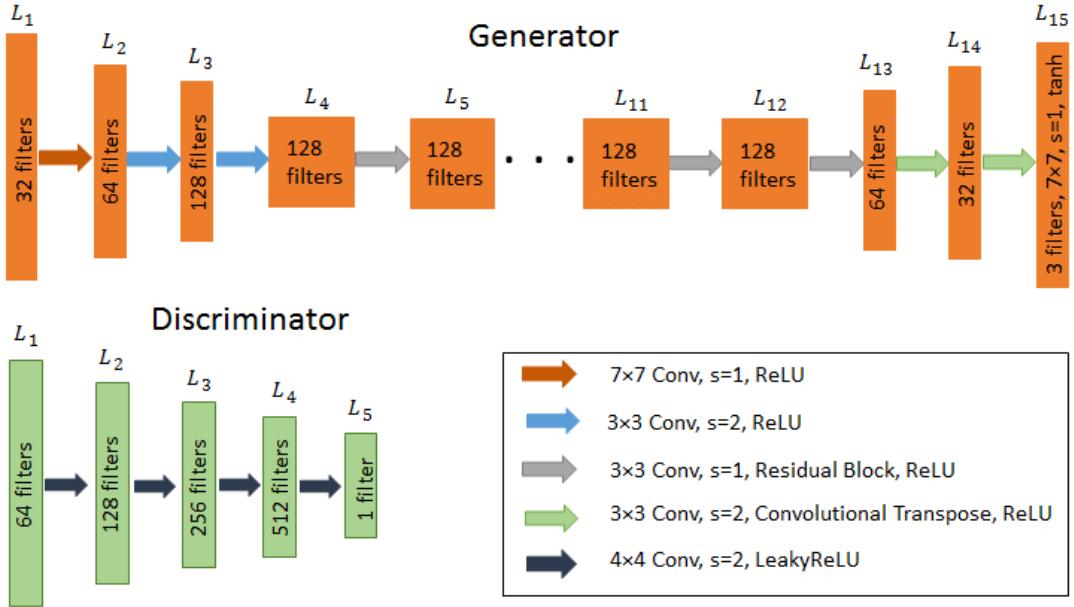


Figura 2.10: Un exemplu de arhitectură CycleGAN [18]

2.3.4 Avantaje și dezavantaje

Rețelele CycleGAN obțin rezultate foarte bune pe partea de transfer de stil, fiind unul dintre algoritmii state-of-the-art pe acest domeniu. Un mare avantaj al acestei rețele este folosirea învățării nesupervizate. Alți algoritmi ce transferă stilul au adesea nevoie de un set de date format din perechi: dacă a este o imagine din domeniul A și b o imagine din domeniul B , rețeaua va fi antrenată cu perechi (a, b) unde $f(a) = b$. Arhitectura CycleGAN elimină astfel de constrângeri asupra setului de date [31], aspect de o importanță deosebită pentru experimentele noastre, deoarece fără un algoritm de tip unpaired style transfer ar fi trebuit să colaborăm cu artiști care să deseneze fiecare imagine din setul nostru de date în varianta ei anime. Mai mult, modul de antrenare al rețelei și obiectivul de menținere al consistenței într-un ciclu ajută CycleGAN-ul să conveargă mai ușor, fiind mai puțin probabil să întâlnim mode collapse în timpul procesului de antrenare. Pe de altă parte, acest model are și câteva dezavantaje. Procesul de antrenare implică păstrarea concomitentă în memorie a patru rețele diferite, ceea ce face antrenarea unui CycleGAN un proces foarte costisitor din punct de vedere timp și resurse computaționale necesare. De asemnea, algoritmii de tip GAN sunt cunoscuți în comunitatea de cercetători ca fiind greu de antrenat, aceștia fiind adesea foarte sensibili la seturi de date nebalansate și modificări ale hiperparametrilor.

2.3.5 Evaluarea imaginilor generate

În timpul antrenării unui CycleGAN este important ca evoluția imaginilor generate să fie urmărită pentru a ne asigura că procesul de antrenare decurge în parametri normali.

De multe ori, această evaluare a imaginilor se face manual, prin compararea vizuală a imaginilor generate de la o epocă la alta sau între experimente diferite. Cu toate acestea, folosirea unei metriki obiective care să cuantifice evoluția rețelei de la o epocă la alta este esențială. Una dintre cele mai folosite metriki pentru evaluarea imaginilor generate este FID (Fréchet Inception Distance). Această metrică se bazează pe reprezentările latente ale unei rețele neurale de tip Inception, antrenată pe un număr foarte mare de imagini. Atât un set de date de poze reale, din domeniul al cărui stil vrem să îl învățăm, cât și un set de date de poze generate de rețea CycleGAN sunt trecute prin rețea de tip Inception. Pentru vectorii de features rezultați din rețea Inception se calculează media și covariația pentru a afla distribuția feature-urilor fiecărui set de date. Scorul FID se calculează pe baza acestor distribuții, un scor FID mai mic însemnând că cele două distribuții sunt mai apropiate, adică imaginile generate și cele reale ar putea fi confundate de un evaluator uman.

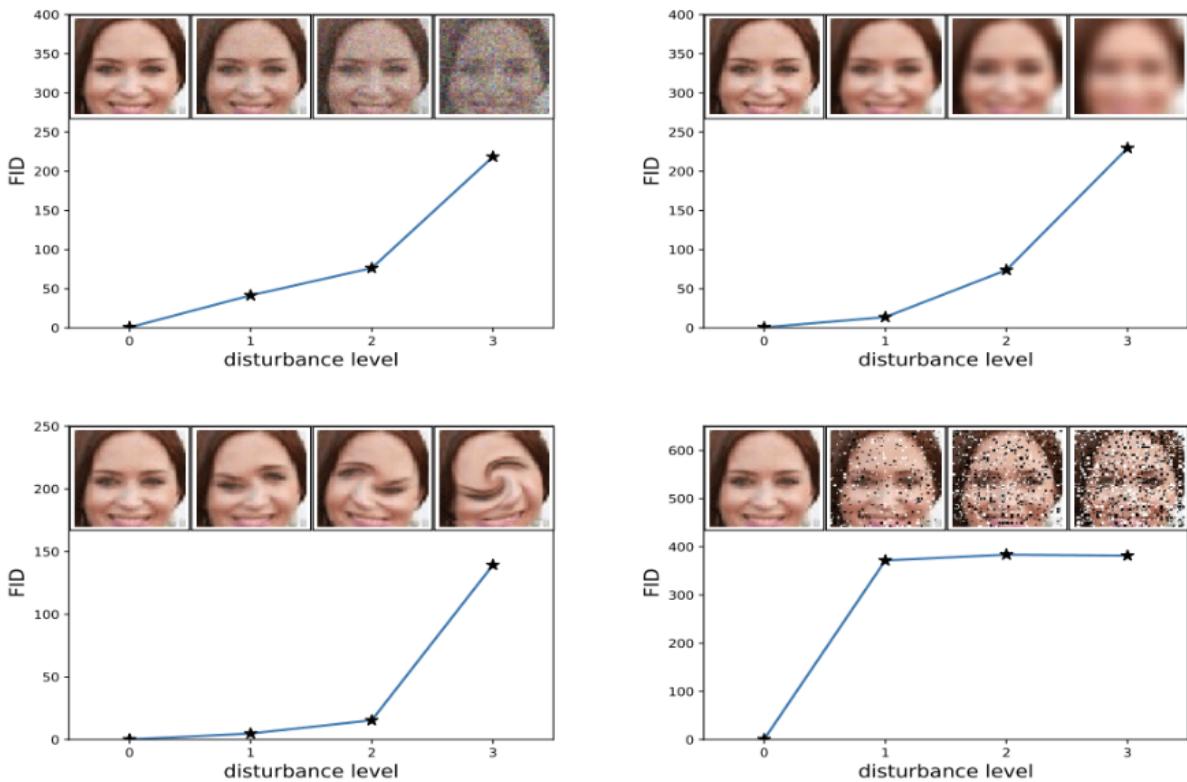


Figura 2.11: Reprezentare vizuală a scorului FID pentru poze generate cu fețe umane [4]

În figura 2.11 se poate observa cum anumite imperfecțiuni ce pot apărea într-o imagine generată afectează valoarea scorului FID. Efectele nedorite precum zgomotul, efectul de blur sau distorsionări ale imaginii duc la un scor FID mai ridicat.

2.4 Segmentarea imaginilor

În domeniul Vederii Artificiale (Computer Vision), task-ul de segmentare al unei imagini reprezintă împărțirea acesteia în regiuni distincte și atribuirea unei etichete fiecărei regiuni delimitate. Spre deosebire de clasificarea imaginilor, care atribuie o singură etichetă întregii imagini, și detectarea obiectelor, care localizează și clasifică obiecte, segmentarea oferă o înțelegere mai detaliată a imaginii prin clasificarea fiecărui pixel individual și delimitarea precisă a limitelor obiectelor. Arhitecturile ce pot rezolva problema segmentării unei imagini au aplicații importante, fiind folosite în imagistica medicală și conducerea autonomă a mașinilor. Arhitectura folosită în experimentele noastre este cea de tip SegFormer, care folosește straturi de transformeri pentru a eticheta fiecare pixel dintr-o imagine.

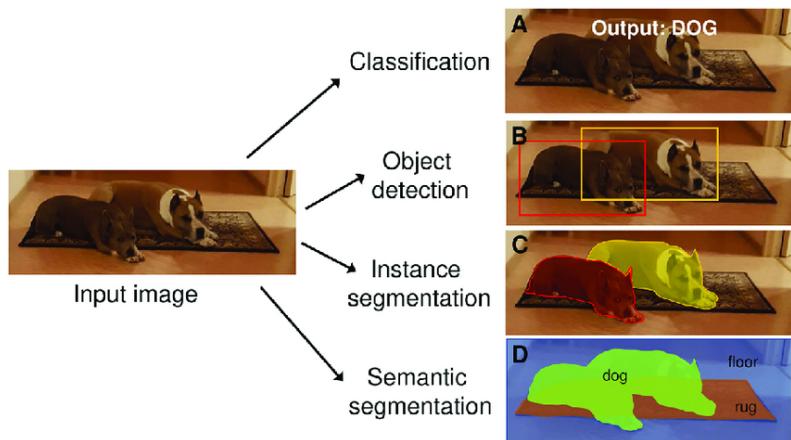


Figura 2.12: Diferențele dintre task-ul de clasificare al imaginii, detecția de obiecte și segmentare [10]

2.4.1 Transformeri

Transformerul este o arhitectură introdusă în anul 2017, în lucrarea "Attention Is All You Need" de către Ashish Vaswani. Acest nou model a reprezentat o inovație în domeniul NLP [6]⁵ prin introducerea unei noi operații cheie - atenția. Arhitectura transformerului este compusă dintr-un encoder și un decoder, ambele componente folosind operația de atenție. Encoderul este format din mai multe blocuri identice de codare, fiecare bloc conținând un strat de atenție multi-head urmat de un strat complet-conectat (feed-forward). Rolul encoderului este de a crea o reprezentare latentă (cunoscută și sub numele de vector de context) a datelor de intrare, ce poate reda informații relevante. Pe de altă parte decoderul este alcătuit din mai multe blocuri identice de decodare, care primesc vectorul de context de la encoder și încearcă să genereze o secvență de text nouă, reprezentând outputul rețelei. Această secvență de text poate fi, de exemplu, traducerea unui text într-o

⁵NLP - Natural Language Processing este domeniul care se ocupă cu modelele de ML care operează pe texte.

altă limbă sau răspunsul la o întrebare primită ca input. Blocurile din decoder conțin trei straturi principale: două straturi de atenție (unul pentru etichetele datelor de intrare și celălalt pentru vectorul de context primit de la encoder) și un strat complet-conectat (feed-forward) [28].

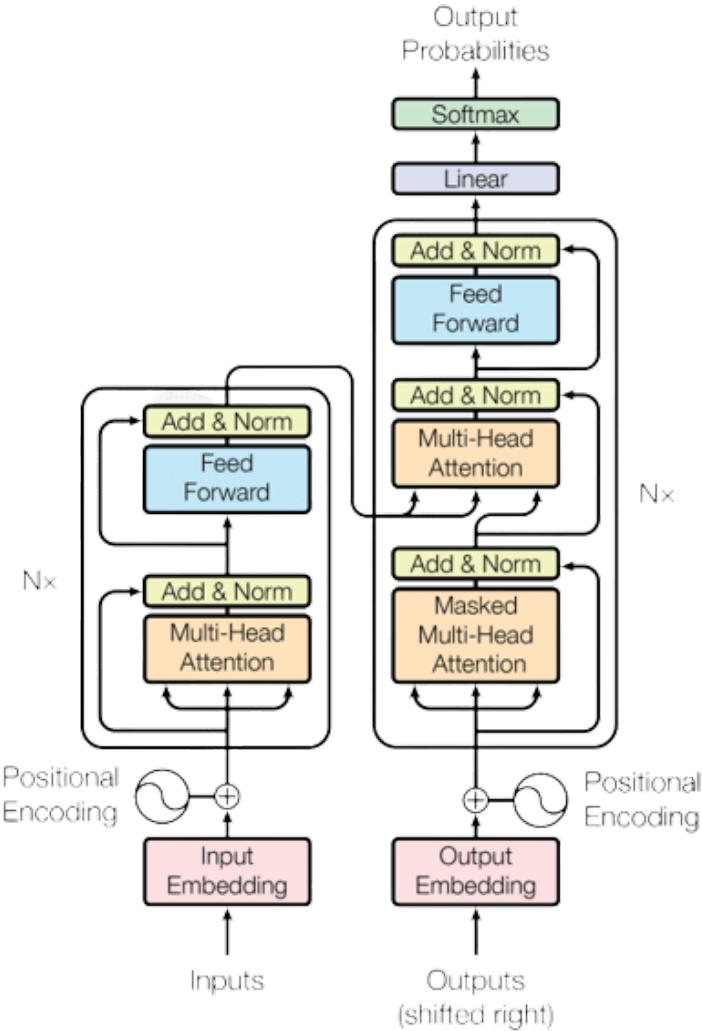


Figura 2.13: Arhitectura unui transformer format dintr-un encoder (blocul din stânga) și un decoder (blocul din dreapta) [28]

2.4.2 Atenția

Straturile de atenție calculează o medie ponderată a valorilor. Aceasta se diferențiază de alte straturi deoarece are nevoie de trei intrări:

- un set de chei (keys) de dimensiune d_k grupate într-o matrice K . Cheile sunt reprezentări (embeddings) ale cuvintelor din propoziția primită ca input;
- o matrice de interogări (query) Q . Fiecare interogare are dimensiunea d_k și reprezintă contextul față de care mecanismul de atenție caută informații (chei) relevante;

- un set de valori (values) de dimensiune d_v , organizate în matricea V. Aceste valori reprezintă informațiile asociate fiecărui cuvânt din propoziția inițială.

Fiecare cuvânt din propoziția primită ca input are asociat un vector cheie k, un vector interogare q și un vector valoare v. Pentru fiecare cuvânt, mecanismul de atenție calculează un scor, înmulțind interogarea cuvântului cu matricea formată din cheile celorlalte cuvinte din propoziție [29]. Aplicând acest algoritm pentru fiecare cuvânt se obține o matrice de scoruri (weight-uri) care se înmulțește cu matricea formată din vectorii valoare a cuvintelor. Vectorii rezultați din înmulțire sunt adunați și se obține rezultatul operației de atenție.

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \quad (2.2)$$

În formula atenției 2.2 de mai sus se poate observa că scorurile obținute au fost împărțite la $\sqrt{d_k}$ și apoi trecute prin funcția softmax. Funcția softmax este folosită pentru normalizarea scorurilor, iar $\frac{1}{\sqrt{d_k}}$ este un factor de scalare ce are rolul de a micșora magnitudinea vectorului de scoruri [28]. Un start de multi-head attention repetă acest proces de mai multe ori, asociind weight-uri ajustabile în timpul antrenării setului de matrice (Q, K, V):

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h) \cdot W^O, \\ \text{unde } \text{head}_i &= \text{Attention}(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V) \end{aligned} \quad (2.3)$$

2.4.3 Transformerul vizual - ViT

Inițial transformerul a fost conceput pentru prelucrarea datelor secvenționale, în special de tip text. Cu toate acestea, datorită rezultatelor din practică ale acestei arhitecturi, s-a încercat adaptarea transformerului pentru a funcționa și pe imagini. Ideea de bază din spatele transformerului vizual (Vision transformer) este transformarea imaginii într-o secvență. O abordare banală asupra acestei probleme ar fi liniarizarea imaginii (transformarea sa dintr-un tensor bidimensional, în cazul imaginilor alb-negru, într-un vector unidimensional). Desigur, această abordare nu este întocmai practică deoarece ar consuma multă memorie, iar un pixel individual dintr-o imagine nu oferă așa de multe informații despre conținutul acesteia. Abordarea oferită de transformerul vizual [7] constă în spargerea imaginii inițiale în patch-uri (imagini mai mici). Astfel, un patch va fi tratat ca un cuvânt dintr-o frază (imaginea întreagă). Fiecare patch este liniarizat și codificat printr-un embedding care să păstreze cât mai multe informații. La această codificare se adaugă niște embeddinguri poziționale, menite să păstreze informații asupra poziției patch-ului în imaginea inițială, iar apoi codificarea obținută este pasată rețelei pentru învățare.

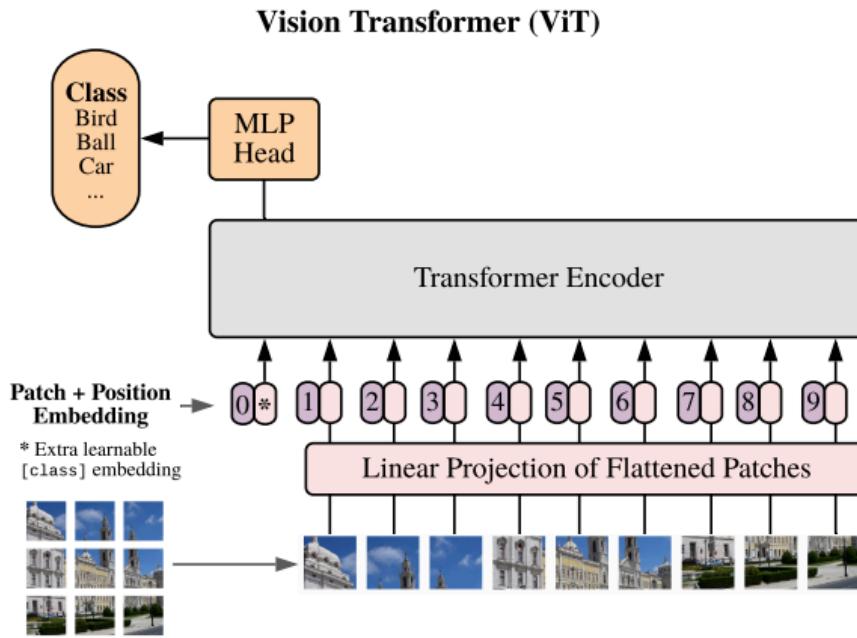


Figura 2.14: Vizualizare a descompunerii imaginii de intrare în patch-uri și arhitectura unui ViT [7]

2.4.4 SegFormerul

Arhitectura SegFormer a fost propusă în anul 2017 în lucrarea "SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers" de către Enze Xie și Wenhai Wang și a reprezentat o inovație în domeniul segmentării, acest model, deși mult mai mic ca număr de parametri față de alte arhitecturi pentru segmentare, reușește să obțină rezultate mai bune decât acestea [30].

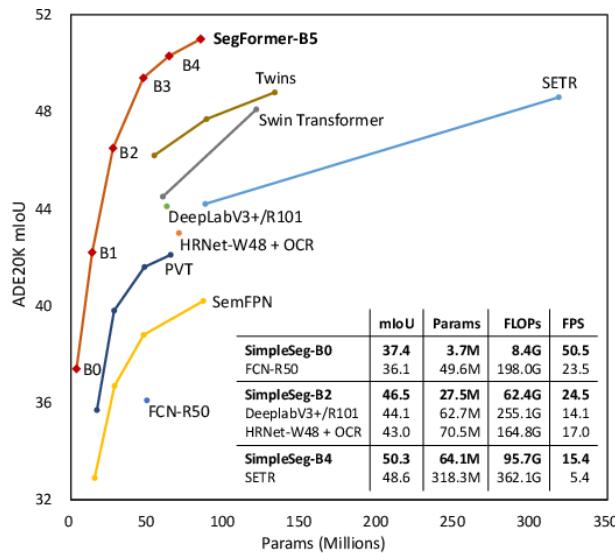


Figura 2.15: Comparația performanței SegFormerului cu alte arhitecturi [30]

În figura 2.15 este ilustrată performanța SegFormerului în paralel cu cea a altor rețele cunoscute. Algoritmul SegFormer are cinci variante (B0-B5) în funcție de dimensiunea arhitecturii. Se poate observa cum varianta B5, care este cea mai adâncă arhitectură de SegFormer, obține rezultate mai bune decât alți algoritmi precum Swin Transformer sau SETR, deși aceste rețele au un număr aproape dublu de parametri față de cei ai SegFormerului [30].

Arhitectura de SegFormer încearcă să redea ierarhizarea feature-urilor specifică unui CNN, dar folosind transformeri. Astfel, SegFormerul este compus la rândul său dintr-un encoder ce folosește ideea de bază a unui Vision transformer, și un decoder.

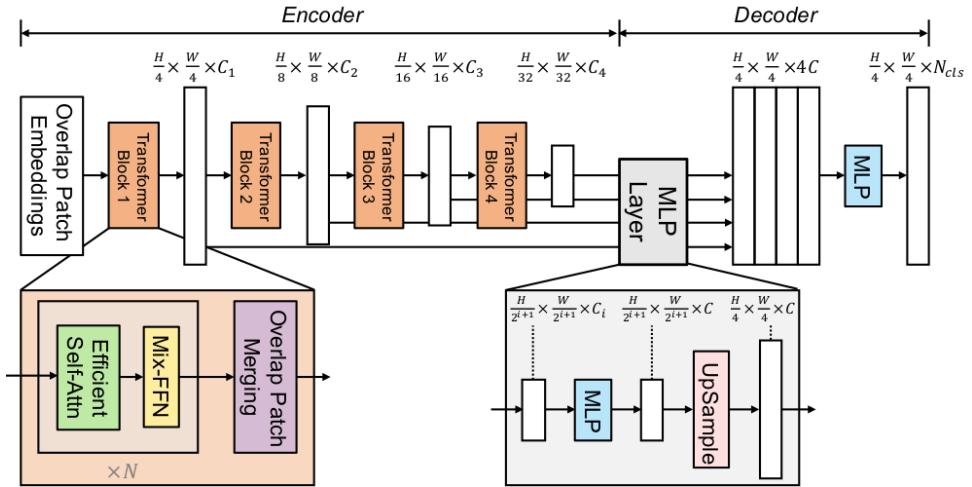


Figura 2.16: vizualizarea arhitecturii SegFormer [30]

În figura 2.16 este ilustrat modul în care SegFormerul folosește patch-uri ce scad gradual în dimensiune $\{\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}\}$. Astfel, encoderul este capabil să învețe atât higher-level features, cât și tipare mai rafinate. În același timp, straturile de transformator sunt combinate cu straturi de conoluție 3×3 care au rolul de a îmbunătăți acuratețea și de a micșora numărul de parametri ai modelului. Decoderul rețelei are o arhitectură cu un număr redus de parametri, acesta constând doar într-o rețea de tip fully connected (MLP) ce oferă un output de forma $\frac{H}{4} \times \frac{W}{4} \times N_{class}$, unde H și W sunt dimensiunile imaginii, iar N_{class} reprezintă numărul claselor [30]. Înainte de a fi procesată de către rețea, imaginea initială este spartă în patch-uri ce se suprapun, iar acestea sunt trecute prin câteva straturi de conoluție pentru a obține niște embeddinguri care extrag eficient feature-urile din subimagini.

Capitolul 3

Abordări recente

În continuare vor fi prezentate o serie de lucrări pe teme similare cu cea propusă. Aceste studii sunt importante deoarece prezintă diverse idei ce au fost folosite în experimentele noastre. De asemenea, rezultatele obținute folosind abordarea propusă în acest studiu pot fi evaluate prin comparația rezultatelor produse de următoarele lucrări.

3.1 Transferul stilului unor pictori celebri

Una dintre aplicațiile propuse în lucrarea ce introduce arhitectura CycleGAN este transferul stilului unor pictori celebri pentru poze cu peisaje. Această aplicație este importantă deoarece a reprezentat una dintre sursele de inspirație în alegerea temei și deoarece autorii introduc un nou obiectiv în antrenare CycleGAN-ului, și anume o funcție de loss de identitate (identity loss). Acest obiectiv a fost adăugat inițial atunci când s-a încercat maparea inversă, de la pictură la poză realistă [31].

$$\mathcal{L}_{\text{identity}}(F, G) = \mathbb{E}_{b \sim p_B} [\|kF(b) - b\|_1] + \mathbb{E}_{a \sim p_A} [\|kG(a) - a\|_1] \quad (3.1)$$

Acest nou obiectiv se asigură ca atunci când generatorul F primește deja o poză ce aparține domeniului țintă B, distanța dintre b și f(b) să fie minimă. Într-adevăr, dacă generatorul primește deja o imagine din domeniul B, acesta nu mai e nevoie să facă nicio transformare asupra imaginii. Adăugarea acestei funcții de loss la obiectivul original sporește calitatea imaginilor generate, atât în cazul experimentului din această lucrare, cât și în cazul experimentelor noastre. Tot aici, sunt evidențiate câteva sugestii pentru îmbunătățirea procesului de învățare precum crearea unui istoric de poze generate pentru antrenarea discriminatorului și înlocuirea funcției logaritm din funcția de loss a GAN-ului clasic cu distanță L2.



Figura 3.1: Câteva dintre rezultatele obținute de autorii acestei lucrări. Pe prima coloană se află imaginea de input, iar în coloanele următoare sunt imaginile generate în stilul unor anumiți pictori celebri. [31]

3.2 Transferul stilului anime

Există mai multe lucrări care abordează problema transferului de stil anime. În această secțiune vom aminti patru studii relevante.

3.2.1 Arhitectura clasică

Xin Chen și Gang Liu abordează tema transferului de stil anime pe diverse poze în lucrarea "AnimeGANv2". Aceștia folosesc o arhitectură similară cu cea clasică, dar mai adâncă, generatorul fiind compus tot din straturi de downsampling, straturi reziduale și straturi de upsampling. De asemenea, autorii au înlocuit normalizarea la nivel de instanță (instance normalization) cu normalizarea la nivel de strat (layer normalization) pentru a obține rezultate mai bune [13].



Figura 3.2: Rezultate ale rețelei AnimeGANv2 [13]

În figura 3.2 se poate observa cum rețeaia reușește să aplique cu succes stilul pe poze cu peisaje (linia 2), dar nu reușește să redea câteva din caracteristicile specifice personajelor anime, atunci când se folosesc imagini cu persoane (linia 1).

Un alt studiu ce se focusează de data aceasta pe generarea de poze cu persoane este "Generating Anime Faces From Human Faces With Adversarial Networks". Aici, autorii

Yu-Jing Lin și Chiou-Shann Fuh folosesc arhitectura clasică de CycleGAN pentru a transfera stilul anime pe poze cu fețe. Mai mult, aceștia încearcă atât arhitectura bazată pe ResNet a generatorului (ca în arhitectura clasică), cât și una bazată pe UNet. Cu toate acestea, cele mai bune rezultate au fost generate de generatorul de tip ResNet [14], cel UNet ajungând adesea la mode collapse în experimentele lor.



Figura 3.3: Rezultate obținute în [14], folosind un generator bazat pe ResNet

3.2.2 Arhitecturi noi

Pentru a transpune mai bine stilul anime pentru pozele cu persoane, dar și pentru a obține imagini de o calitate mai înaltă, fără artefacte, mai mulți cercetători au încercat să schimbe arhitectura de bază a CycleGAN-ului.

Spre exemplu, în lucrarea ”AttentionGAN: Unpaired Image-to-Image Translation using Attention-Guided Generative Adversarial Networks”, autorii introduc atenția atât în arhitectura generatorului, cât și în cea a discriminatorului. Atenția este folosită pentru a genera atât o mască de conținut (ce conține imaginea transformată în stilul anime), cât și o mască de atenție (cu zone importante la care generatorul trebuie să producă mai multe detalii, precum ochii, gura, părul). Cele două măști sunt combinate pentru a obține un rezultat final fără artefacte, ce reușește să redea trăsăturile reconoscibile ale personajelor anime [26].



Figura 3.4: Rezultate obținute folosind AttentionGAN. Pe prima linie se află imaginile de input, pe a doua măștile de atenție și pe ultima imaginile rezultat [26]

O altă lucrare, ”UVCGAN: UNet Vision Transformer cycle-consistent GAN for unpaired image-to-image translation” propune o arhitectură mai complexă a generatorului, bazată pe UNet-ViT (o arhitectură ce combină rețea UNet cu un Pixelwise ViT). Mai mult, aceștia recomandă preantrenarea generatoarelor pe un task supervizat de colorare [27]. Imaginele anime sunt sparte în patch-uri, iar valorile pixelilor din unele patch-uri sunt înlocuite cu 0, generatorul trebuind să refacă imaginea inițială.

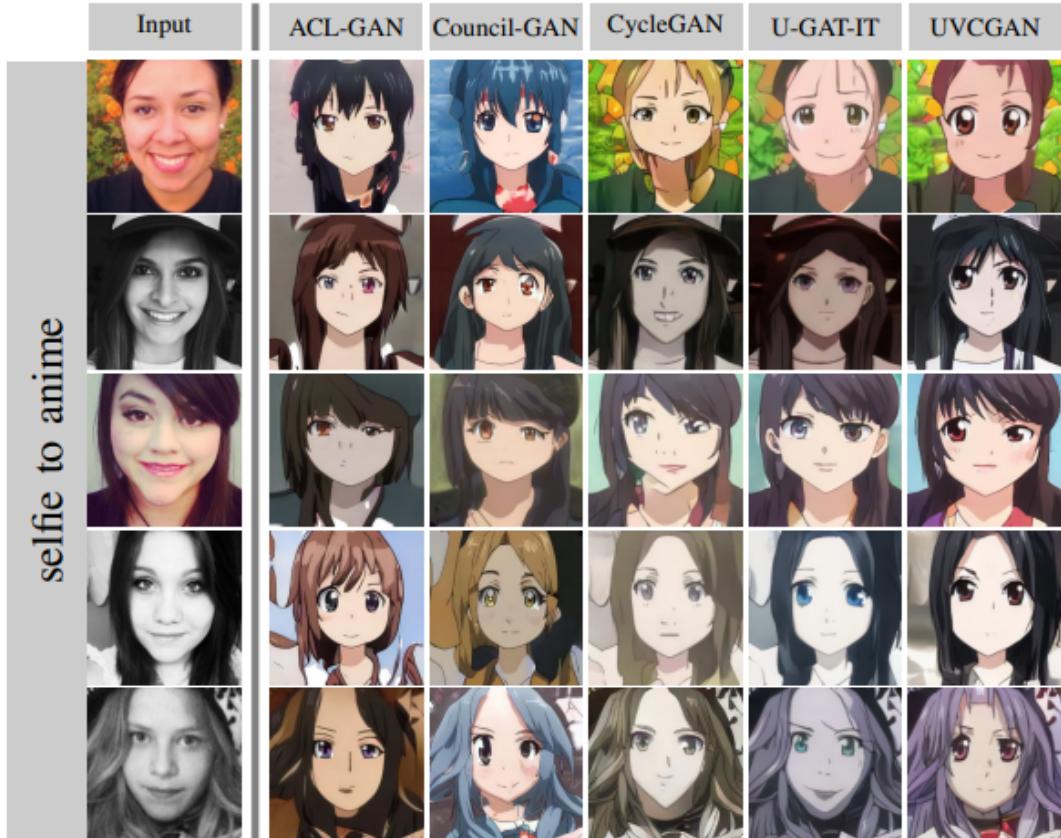


Figura 3.5: O comparație între rezultatele diferitelor variante de GAN, făcută de autorii arhitecturii UVCGAN [27]

În figura 3.5 se poate observa avantajul preantrenării pe un task de colorare. Atunci când imaginea de input este alb-negru, UVCGAN (coloana 6) reușește să aducă mai multe culori în imaginea generată față de o arhitectură CycleGAN obișnuită (coloana 4). În același timp, rezultatele rețelei UVCGAN au o structură a feței mult mai armonioasă și redau mai multe detalii față de celelalte arhitecturi.

Capitolul 4

Tehnologii folosite

În domeniul învățării automate, limbajul cel mai des utilizat este Python [3]. Într-adevăr, sintaxa limbajului Python este concisă și ușor de înțeles, permitând construirea rapidă atât a modelelor, cât și a unor aplicații sau site-uri web ce pot infera algoritmii de ML. În plus, limbajul Python vine cu un întreg ecosistem de librării (framework-uri) puternice, care oferă implementări eficiente ale algoritmilor folosiți în procesul de dezvoltare. Limbajul Python a fost folosit și în cadrul acestui studiu, atât pentru partea de construire și antrenare a rețelelor, cât și pentru realizarea unei interfețe grafice care să permită inferența modelelor antrenate și pentru utilizatorii cu mai puține cunoștințe tehnice. S-au folosit diverse framework-uri pentru următoarele:

- design-ul arhitecturilor și antrenarea rețelelor: PyTorch, HuggingFace;
- manipularea tensorilor: NumPy, PyTorch;
- prelucrarea imaginilor: OpenCV, PIL;
- vizualizarea graficelor: Matplotlib;
- crearea interfeței grafice (GUI): Tkinter.

Librăria PyTorch este una dintre cele mai folosite din domeniul învățării automate. Această librărie oferă implementări ale anumitor straturi (precum straturi convoluționale sau transformer) sub formă de clase (în modulul nn) ceea ce face foarte ușoară dezvoltarea și customizarea unei arhitecturi mai complexe care folosește diverse tipuri de straturi uzuale. Mai mult, biblioteca realizează eficient algoritmul de backpropagation și actualizare a weight-urilor din timpul antrenării prin construirea automată a grafurilor computaționale. Un avantaj al acestui framework, spre deosebire de alte librării similare precum TensorFlow, este construirea dinamică a grafului computațional care împreună cu organizarea straturilor în clase oferă mai multă expresivitate și flexibilitate programatorului în procesul de construire, ușurând în același timp și depanarea arhitecturilor. În ultimul rând,

PyTorch oferă și suport pentru calculul eficient pe GPU, fapt ce poate spori semnificativ timpul de antrenare al unei rețele adânci [2].

HuggingFace este atât o bibliotecă ce oferă suport pentru modelele de ML ca Pytorch, cât și o platformă open-source, specializată în prelucrarea limbajului natural (NLP). Am ales să folosim și această tehnologie pentru a avea acces la modele pre-antrenate pe un număr mare de date și la anumite seturi de date. HuggingFace facilitează procesul de fine-tuning al modelelor pre-antrenate prin intermediul unei clase speciale numite Trainer, care reușește să construiască un întreg proces de învățare folosind foarte puțin cod. Modelele rafinate pot fi ulterior încărcate pe site-ul celor de la HuggingFace într-un repository pentru a fi folosite foarte ușor de întreaga comunitate.

Toate experimentele au fost făcute în Jupyter Notebook, o aplicație web open-source care oferă posibilitatea de a scrie modular, codul fiind structurat în celule ce pot fi rulate separat. Acest lucru este foarte avantajos în cadrul experimentelor, deoarece oferă un grad mare de interactivitate la fiecare pas al procesului de dezvoltare (fie că vorbim de încărcarea și preprocesarea datelor, antrenare modelelor sau vizualizarea rezultatelor).

Pentru interfață grafică, s-a folosit Tkinter deoarece acest framework este simplu și ușor de învățat și poate fi folosit pentru construirea de aplicații rapide, ce pot rula pe orice sistem de operare. Tkinter oferă suport pentru desenarea de forme geometrice și așezarea lor armonioasă pe ecran, precum și anumite elemente specifice unei interfețe grafice precum butoane sau formulare.

Nu în ultimul rând, am folosit GitHub¹, respectiv Git, pentru păstrarea și organizarea fișierelor din întregul proiect.

¹GitHub este o platformă populară pentru gestionarea codului sursă, colaborarea în echipă și dezvoltarea proiectelor software.

Capitolul 5

Abordarea propusă

Vom rezolva problema transferului de stil localizat pe fețele persoanelor printr-un ansamblu de rețele. Prima rețea va fi de tip SegFormer și va fi antrenată pe un task de segmentare pentru a delimita zonele din poză corespunzătoare feței și părului unei persoane. Pe baza acestei segmentări, imaginea va fi decupată pentru a conține doar fața persoanei respective, urmând ca mai apoi să fie folosită ca input pentru un CycleGAN care să transforme persoana într-un personaj anime. Rezultatul CycleGAN-ului va fi apoi "lipit" pe poza inițială, creând efectul de colaj dorit.

5.1 Transferul stilui anime

Pentru task-ul de transfer al stilului a fost antrenată o rețea CycleGAN de la zero. Inițial s-a plecat de la arhitectura clasică, urmând toate indicațiile de implementare din [31], însă pentru a obține rezultate mai bune au fost încercate mai multe idei proprii sau inspirate din celelalte studii menționate în capitolul anterior.

5.1.1 Crearea unui set de date

Un mare avantaj al CycleGAN-ului este faptul că acesta nu necesită perechi de date din cele două domenii, fapt ce facilitează mult procesul de strângere al datelor de antrenare. Așadar, pentru a genera pozele dorite este suficient să strângem un set de date cu persoane și unul cu personaje anime. Pentru pozele cu personaje anime, s-au căutat diverse seturi de date pe internet care să se plieze pe problema propusă. În experimentele noastre au fost folosite următoarele seturi de date: [High-Resolution Anime Face Dataset \(512x512\)](#), [Selfie2anime](#) și [Anime Face Dataset](#).

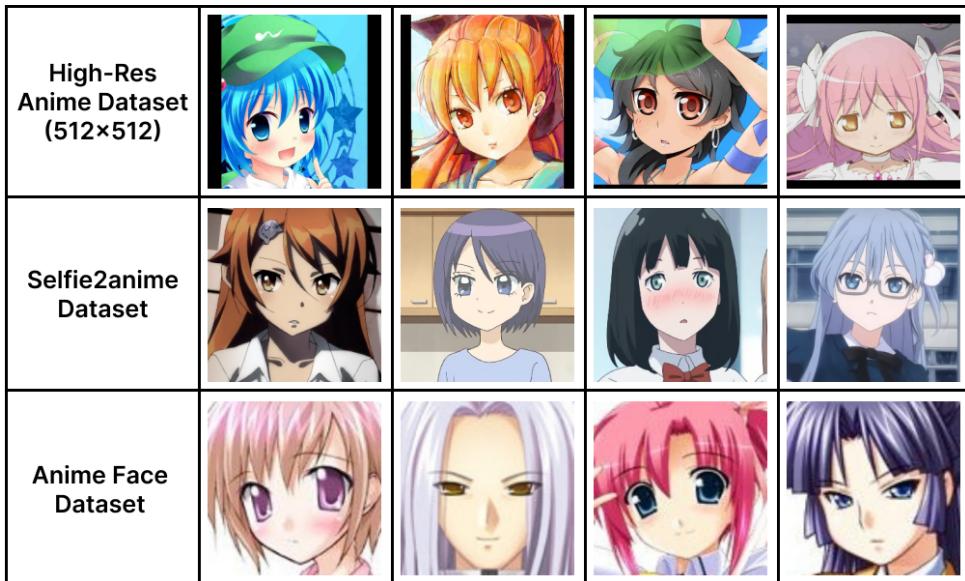


Figura 5.1: Exemple de imagini din seturile de date cu personaje anime

Din aceste seturi de date mari au fost extrase imagini și reorganizate în următoarele subseturi:

- **Anime6k** - acest set de date conține 6000 de poze de antrenare extrase din High-Resolution Anime Face Dataset (512x512). Setul de date din care au fost preluate imaginile era foarte divers, conținând atât imagini din serialele anime, cât și din manga. Pentru setul de date Anime6k au fost selectate manual doar imaginile color care conțineau un singur personaj anime;
- **Selfie2anime** - setul de date de aproximativ 3400 de poze anime folosit de [27];
- **Anime10k** - un set de date format din aproximativ 6000 imagini luate din Anime6k, 3000 de imagini din Selfie2Anime și 1000 de imagini luate din Anime Face Dataset.

Pozele cu persoane au fost preluate din [AISegment.com - Matting Human Datasets](#). Acest set de date a fost ales datorită diversității sale (conținând poze cu persoane de diferite vârste în diverse ipostaze), dar și pentru că oferea pentru fiecare poză și varianta în care tot fundalul este transparent, putând fi folosite și la segmentare. Înainte de a le folosi pentru antrenare, pozele au fost trecute printr-un detector facial și au fost decupate pentru a aduce față în prim plan.



Figura 5.2: Exemple din setul de date cu persoane, după ce imaginile au fost cropate folosind un detector facial

Pentru fiecare experiment s-a folosit un număr egal de poze cu personaje anime și poze cu persoane reale, iar pentru încărcarea eficientă a datelor în timpul antrenării am creat o clasă specială MyDataset. Această clasă suprascrie operatorul de indexare și este folosită de DataLoader (o clasă din PyTorch) pentru încărcarea datelor în memorie și împachetarea lor într-un batch. Eficiența provine din faptul că doar batch-ul (grupul de poze) curent care este procesat de rețea este încărcat în memorie, ci nu tot setul de date.

5.1.2 Preprocesarea imaginilor

Folosind modulul transforms din Pytorch am creat un lanț de operații (pipeline) de preprocesare ce vor fi aplicate pe fiecare batch de către DataLoader. Provenind din seturi de date diferite, era important ca imaginile să fie aduse la aceeași dimensiune (în experimente s-a folosit 256x256 de pixeli). Mai mult, pentru ajutarea procesului de învățare și evitarea problemelor precum overfitul sau mode collapse, datele au fost standardizate și s-au aplicat următoarele tehnici de augmentare a datelor: oglindirea orizontală a imaginilor cu o probabilitate de 50% [31], decuparea aleatoare a imaginilor (acestea au fost redimensionate la 290x290 și după tăiate la 256x256) [27] și rotirea imaginilor cu un unghi de maxim 10°. Un alt pas foarte important din preprocesare a fost aplicarea unui filtru de blur ce scade în intensitate gradual, permitând modelului să se focuseze pe niște feature-uri low-level (precum culorile) la începutul învățării, urmând ca mai apoi să învețe detalii precum modul de a desena ochii în stil anime. S-a plecat de la un filtru de blur (Gaussian Blur) cu un kernel de dimensiune 13 și un factor de blur de 40, urmând ca aceste valori să scadă gradual pe parcursul a 150 de epoci (din totalul de 200 de epoci dintr-un experiment).

5.1.3 Arhitectura modelelor

Arhitectura rețelelor este foarte asemănătoare cu cea clasică, atât în cazul generatorului, cât și în cazul discriminatorului. Diferențele apar prin înlocuirea straturilor de normalizare de instanță cu straturi de normalizare pe batch. Deși în [31] este sugerată folosirea normalizării de instanță, în experimentele noastre am remarcat obținerea unor rezultate mai bune folosind al doilea tip de normalizare, asemănător cu [14], dar și o îmbunătățire de 10-15% în ceea ce privește timpul de antrenare. Am experimentat cu batch-uri de 2, 4 și 8 imagini, dar cele mai bune rezultate au fost obținute folosind batch-uri de 4. În același timp, la nivelul discriminatorului am înlăturat funcția sigmoidă de pe ultimul strat al rețelei. Acest lucru este folositor în antrenare deoarece în loc să avem o etichetă globală pe toată imaginea (real sau generat), vor fi etichetate patch-uri din imaginea generată, oferind mai multe informații din care generatorul poate învăța.

Pentru generator s-au creat straturile ConvBlock (cu operații de conoluție) și ResBlock (blocuri reziduale cu legături skip), iar pentru discriminator am creat clasa DiscBlock ce

conține la rândul ei straturi de convoluție.

Numele stratului	Componente din modelul nn
ConvBlock	ReflectionPad2d, Conv2d sau ConvTranspose2d, BatchNorm2d, ReLU
ResBlock	2 straturi de tip ConvBlock, ultimul strat fără funcția de activare ReLU
DiscBlock	ConvBlock, BatchNorm2d, LeakyReLU

Tabela 5.1: Straturile folosite

În descrierea generatorului și a discriminatorului, notația $NumeBloc(x, y)$ va reprezenta numărul x de features care intră în stratul de convoluție din bloc, iar y numărul de features de ieșire. Generatorul este format din:

- strat de intrare, un ConvBlock(3, 64) cu kernel=7, padding=3, stride=1;
- 2 straturi de downsampling de tip ConvBlock($64 * 2^{i-1}$, $64 * 2^i$), unde i este numărul stratului, cu kernel=3, padding=1, stride=2;
- 9 straturi ResBlock(256, 256), cu kernel=3, stride=1, padding=1 pentru ambele straturi din convoluție ce fac parte din ResBlock;
- 2 straturi de upsampling ConvBlock($256 / 2^{i-1}$, $256 / 2^i$), ce folosesc deconvoluție cu kernel=3, stride=2, output_padding=1;
- stratul de ieșire, format dintr-un strat Conv2d(64, 3) cu kernel=7, stride=1, padding=3 și funcția de activare nn.Tanh.

Arhitectura discriminatorului folosește doar convoluții de kernel=4 și padding=1. Aceasta este alcătuită din:

- stratul de intrare format dintr-un DiscBlock(3, 64) cu stride=2;
- straturi ascunse, 3 blocuri DiscBlock($64 * 2^{i-1}$, $64 * 2^i$), cu stride=1 pentru ultimul strat ascuns și stride=2 pentru celelalte;
- stratul de ieșire, un Conv2d(512, 1) cu stride=1.

În experimentele noastre s-a încercat și adăugarea unui strat de nn.Dropout(0.5) între cele două straturi de convoluții dintr-un ResBlock pentru a evita overfitul, însă pozele generate astfel erau mult prea puțin modificate.

5.1.4 Modul de antrenare

Pentru antrenarea rețelei s-a construit clasa CycleGan, care se ocupă de gestionarea celor patru rețele (două generatoare și două discriminatoare). Parametrii unui experiment (precum learning rate-ul sau coeficienții funcțiilor de loss λ_{GAN} , λ_{cyc} , λ_{id}) sunt citiți dintr-un fișier de tip json de către clasa CycleGan, aşadar pentru a rula experimente diferite este necesară doar modificarea fișierului options.json. În același timp, această clasă se ocupă și de inițializarea parametrilor rețelelor, salvarea rezultatelor intermediare, a loss-ului pentru fiecare epocă, precum și weight-urile corespunzătoare ultimei epoci și a celei mai bune epoci. Deoarece procesul de antrenare durează foarte mult ($>48h$), a fost necesară implementarea unui mecanism de reluare a antrenării de la ultima epocă salvată. Pentru aceasta s-au folosit dicționarele de stare (state_dict) din PyTorch care sunt salvate tot prin intermediul clasei CycleGan.

Modelul se antrenează timp de 200 de epoci, primele 100 epoci având un learning rate mai mare, de 0.0002 (epocile de "încălzire" în care generatorul este lăsat să experimenteze mai mult), urmând ca valoarea coeficientului de învățare să descrească gradual pe parcursul celorlalte 100 de epoci (epocile de stabilizare). Din experimentele noastre, valoarea optimă pentru coeficientul de învățare este 0.0002, încercările cu un learning rate mai mare având probleme în a converge, iar folosirea unui learning rate mai mic duce adesea la underfit. Pentru optimizarea gradienților s-a folosit algoritmul Adam, iar loss-urile de identitate și consistență au fost calculate folosind distanța L1, respectiv L2 pentru loss-ul GAN-ului și al discriminatorului, conform [31]. Pentru calcularea loss-ului generatorului se face media aritmetică între loss-urile celor două generatoare pe fiecare obiectiv, urmând ca cele trei valori de loss obținute să fie adunate folosind anumite ponderi λ , ce sunt descrise în tabelul 5.2. Analog, loss-ul total al discriminatorului este calculat ca media aritmetică între loss-urile celor două discriminatoare.

Denumire	λ_{GAN}	λ_{cyc}	λ_{id}
Valorile implicate [31]	1	10	5
Consistență mică	1	1	1
Consistență medie	1	5	2.5
Consistență mare	1	14	7

Tabela 5.2: Combinățiile de coeficienți folosite în experimente inspirate de [27]

La începutul antrenării, toți parametrii rețelei sunt inițializați cu o inițializare Xavier (dacă nu se reia procesul de antrenare) și se setează seedul funcțiilor ce folosesc numere generate aleator pentru reproductibilitatea experimentelor. În timpul antrenării, discriminatorul este cel care determină generatorul să se perfecționeze. Pentru a spori performanța discriminatorului (și ulterior și pe cea a generatorului), acesta este evaluat

la finalul unei epoci, nu doar pe ultima poză generată, ci pe un set de 50 de poze generate anterior. Clasa ImagePool este cea care se ocupă de păstrarea acestui istoric de dimensiune fixă, de imagini generate (atunci când o nouă imagine este adăugată, dacă numărul total de poze este >50 , se va elimina o poză în mod aleator).

5.1.5 Evaluarea modelului

Este important ca în timpul antrenării să fie urmărite rezultatele intermediare de la finalul fiecărei epoci. Epocile de încălzire au un impact crucial asupra felului în care va performa generatorul, de aceea dacă acesta nu reușește să reproducă anumite elemente specifice stilului anime după 70-80 de epoci, experimentul poate fi considerat eşuat și ar fi mai bine ca rețeaua să fie reantrenată folosind alți hiperparametri. Pe lângă examinarea vizuală care poate fi adesea subiectivă, pentru compararea rezultatelor între experimente s-a folosit și scorul FID. Aceasta este calculat pentru un set de 1000 de poze de validare la finalul fiecărei epoci, salvându-se mereu epoca cu scorul cel mai mic. Acest mecanism s-a dovedit a fi extrem de benefic în practică deoarece în procesul de antrenare sunt momente în care generatorul este mai puternic decât discriminatorul (aici se generează cele mai bune rezultate) și momente în care discriminatorul este mai puternic (generatorul încercând să se recalibreze, produce niște rezultate mult mai slabe calitativ). De aceea este important să oprim procesul de antrenare la epoca cu scorul FID cel mai mic, ci nu neapărat la ultima epocă din cele 200 (analog, dacă FID-ul nu se îmbunătățește în decurs de 50-60 de epoci, procesul de antrenare nu decurge cum trebuie - experiment eşuat).

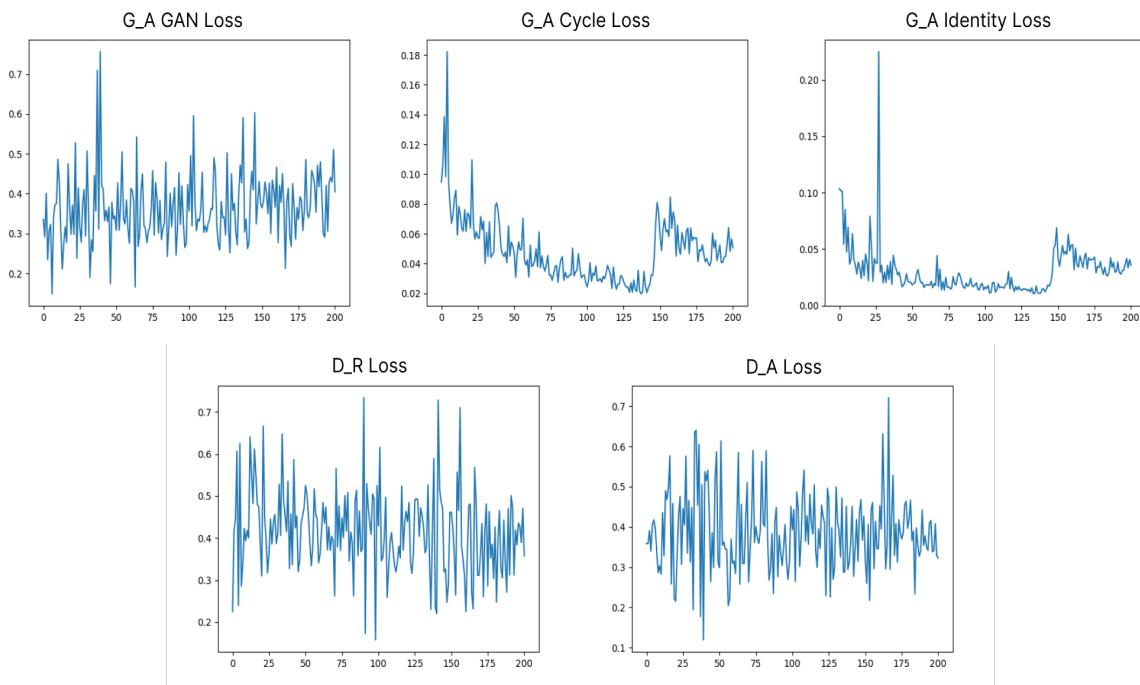


Figura 5.3: Evoluția valorii loss-ului de-a lungul epocilor pentru generatorul de anime și cele două discriminatoare

Pe lângă scorul FID este important să urmărim și evoluția loss-ului rețelei. În figura 5.3 sunt ilustrate graficele funcției de loss pentru fiecare obiectiv. Se observă că acestea nu seamănă cu graficul de loss ideal, prezentat în capitolul de concepte teoretice, însă acest lucru este normal din cauza modului de învățare adversarial. Cu toate acestea, valorile loss-ului nu trebuie să fluctueze foarte mult în timpul antrenării (gradienți explozivi) dacă procesul de antrenare decurge bine. Spre finalul antrenării, ultimele 30-40 de epoci, este important ca valoarea loss-ului să se stabilizeze. Dacă această valoare fluctuează prea mult, sau valoarea loss-ului este prea mare, atunci înseamnă că procesul de învățare poate continua (putem încerca să antrenăm mai mult de 200 de epoci) sau putem considera experimentul eşuat, generatorul nefiind stabil. Pe lângă abordarea descreșterii coeficientului de învățare după 100 de epoci, propusă în [31], s-a încercat și o strategie de tip "decay on plateau" (scăderea loss-ului dacă scorul FID nu se îmbunătățește după un anumit număr de epoci), însă această strategie nu funcționează datorită modului adversarial de învățare, acest lucru putând fi observat și în figura 5.3.

5.2 Segmentarea semantică

Pentru trask-ul de segmentare semantică s-a folosit versiunea B5 a arhitecturii de SegFormer, deoarece aceasta are capacitatea de a face cele mai precise preziceri (vezi figura 2.15). Folosind platforma HuggingFace și mecanismul de transfer learning se poate antrena destul de ușor un astfel de model.

5.2.1 Alegerea setului de date

Initial s-a dorit refolosirea setului de date cu persoane de la task-ul de generare de imagini. Puteau fi folosite pozele cu fundal transparent și prelucrate pentru a obține măștile de segmentare, unde 1 reprezinta eticheta pentru persoane și 0 pentru fundal. Mască de segmentare era completată parcurgând imaginea inițială, valoarea într-un punct fiind 1 (persoană) dacă la poziția respectivă imaginea nu era transparentă (valoarea de pe canalul 4 al imaginii era diferită de 0) sau 0 altfel. Cu toate acestea, s-a dorit folosirea unui set de date mai explicit, cu mai multe etichete și de aceea s-a optat spre folosirea [mattmdjaga/human_parsing_dataset](#), disponibil pe site-ul HuggingFace.

Acest set de date are 17,706 de perechi de imagini cu persoane și măștile de segmentare din care au fost descărcate 1500 de poze de antrenare și 300 de poze pentru validare. Măștile de segmentare cuprind următoarele etichete:

Numele etichetei	Valoare	Numele etichetii	Valoare	Numele etichetii	Valoare
Background	0	Pants	6	Left-leg	12
Hat	1	Dress	7	Right-leg	13
Hair	2	Belt	8	Left-arm	14
Sunglasses	3	Left-shoe	9	Right-arm	15
Upper-clothes	4	Right-shoe	10	Bag	16
Skirt	5	Face	11	Scarf	17

Tabela 5.3: Etichetele din setul de date

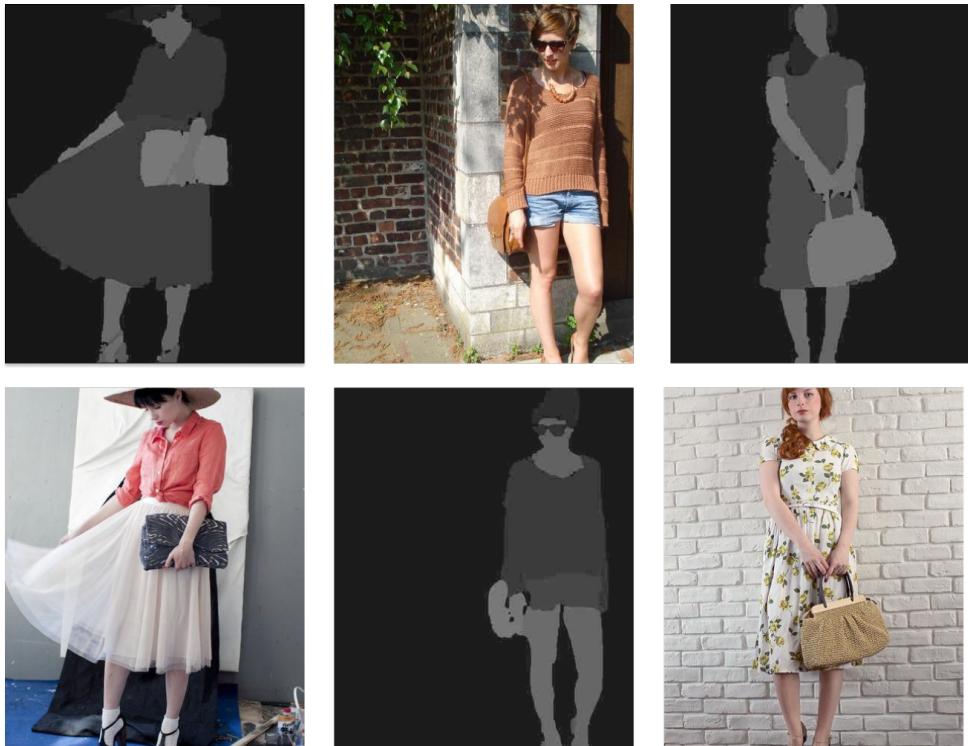


Figura 5.4: Exemple de imagini și măști de segmentare din setul de date

5.2.2 Rafinarea modelului pre-antrenat

S-a plecat de la modelul de bază ["nvidia/mit-b5"](#) care este un SegFormer antrenat pe cele 1000 de labeluri din setul de date [ImageNet](#). Prin intermediul API-ului oferit de librăria HuggingFace, procesul de antrenare a fost gestionat de clasa Trainer, specifică acestei librării. Pentru preprocesarea pozelor am folosit filtrul de ColorJitter oferit de modulul transforms din librăria PyTorch pentru a modifica ușor culoarea, contrastul sau luminozitatea pozelor, dorind să antrenăm un model robust, invariabil la acest tip de modificări. După aplicarea acestui filtru, pozele au fost trecute prin SegformerImageProcessor, care folosește straturi de conoluție pentru a genera embeddinguri de pe care rețeaua SegFormer va învăța. Ca parametri de antrenare, s-a folosit un coeficient de învățare de

0.00006 și batch-uri de dimensiune 2. Modelul a fost antrenat pe noul set de date pentru 15 epoci, salvând la fiecare 20 de iterații valoarea loss-ului, atât pentru antrenare, cât și pentru validare, și scorul modelului pentru monitorizarea procesului de antrenare. La evaluarea modelului s-au folosit IoU¹ și MeanIoU².

La finalul antrenării, modelul a fost urcat și făcut public pe platforma celor de la HuggingFace pentru a fi folosit și în alte experimente de către comunitatea de cercetători, fiind disponibil la acest [link](#). Modelul a strâns 649 descărcări în mai puțin de o lună de la data publicării. Mai mult, atât scripturile folosite pentru antrenarea CycleGAN-ului, cât și cel folosit pentru SegFormer sunt disponibile pe GitHub și pot fi accesate prin acest [link](#).

¹un mod de calcul al acurateții ce implică calcularea raportului dintre aria de intersecție și aria formată prin reuniunea predicției cu cea a etichetei

²reprezintă media scorurilor IoU, calculate per fiecare clasă

Capitolul 6

Experimente și rezultate

6.1 Antrenarea CycleGAN-ului

Pentru antrenarea CycleGAN-ului s-a testat întâi performanța arhitecturii originale, aşa cum a fost ea introdusă în [31], folosind setul de date Anime6k și pozele cu oameni nedecupate.



Figura 6.1: Imagini generate în timpul antrenării

În figura 6.1 se poate observa că modelul reușește să redea anumite trăsături specifice domeniului anime (ochii, textura părului, culorile), însă structura pozei generate nu mai seamănă cu cea a imaginii de intrare. Inițial s-a crezut că folosirea pozelor cu persoane cu background negru (obținute prin înlocuirea backgroundului transparent) poate ajuta rețeaua să se concentreze doar pe transformarea persoanelor, însă acest lucru s-a dovedit a fi fals în practică. Generatorul încearcă să transforme întreaga imagine și nu are foarte multe informații din care să poată învăța dacă mare parte din imagine are valoarea 0. Această încercare a dus la apariția mode collapse-ului, aspect observat în timpul inferenței modelului, toate pozele generate având aceeași culoare.



Figura 6.2: Încercările inițiale au dus la mode collapse

Aplicând toate modificările asupra arhitecturii și sugestiile de antrenare propuse în capitolul anterior, se poate evita mode collapse-ul, obținându-se poze de o calitate mult mai bună.



Figura 6.3: Colecție de poze generate din experimentele noastre

Spre deosebire de rezultatele inițiale, pozele din figura 6.3 reușesc să redea aceleasi trăsături specifice stilului anime, dar păstrând culorile și compoziția din pozele reale, aspect extrem de important deoarece masca de segmentare generată trebuie să se potrivească și cu varianta anime a persoanei. În continuare vor fi prezentate rezultatele obținute în experimentele noastre, evidențiind felul în care setul de date folosit sau combinația de hiperparametri pot afecta imaginile generate.

Real Image	Low Consistency	Medium Consistency	Default Consistency	High Consistency

Figura 6.4: Rezultate obținute în funcție de hiperparametrii aleși

Figura 6.4 prezintă rezultatele obținute prin antrenarea rețelei CycleGAN pe setul de date Anime6k, folosind combinațiile de hiperparametri specificate în capitolul anterior. Se poate remarcă faptul că, contrar intuiției, mărirea coeficientului pentru loss-ul de consistență într-un ciclu duce la o modificare mult mai accentuată a pozei inițiale. Acest fenomen poate fi explicat prin faptul că rețea învăță foarte mult prin reconstruirea imaginii anime inițiale, pornind de la o imagine cu persoane reale generată. Cu toate acestea, trebuie să se găsească un echilibru pentru coeficientul acestui loss, deoarece se poate observa în ultima coloană a tabelului că un loss de consistență prea mare poate duce la apariția mai multor artefacte în pozele generate. Coeficientul loss-ului pentru GAN trebuie să aibă o valoare scăzută, deoarece s-a observat pe cale experimentală că oferirea unei ponderi mari pentru acest obiectiv duce la transformări mult prea drastice în structura pozei, asemănător exemplelor din figura 6.1 și la apariția mode collapse-usului.



Figura 6.5: Generarea pozelor cu oameni reali din poze cu personaje anime

Cel de-al doilea generator ce aproximează funcția inversă are un task mult mai greu de făcut deoarece patternurile specifice domeniului pozelor cu persoane sunt mult mai complexe (cu mai multe detalii, umbre, texturi) față de cele animate, care sunt mult simplificate și stilizate de artiști. Totuși, se pot extrage niște exemple convingătoare, ca în figura 6.5. Gradul de realism al acestor poze este mai puțin relevant pentru taskul principal, însă acest generator este important în reconstruirea imaginilor, deci cu cât acesta învăță mai bine caracteristicile specifice domeniului cu persoane reale, cu atât și generatorul de personaje anime va fi mai puternic.

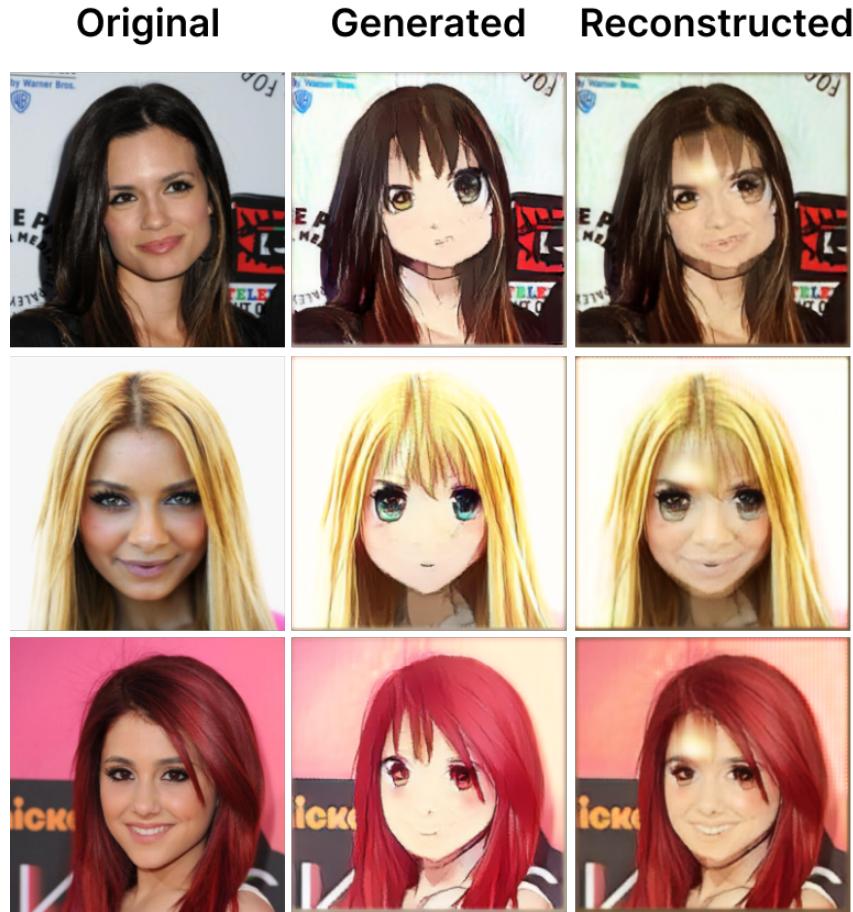


Figura 6.6: Reconstruirea pozelor

În figura 6.6 este ilustrat mecanismul de reconstruire al imaginilor. O imagine cu o persoană este transformată în personaj anime folosind primul generator, iar apoi poza generată este pasată ca input pentru cel de-al doilea generator. Loss-ul de consistență este calculat ca distanță dintre imaginea inițială (coloana 1) și cea reconstruită (coloana 3). Așa cum am menționat anterior, este bine să focusăm procesul de învățare pe acest obiectiv pentru a conserva structura poziei inițiale.

Pe lângă alegerea parametrilor, rezultatul final este puternic influențat și de calitatea și volumul de date. În general, dacă setul de date are un stil relativ consistent, CycleGAN-ul nu are nevoie de un număr foarte mare de date. Acest lucru poate fi observat în cazul antrenării pe subsetul Selfie2anime care conține puțin peste 3 mii de poze. Tot aici, se mai observă un alt avantaj al obiectivului de consistență. Rețeaua se concentrează mai mult pe schimbarea culorilor și a texturii și mai puțin pe modificarea structurii [31] și de aceea, deși Selfie2anime conține doar poze cu personaje feminine (și în general și celelalte subseturi au un bias în această direcție), generatorul reușește să transforme toate pozele cu persoane, indiferent de sexul acestora.

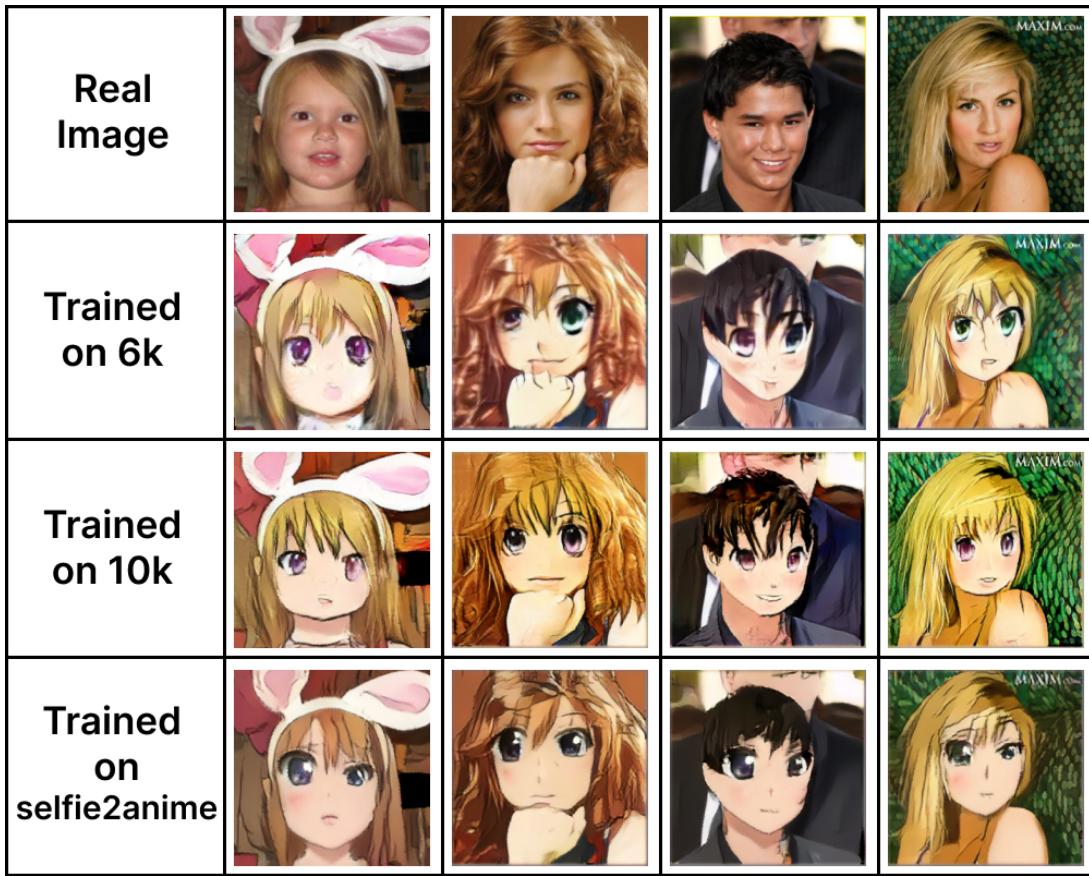


Figura 6.7: Rezultate obținute în funcție de setul de date folosit

Figura 6.7 prezintă rezultatele rețelelor antrenate pe seturi de date diferite, dar folosind aceeași hiperparametri (valorile implicate). Se poate observa clar că toate cele 3 modele au reușit să conveargă, producând rezultate stabile și convingătoare, însă există clar o diferență între stilul redat de fiecare rețea în parte. Deși rețeaua nu are nevoie de foarte multe date pentru a reda stilul anime, se remarcă faptul că mărirea volumui de date are niște beneficii, pozele generate folosind seturile Anime6k și Anime10k având mai multe detalii (spre exemplu în zona părului) și mai puține imperfecțiuni (pozele generate de rețeaua antrenată pe Selfie2anime au de multe ori ochii de dimensiuni diferite). Pe de altă parte, și folosirea unor imagini foarte diverse poate face ca rețelele să conveargă mai greu. Spre exemplu, unele dintre imaginile generate de rețeaua antrenată pe Anime10k nu au ochii transformați în stil anime, fenomen mult mai puțin frecvent pentru rețeaua antrenată pe Anime6k. Credem că acest lucru se întâmplă deoarece setul de date Anime10k este format prin unirea a trei seturi de date cu stiluri diferite, astfel rețelei fiind mult mai greu să generalizeze modelul pentru ochi care necesită și schimbări structurale, nu doar de culoare.

S-a încercat și folosirea unui set de date compus doar din poze din setul de date Anime Data Faces și s-au obținut următoarele rezultate:



Figura 6.8: Rezultate obținute prin antrenarea rețelei doar pe poze cu fețe

În acest caz se observă că modelul a reușit să redea foarte bine stilul anime. Mai mult, deoarece pozele erau centrate doar pe față, task-ul a devenit mai ușor, ceea ce a facilitat procesul de învățare. Din toate experimentele realizate, rețeaua antrenată pe setul de date cu fețe a reușit să obțină cel mai bun scor FID (49.82), însă pentru ansamblul de modele s-a decis să nu fie folosită această variantă, deoarece pozele generate aveau forma feței modificată prea mult (figura 6.8), iar măștile de segmentare nu se mai potriveau.

Parameters \ Dataset	Low Consistency	Mid Consistency	Default Consistency	High Consistency
Anime6k	90.64	84.40	66.30	72.15
Anime10k	98.31	93.77	88.54	87.10
Selfie2anime	107.52	93.20	90.41	97.18

Figura 6.9: Scorurile FID pentru cele mai bune epoci din experimentele realizate, grupate în funcție de setul de date și de parametrii folosiți

Tabelul 6.9 rezumă performanța tuturor experimentelor efectuate, evaluate prin calcularea scorului FID pe 1000 de poze generate din setul de validare cu persoane reale. Este important de menționat faptul că scorul FID este mai relevant atunci când comparăm modelele care diferă doar prin parametrii diferenți, deoarece scorul FID este puternic influențat de setul de date folosit. Este de așteptat ca atunci când folosim un set de

date foarte divers precum Anime10k să obținem un scor FID mai mare. De asemenea, pe lângă urmărirea valorilor funcției de loss, putem să estimăm dacă modelul este stabil dacă diferența dintre scorul FID obținut la finalul antrenării (epoca 200) și cel mai bun scor FID nu este foarte mare (în cazul experimentelor noastre s-a observat o diferență de maxim 10).

6.2 Antrenarea SegFormerului

Antrenarea unui model SegFormer stabil a fost mult mai ușoară deoarece s-a optat spre folosirea unui model pre-antrenat. Astfel, după rafinarea modelului s-au obținut următoarele acurateți, calculate folosind metrica IoU:

Numele etichetei	IoU	Numele etichetii	IoU	Numele etichetii	IoU
Face	0.8294	Hair	0.8171	Sunglasses	0.6400
Hat	0.7625	Bag	0.7729	Scarf	0.2956
Right-arm	0.7705	Left-arm	0.7579	Upper-clothes	0.7700

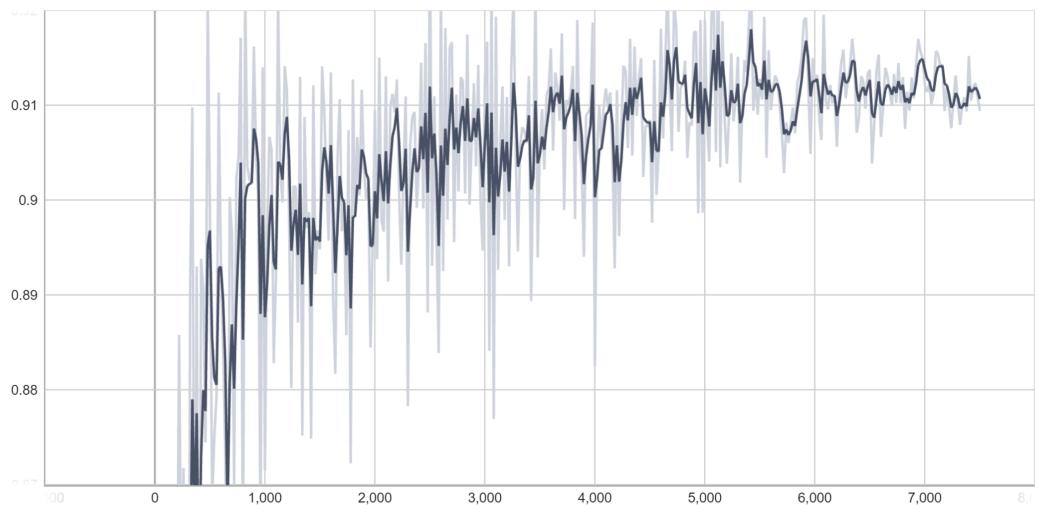
Tabela 6.1: Valoarea IoU pe fiecare etichetă

Numele etichetei	Acuratețe	Numele etichetii	Acuratețe	Numele etichetii	Acuratețe
Face	0.9094	Hair	0.8974	Sunglasses	0.7540
Hat	0.8561	Bag	0.8232	Scarf	0.3662
Right-arm	0.8287	Left-arm	0.8506	Upper-clothes	0.8553

Tabela 6.2: Acuratețea la nivel de pixel pe fiecare etichetă

Valoarea totală a acurateții globale este de 0.8256, iar media IoU este de 0.6258, calculată pe toate cele 17 clase. Cele mai importante clase sunt cele care vor fi folosite în cadrul ansamblului de rețele, și anume etichetele pentru față și păr, acestea având un IoU de 0.8294, respectiv 0.8171. Anumite clase au un scor al acurateții/IoU mai mic, însă acest lucru nu indică faptul că modelul nu poate învăța trăsăturile specifice acestora. Întrucât pentru rafinare s-au folosit doar 1500 de poze, au existat mai puține exemple de poze ce conțineau ochelari de soare sau eșarfe, spre deosebire de alte clase precum față, care este prezentă în fiecare poză. Dacă se adaugă mai multe date de antrenare ce conțin și aceste clase mai rare, ne așteptăm ca valoarea acurateții în aceste cazuri să crească, însă acest lucru nu a fost necesar pentru ansamblul nostru, deoarece se vor folosi doar două etichete.

Acurăte pentru eticheta față



Acurăte pentru eticheta păr

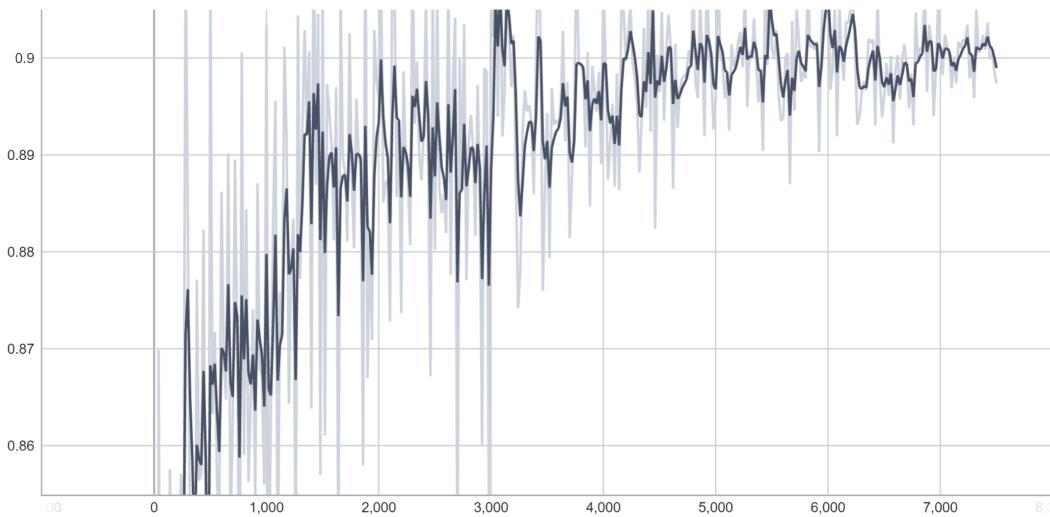


Figura 6.10: Evoluția acurateții modelului de-a lungul epocilor

Pentru a ne asigura că procesul de învățare decurge bine și că modelul este stabil, este important să urmărim și aici evoluția loss-ului și a acurateții de-a lungul epocilor. În graficele din figura 6.10 am evidențiat valorile acurateții pentru cele două etichete ce vor fi folosite în cadrul ansamblului de rețele. Uitându-ne la grafic putem observa că valoarea acurateții crește gradual și se stabilizează spre finalul procesului de antrenare. Această evoluție a acurateții este specifică unui proces de învățare optim. Mai mult, antrenarea îndelungată după ce acuratețea modelului s-a stabilizat poate duce adesea la overfit.

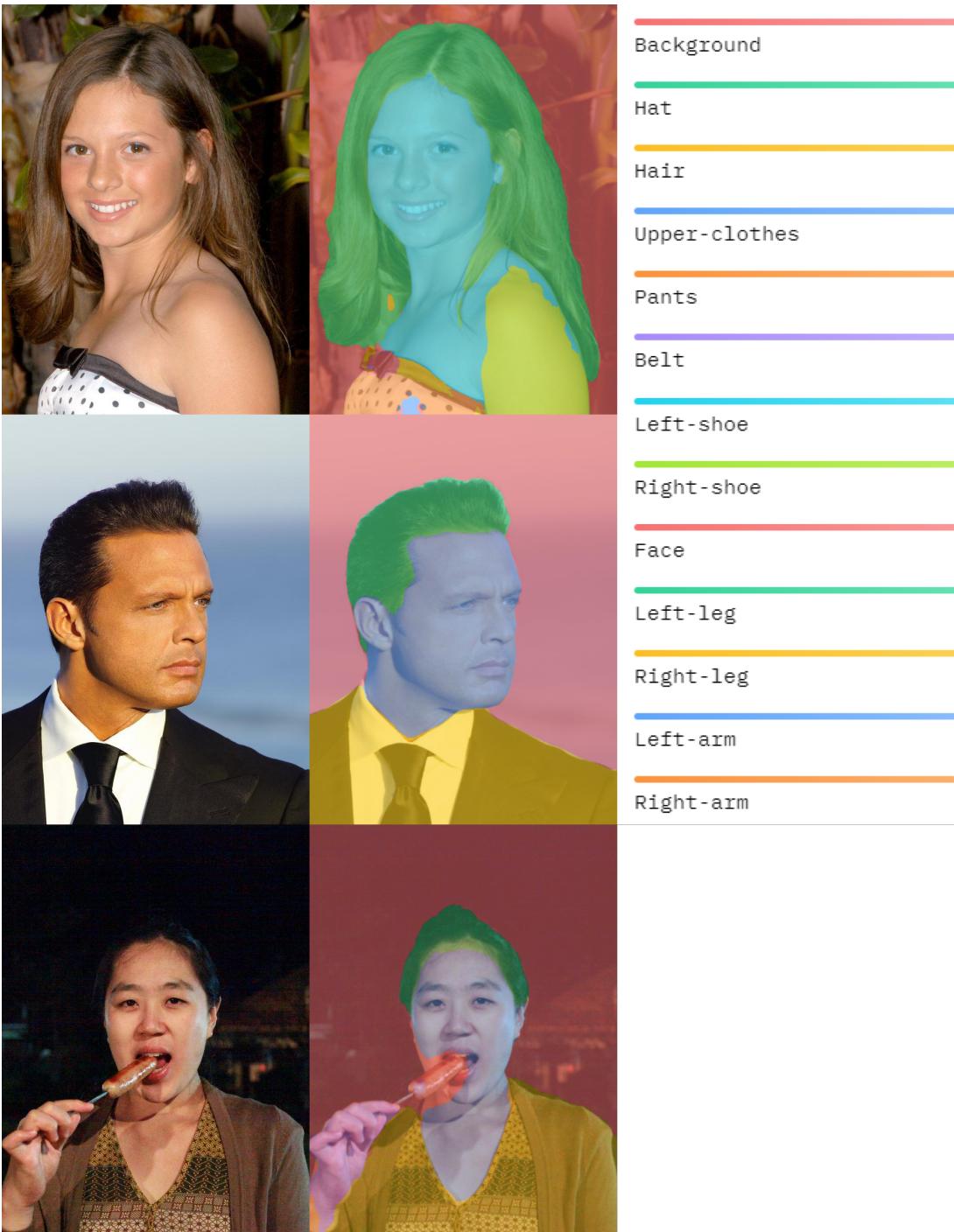


Figura 6.11: Rezultatele segmentării

În figura 6.11 s-a folosit modelul antrenat pentru segmentarea unor poze din setul de date cu oameni reali. Pe coloana din stânga sunt pozele inițiale iar în coloana din dreapta rezultatele după segmentare. Fiecare etichete i-a fost asociată o culoare, conform legendei din partea dreaptă, iar pozele au fost colorate în funcție de masca de segmentare generată de rețea.

Capitolul 7

Prezentarea aplicației

Pentru inferență ușoară a modelului și pentru a oferi un suport vizual utilizatorilor s-a construit și o aplicație (interfață vizuală - GUI) care să însoțească cele două rețele. Rolul acestei aplicații este de a ghida utilizatorul prin pașii procesului de prelucrare al imaginilor pentru a obține rezultatele generate.

7.1 Funcționalități

Aplicația asigură îndeplinirea următoarelor funcții:

1. Încărcarea pozelor în formatul dorit (PNG, JPG, JPEG, etc.) în aplicație și afișarea imaginii încărcate;
2. Segmentarea imaginii încărcate folosind rețea de tip SegFormer antrenată și afișarea zonelor selectate din poză (clasele selectate sunt față și părul);
3. Editarea zonei selectate prin desenare peste aceasta (utilizatorul poate să steargă sau să adauge anumite zone);
4. Generarea mai multor variante anime a zonei selectate folosind una dintre cele trei versiuni de CycleGAN, selectată de utilizator;
5. Selectarea unei variante de poză generată și descărcarea ei în formatul dorit, la aceeași rezoluție cu poza inițială.

7.2 Structura aplicației

Aplicația a fost construită folosind Tkinter și conține următoarele pagini:

1. Pagina de start, ce prezintă utilizatorului instrucțiunile de folosire ale aplicației;
2. Pagina de încărcare a pozei;

3. Pagina de vizualizare și editare a rezultatului Segmentării (tot aici utilizatorul alege și versiunea CycelGAN-ului);
4. Pagina de vizualizare a pozei generate (aici utilizatorul putând explora mai multe variante și descărca varianta preferată).

Fiecare pagină este modelată de o clasă ce moștenește din clasa Frame, specifică librăriei Tkinter. Astfel, în timpul rulării aplicației, o singură pagină este încărcată în memorie o dată, iar tranziția dintre pagini se face prin crearea unei noi instanțe a clasei respective și schimbarea cadrului curent (frame-ul), prin apăsarea unui buton din interfața grafică. Fereastra are dimensiune fixă, iar pozele pot fi de orice dimensiune, aşa că pentru afișarea armonioasă a pozei în interfață s-a folosit un DTO¹ (clasa TaskImage), care păstrează în memorie atât poza originală, cât și o variantă redimensionată ce redă proporțiile initiale. Tot prin intermediul acestui DTO se memorează și rezultatul segmentării (masca de segmentare), pentru a îmbunătăți eficiența timp a programului (procesul de segmentare făcându-se doar o dată per imagine). De asemenea, tot pentru îmbunătățirea vitezei aplicației, operațiile de segmentare și generare sunt făcute pe GPU (dacă acesta este disponibil), iar modelul de generator este încărcat în memorie doar atunci când e nevoie de el (doar o singură variantă este încărcată în memorie, nu toate 3).

Stilul aplicației este unul omogen, intuitiv și ușor de utilizat, având butoane dedicate pentru fiecare acțiune. Pentru o mai bună organizarea a codului s-au creat frame-uri template (BaseFrame și BaseFrameWithImage) care sunt extinse de celelalte pagini.

În cadrul ansamblului de modele s-a optat pentru includerea interacțiunii umane. Așadar, înainte de a pasa imaginea generatorului, utilizatorul poate evalua și modifica rezultatul segmentării pentru a obține rezultate mai bune. La editarea unei măști de segmentare, imaginea este afișată într-o fereastră nouă, împreună cu masca de segmentare, folosind librăria OpenCV. Fereastra nou deschisă are un eveniment care preia click-urile de la utilizator și desenează un cerc (de rază proporțională cu dimensiunea imaginii) la locația click-ului, creând un efect de pensulă. Utilizatorul poate selecta (click dreapta) sau deselecta (click stânga) anumite porțiuni din poză.

¹Data Transfer Object - o clasă simplă ce facilitează transferul de date în intermediul unei aplicații

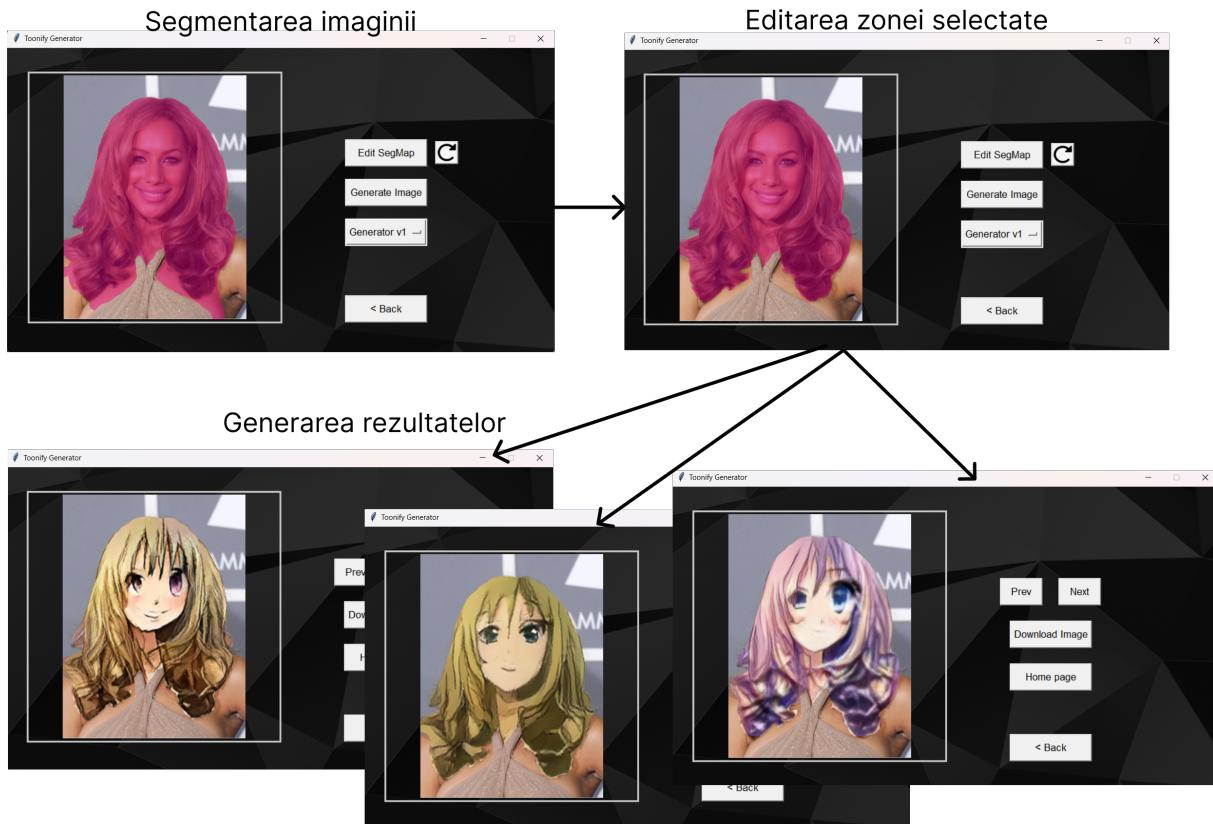


Figura 7.1: Procesul de generare de imagini prin intermediul aplicației

În figura 7.1 se observă pașii prin care un utilizator trece pentru a genera poze utilizând aplicația. Pentru fiecare poză încărcată se pot genera până la 15 versiuni diferite ale personajului anime bazate pe persoana din poză. Aceste versiuni sunt obținute prin folosirea a diferitor modele din experimentele noastre și prin aplicarea unor filtre de culori pe poza originală. În cadrul aplicației s-a folosit câte un model antrenat cu parametri implicați pe fiecare din cele trei seturi de date.



Figura 7.2: Variante diferite obținute folosind un singur model, prin aplicarea filtrelor de culoare

7.3 Galerie de rezultate

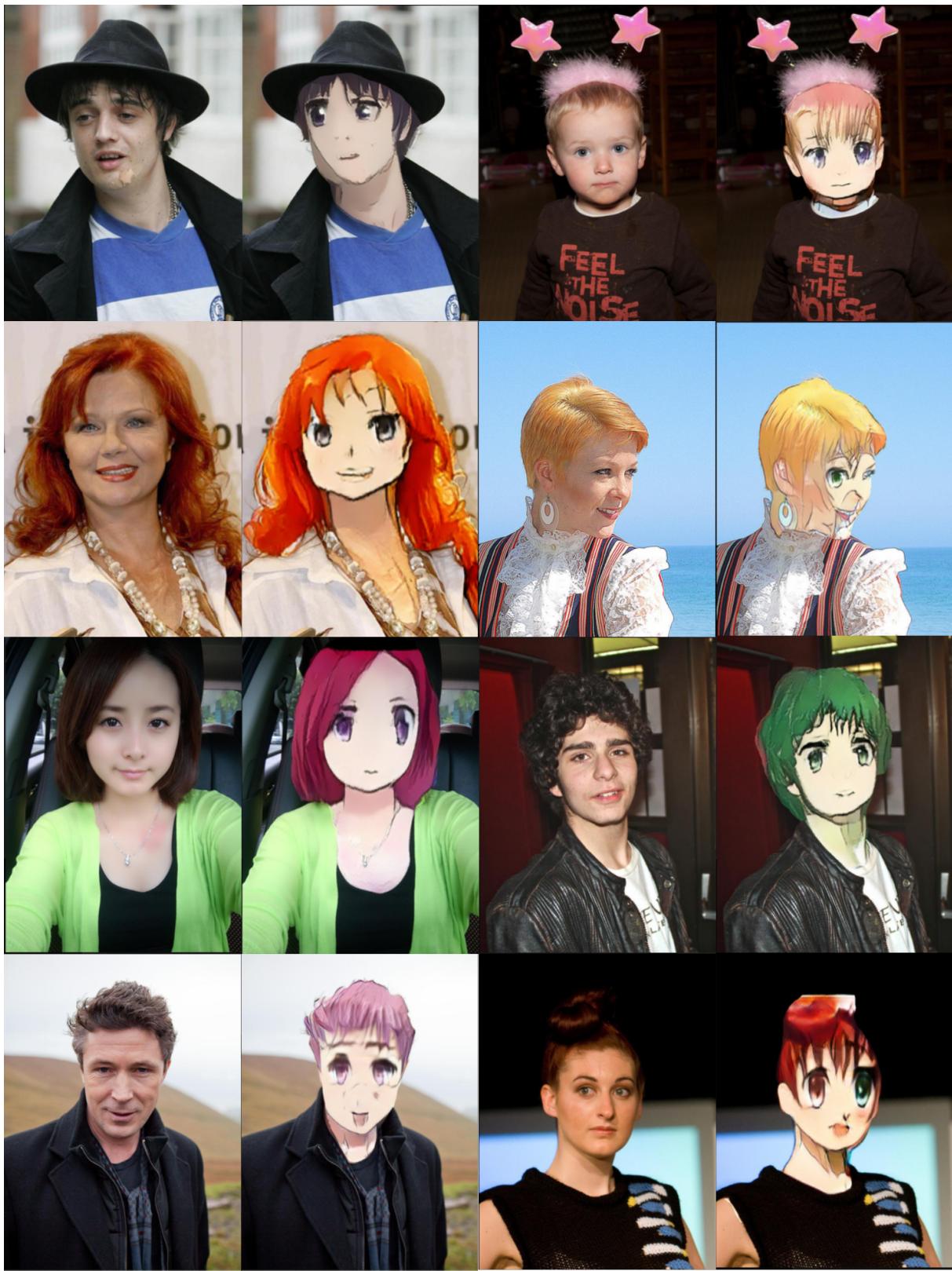


Figura 7.3: Diverse imagini obținute prin intermediul aplicației

Capitolul 8

Concluzii și dezvoltări ulterioare

Se dorește ca acest studiu să se dezvolte pe mai multe planuri în viitor:

- **Îmbunătățirea algoritmilor de învățare automată:** Arhitecturile prezente pot fi îmbunătățite folosind ideile din [27] și [26], mecanismul de atenție putând genera imagini mai bine structurate cu mai puține artefacte. Mai mult, modele de arhitectură noi apărute ca Stable Diffusion pot reprezenta o altă metodă de a transfera stilul anime la o calitate înaltă;
- **Îmbunătățirea aplicației:** în cadrul aplicației au fost luate în calcul doar două (față și părul) din cele 17 clase pe care a fost antrenat algoritmul de segmentare. O funcționalitate interesantă ar fi posibilitatea de a selecta ce clase din cele 17 se dorește a fi segmentate pentru ca mai apoi să fie transformate în anime. Acest lucru ar îmbunătăți efectul artistic de colaj și ar permite utilizatorilor să-și exprime creativitatea mai mult;
- **Generalizarea task-ului:** acest studiu arată că arhitectura propusă funcționează pe un caz destul de specific, însă studiile viitoare s-ar putea concentra pe antrenarea unui SegFormer cu mai multe clase pentru a transfera stilul oricărui obiect din imagine.

Așa cum a fost menționat la începutul lucrării, acest studiu are diverse aplicații practice. Comunitatea iubitorilor de anime este foarte mare și sunt o mulțime de evenimente organizate la nivel internațional, dedicate acestui stil de animație. Cu toate acestea, pentru a anima o poveste este nevoie de foarte multă muncă, mulți artiști fiind nevoiți să deseneze fiecare scenă dintr-un serial anime. Sperăm că acest program cu învățare automată să poată fi folosit ca o unealtă ce oferă inspirație artiștilor și facilitează procesul de creație, sau o modalitate de exprimare pentru pasionații de anime care doresc să vadă cum ar arăta transpuși în lumea pe care o iubesc atât de mult. Tot aici, datorită vitezei ridicate de generare a arhitecturii CycleGAN, putem vorbi nu doar de editare la nivel de poză, ci și editarea video, în timp real, lucrul ce ar putea deschide portile și spre utilizarea

acestei arhitecturi pentru un joc cu realitate augmentată. Nu în ultimul rând, generalizarea task-ului de segmentare și învățarea și altor stiluri poate veni în ajutorul oricărui designer grafic în crearea de materiale grafice.

În concluzie, studiul de fată propune o arhitectură nouă, formată din două rețele de ultimă generație, capabilă să transfere stilul anime, într-un mod localizat, pe poze cu persoane reale. Această arhitectură este ușor de integrat în cadrul unei aplicații și poate avea multe utilizări în practică.

Bibliografie

- [1] Arthur Arnx, „First neural network for beginners explained (with code)”, în *Towards Data Science* (2019), URL: <https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cf37e06eaf>.
- [2] Angela Beklemysheva, „What is PyTorch?”, în *TechTarget* (2022), URL: <https://www.techtarget.com/searchenterpriseai/definition/PyTorch>.
- [3] Angela Beklemysheva, „Why Use Python for AI and Machine Learning?”, în *Steel Kiwi* (2020), URL: <https://steelkiwi.com/blog/python-for-ai-and-machine-learning/>.
- [4] Jason Brownlee, „How to Implement the Frechet Inception Distance (FID) for Evaluating GANs”, în *Machine Learning Mastery* (2019), URL: <https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/>.
- [5] Saul Dobilas, „GANs: Generative Adversarial Networks — An Advanced Solution for Data Generation”, în *Towards Data Science* (2012), URL: <https://towardsdatascience.com/gans-generative-adversarial-networks-an-advanced-solution-for-data-generation-2ac9756a8a99>.
- [6] Ketan Doshi, „Transformers Explained Visually (Part 1): Overview of Functionality”, în *Towards Data Science* (2020), URL: <https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452>.
- [7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit și Neil Houlsby, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, 2021, arXiv: [2010.11929 \[cs.CV\]](https://arxiv.org/abs/2010.11929).
- [8] Kyt Dotson, „IBM could replace up to 7,800 jobs with AI in a few years, CEO says”, în *SiliconANGLE* (2023), URL: <https://siliconangle.com/2023/05/02/ibm-replace-7800-jobs-ai-years-ceo-says/>.

- [9] Gavin Edwards, „Machine Learning | An Introduction”, în *Towards Data Science* (2018), URL: <https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0>.
- [10] Szuzina Fazekas, „Artificial intelligence and neural networks in radiology – Basics that all radiology residents should know”, în *AKJournals* (2022), URL: <https://akjournals.com/view/journals/1647/14/2/article-p73.xml>.
- [11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville și Yoshua Bengio, *Generative Adversarial Networks*, 2014, arXiv: [1406.2661 \[stat.ML\]](https://arxiv.org/abs/1406.2661).
- [12] Larry Hardesty, „Explained: Neural networks”, în *MIT News* (2017), URL: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>.
- [13] Xin Chen Jie Chen Gang Liu, „AnimeGAN: A Novel Lightweight GAN for Photo Animation”, în (2020), URL: <https://tachibananayoshino.github.io/AnimeGANv2/>.
- [14] Chiou-Shann Fuh Yu-Jing Lin, „Generating Anime Faces From Human Faces With Adversarial Networks”, în *National Taiwan University* (2018), URL: <https://www.csie.ntu.edu.tw/~fuh/personal/GeneratingAnimeFacesfromHumanFaceswithAdversarialNetworks.10.A6-1.pdf>.
- [15] George V. Jose, „Useful Plots to Diagnose your Neural Network”, în *Towards Data Science* (2019), URL: <https://towardsdatascience.com/useful-plots-to-diagnose-your-neural-network-521907fa2f45>.
- [16] Vijay Kanade, „What Is Machine Learning? Definition, Types, Applications, and Trends for 2022”, în *SiliconeANGLE* (2022), URL: <https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-ml/>.
- [17] Vedant Kumar, „Convolutional Neural Networks”, în *Towards Data Science* (2020), URL: <https://towardsdatascience.com/convolutional-neural-networks-f62dd896a856>.
- [18] Irene Yu-Hua Gu Muhaddisa Barat Ali, „Domain Mapping and Deep Learning from Multiple MRI Clinical Datasets for Prediction of Molecular Subtypes in Low Grade Gliomas”, în *Brain Sci.* (2020), URL: <https://doi.org/10.3390/brainsci10070463>.
- [19] Jessica Powers Niklas Donges, „What Is Transfer Learning? Exploring the Popular Deep Learning Approach.”, în *Built In* (2022), URL: <https://towardsdatascience.com/machine-learning-general-process-8f1b510bd8af>.
- [20] Prakash Pandey, „Deep Generative Models”, în *Towards Data Science* (2018), URL: <https://towardsdatascience.com/deep-generative-models-25ab2821af3>.

- [21] Ashish Patel, „Chapter-7 Under-fitting, over-fitting and its solution”, în *Medium* (2018), URL: <https://medium.com/ml-research-lab/under-fitting-over-fitting-and-its-solution-dc6191e34250>.
- [22] Victor Roman, „How To Develop a Machine Learning Model From Scratch”, în *Towards Data Science* (2018), URL: <https://towardsdatascience.com/machine-learning-general-process-8f1b510bd8af>.
- [23] Sabyasachi Sahoo, „Residual blocks — Building blocks of ResNet”, în *Towards Data Science* (2018), URL: <https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec>.
- [24] Ausif Mahmood Saleh Albelwi, „A Framework for Designing the Architectures of Deep Convolutional Neural Networks”, în *MDPI* (2017), URL: <https://www.mdpi.com/1099-4300/19/6/242>.
- [25] Markus Schmitt, „What Is Machine Learning? – A Visual Explanation”, în *Data Revenue* (2022), URL: <https://www.datarevenue.com/en-blog/what-is-machine-learning-a-visual-explanation>.
- [26] Hao Tang, Hong Liu, Dan Xu, Philip H. S. Torr și Nicu Sebe, *AttentionGAN: Unpaired Image-to-Image Translation using Attention-Guided Generative Adversarial Networks*, 2021, arXiv: [1911.11897 \[cs.CV\]](https://arxiv.org/abs/1911.11897).
- [27] Dmitrii Torbunov, Yi Huang, Haiwang Yu, Jin Huang, Shinjae Yoo, Meifeng Lin, Brett Viren și Yihui Ren, *UVCGAN: UNet Vision Transformer cycle-consistent GAN for unpaired image-to-image translation*, 2022, arXiv: [2203.02557 \[cs.CV\]](https://arxiv.org/abs/2203.02557).
- [28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser și Illia Polosukhin, *Attention Is All You Need*, 2017, arXiv: [1706.03762 \[cs.CL\]](https://arxiv.org/abs/1706.03762).
- [29] Yugesh Verma, „A Beginner’s Guide to Using Attention Layer in Neural Networks”, în *Mystery Vault* (2021), URL: <https://analyticsindiamag.com/a-beginners-guide-to-using-attention-layer-in-neural-networks/>.
- [30] Enze Xie, Wenhui Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez și Ping Luo, *SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers*, 2021, arXiv: [2105.15203 \[cs.CV\]](https://arxiv.org/abs/2105.15203).
- [31] Jun-Yan Zhu, Taesung Park, Phillip Isola și Alexei A. Efros, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, 2020, arXiv: [1703.10593 \[cs.CV\]](https://arxiv.org/abs/1703.10593).