### UNIVERSITATEA DIN BUCUREŞTI

**FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ**

**SPECIALIZAREA INFORMATICĂ**

**Lucrare de licenţă**

# Find your music

### Absolvent

### Andreea Diana Gherghescu

**Coordonator ştiinţific**

**Conf. dr. Paul Irofti**

**Bucureşti, iunie 2023**

# 

## 

## 

## 

## Rezumat

Find your music are ca scop introducerea în lumea muzicii şi uşurarea procesului de găsire a unei melodii. După introducerea unui segment muzical de 10 secunde, algoritmul va scana baza sa de date şi va întoarce numele melodiei recunoscute şi interpretul acesteia.

Pentru fiecare melodie din baza de date, vor exista versiuni ale acesteia în alte genuri muzicale, prelucrate cu ajutorul modelelor de inteligenţă artificială. Acest aspect va permite aplicaţiei recunoaşterea melodiilor în contexte nefavorabile, precum existenţa zgomotului, reinterpretări sau mixaje.

## Abstract

Find your music aims to be an introduction to the world of music and ease the process of finding a song. After entering a 10-second music segment, the algorithm will scan its database and return the name of the recognized song and its artist.

For each song in the database, there will be versions of it in other musical genres, processed with the help of artificial intelligence models. This aspect will allow the application to recognize songs in unfavorable contexts, such as the existence of noise, reinterpretations or mixes.

# 

# 

# 

# 

# 

# Cuprins

**1** [**Introducere 5**](#_80fc8jnv0cam)

[1.1 Prezentarea aplicaţiei](#_mmxelsawe8ed) 5

[1.2 Nevoia pentru aplicaţie](#_q57h3n8fel05) 5

[1.3 Aplicaţii similare](#_fp04a0j99du0) 6

[1.4 Evidenţierea trăsăturilor diferite ale aplicaţiei Find your music](#_cdmy9gtz82ht) 7

[1.5 Structura lucrării](#_kr5d8wavk66m) 7

**2** [**Preliminarii 8**](#_l1xpe730gt31)

[2.1 Despre sunet şi reprezentarea sa în format digital](#_yrblcxr1czei) 8

[2.1.1 Ce este sunetul?](#_s9j1nfgu4vm) 8

[2.1.2 Metode de reprezentare a sunetului](#_gagtfb70w2gz) 8

[2.2 Reţele neuronale artificiale](#_7wknesj3maoz) 11

[2.2.1 Definiţie](#_9s0dt5ys7umg) 11

[2.2.2 Structura](#_ipl0hotkydf7) 11

[2.2.3 Straturi neuronale](#_3l0p12anh4r) 13

[2.3 Funcţii de activare](#_dygtk7mtfzx9) 16

[2.4 Alte tipuri de straturi](#_r3qb0q4rxu5o) 18

[2.5 Funcţii obiectiv](#_3rjhsscl7e4z) 19

[2.6 Rata de învăţare](#_wfvgao5m2ws1) 23

[2.7 Optimizatori](#_l6fyh5acozhi) 24

[2.8 Reţeaua Siameza (Siamese Network)](#_2r2hk4ob4cl8) 25

[2.9 Reţele Generative Adversariale (Generative Adversarial Networks)](#_e52n80w7azft) 26

[2.10 Arhitectura UNET](#_e2wmyez41ey4) 28

[**3**](#_114iho63t5in)  [**Algoritmi 31**](#_myqxmem3f0cx)

[3.1 Setul de date](#_f2pv40a8amrx) 31

[3.2 Crearea spectrogramelor](#_jqmyhqqexi4v) 31

[3.3 Algoritmul de schimbare de gen muzical](#_hjox0mdxvzio) 32

[3.3.1 Introducere în problema transferului de stiluri](#_afjkmimay182) 32

[3.3.2 Neural Style Transfer](#_b7g1rltwbu72) 32

[3.3.3 Deep Convolutional GAN - DCGAN](#_vxj2w4ww7azp) 33

[3.3.4 CycleGAN](#_y12ig4lcrq0q) 35

[3.4.5 Siamese Discriminator GAN (SDGAN)](#_p8d3se6vuyq1) 36

[3.4.6 TraVeLGAN (Transformation Vector Learning GAN)](#_t08w6q56je8d) 37

[3.3 Algoritmul de recunoaştere muzicală](#_7mrjd62umzrs) 39

[3.3.1 Introducere](#_tg2anyrqgehe) 39

[3.3.2 “Constelaţia”](#_66v1twmw4wfw) 39

[3.3.3 Dicţionarul de hash-uri](#_mz3rryfo8ix3) 40

[3.3.4 Baza de date finală](#_dgrc0hhdkrmj) 41

[3.3.5 Procesul de recunoaştere](#_wwqy933piznu) 41

[**4**](#_tdrf79pccpn4) [**Tehnologii utilizate în implementarea algoritmilor 42**](#_3d44zfkm5t6f)

[4.1 Python](#_swmt4rasv5so) 42

[4.2 PyTorch](#_5n0edyf3bkdb) 42

[4.2.1 PyTorch vs TensorFlow](#_zajwfn46kz96) 42

[4.2.2 PyTorch vs Keras](#_1t08i8ig8iza) 43

[4.3 Librării Python folosite](#_qvykcjdts5rr) 43

[4.4 Particularităţi hardware](#_emqi6n30lmfh) 47

[**5**](#_7ad8v6j4qynp) [**Rezultate obţinute şi comparaţii 48**](#_pu66l7rlymtz)

[5.1 Transferul de gen muzical](#_hf42fz4vfdul) 48

[5.1.1 DCGAN](#_thku3k70ezcg) 49

[5.1.2 SDGAN](#_ksju1linbs0x) 51

[5.1.3 TraVeLGAN](#_3lk54v6bvxqh) 55

[5.2 Recunoaşterea muzicală](#_nyd5lvjq1bih) 58

[**6**](#_burlu235adq) [**Concluzii 64**](#_urp97zps7xym)

6.1 [Direcţii viitoare](#_1q281zi51772) 64

[**Bibliografie 65**](#_8oqf6t4d3ta)

# 

# **Capitolul 1**

# **Introducere**

## **1.1 P**rezentarea aplicaţiei

Prezenta documentaţie însoţeşte lucrarea practică, ce este compusă dintr-o serie de algoritmi şi un set de date alcătuit din melodii pe care aceştia lucrează.

Algoritmii au următoarele scopuri:

* preprocesarea şi prelucrarea melodiilor pentru a fi folosite în algoritmii principali;
* realizarea transferului de gen muzical asupra melodiilor;
* recunoaşterea unei melodii dintr-un set de date, având dat un segment de 10 secunde din aceasta.

Având o funcţionalitate principală uşor de apelat, aplicaţia poate fi foarte utilă în situaţii în care utilizatorii vor să recunoască rapid o melodie pe care o aud în restaurante, pe strada, la televizor etc.

## **1.2 Nevoia pentru aplica**ţ**ie**

În zilele de astăzi, muzica se regăseşte peste tot: în filme, în sălile de fitness, în holurile de aşteptare, în sălile de clasă. Multe persoane îşi găsesc de-a lungul timpului una sau mai multe melodii de „confort”, care le ajută în îndeplinirea a diverse sarcini. Find your music le oferă mijlocul prin care pot descoperi această muzică.

Uneori, când ascultăm o melodie pentru prima oară, ne aduce aminte de o întâmplare sau o persoană şi vrem să o salvăm pentru a o mai asculta. Find your music poate ajuta în recunoaşterea melodiei.

În perioadele cu examene pentru studenţi, aceştia preferă deseori să înveţe având în fundal anumite melodii. S-au creat playlisturi speciale în acest scop, dar nu toate se potrivesc tuturor persoanelor. Un student aflat într-o sală de studiu poate folosi aplicaţia Find your music pentru a găsi melodia ascultată de colegul sau.

Dat fiind programul de lucru al adulţilor, viaţa lor poate intra uneori într-o rutină. O bună modalitate de a diversifica lucrurile este prin descoperirea de muzică nouă. Prin Find your music oamenii pot colecta muzică nouă din diverse locuri în care o aud pentru a adăuga un strop de diversitate în munca lor.

Find your music poate fi de folos dansatorilor, coregrafilor sau producătorilor de film în găsirea unei melodii potrivite pentru dansul sau ecranizarea lor.

Find your music poate reprezenta de asemenea un mijloc de relaxare, prin colectarea unor melodii plăcute din diverse medii.

## **1.3 Aplica**ţ**ii similare**

**Shazam**

Shazam este o aplicaţie mobilă de recunoaştere muzicală, ce scanează 10 secunde de sunet şi întoarce numele melodiei şi artistul său.

Este actualizată constant, putând recunoaşte aproape orice melodie din lume. Are o bază de date proprie şi privată, de dimensiuni foarte mari, stocată într-un server local.

Utilizatorii se pot loga în aplicaţie pentru a-şi păstra un istoric cu melodiile scanate.

În plus, Shazam generează playlist-uri personalizate pe baza scanărilor. Acestea conţin însă variante de 10 secunde ale melodiilor, neputând fi redate melodiile complete în aplicaţie. Shazam este conectat cu Apple Music şi poate transfera playlist-urile generate acolo.

Shazam oferă statistici despre cele mai frecvente melodii scanate în lume sau în diverse ţări în ultima perioadă de timp.

Desigur, din cauza zgomotului sau a altor factori precum interpretarea diferită faţă de original (alt ritm, altă tonalitate etc.) sau schimbarea solistului, Shazam nu obţine acurateţe de 100% în recunoaşterea melodiilor, uneori întorcând rezultate greşite sau chiar mesaje de eroare.

## **1.4 Eviden**ţ**ierea tr**ă**s**ă**turilor diferite ale aplica**ţ**iei Find** y**our** m**usic**

Obiectivul principal prin care Find your music se diferenţiază de aplicaţia Shazam este prelucrarea melodiilor înaintea adăugării lor în baza de date prin schimbarea genului muzical. Astfel, pentru fiecare melodie vor fi reţinute mai multe variante ale sale, în diverse genuri muzicale, în scopul creşterii acurateţei la identificarea lor în special în contexte precum interpretări live sau reinterpretări.

Deşi Find your music nu este scalată pentru a recunoaşte orice melodie din lume, fiind în acest stadiu încă un prototip, prin folosirea algoritmului de transfer de stil consider că poate ajunge la rezultate cel puţin la fel de bune ca ale aplicaţiei Shazam.

## 1.5 Structura lucrării

Lucrarea are următoarea structură:

1. **Introducere** - scurtă prezentare a scopului lucrării;
2. **Preliminarii** - descriere a conceptelor teoretice folosite în implementarea algoritmilor;
3. **Algoritmi** - metodele prin care este atins scopul final, descriere a posibilelor dificultăţi, comparaţii între metode;
4. **Tehnologii utilizate în implementarea algoritmilor** - despre tehnologiile folosite şi avantajele pe care le prezintă;
5. **Rezultate obţinute şi comparaţii** - prezentarea experimentelor făcute şi a rezultatelor obţinute, comparaţii;
6. **Concluzii** - încheierea lucrării, prezentarea direcţiilor viitoare.

# Capitolul 2

# Preliminarii

## **2.1 Despre sunet** ş**i reprezentarea sa** î**n format digital**

### **2.1.1 Ce este sunetul?**

Conform dicţionarului Enciclopediei Britannica [3], sunetul este „o disturbaţie mecanică dintr-o stare de echilibru ce se propagă printr-un mediu material elastic”. Mai pe scurt, sunetele sunt vibraţii continue, cauzate de schimbarea presiunii aerului. Ajungând în ureche, ele lovesc timpanul ce le propagă mai departe către nervii auditorii care le interpretează drept „sunete”.

### **2.1.2 Metode de reprezentare a sunetului**

#### **Domeniul timpului**

Fiind în sine o undă mecanică, cel mai intuitiv mod de a reprezenta sunetul este în domeniul timpului, sub formă de unde, în funcţie de timp, frecvenţă, amplitudine şi fază. În dispozitivele electronice, sunetul este interpretat prin semnale electrice iniţial continue, numite semnale analoage. Pentru a le putea procesa şi a lucra cu ele însă, trebuie discretizate prin eşantionare, ce presupune cuantizarea măsurătorilor în timp şi amplitudine.

Teorema de eşantionare Nyquist–Shannon enunţă că „dacă o funcţie nu conţine frecvenţe mai ridicate de , atunci poate fi determinată complet de ordonatele sale la o secvenţă de puncte spaţiate de mai puţin de secunde” [42], unde este momentul în timp, iar este lungimea de bandă. Aşadar, o frecvenţă de eşantionare suficientă este mai mare de eşantioane pe secundă. Echivalent, „pentru o frecvenţă de eşantionare data *fs,* reconstituirea perfectă este garantat posibilă pentru o lungime de banda ”.

Urechea umană poate detecta frecvenţe aproximativ între 20 - 20000 Hz, standardul pentru înregistrarea audio-urilor fiind o frecvenţă de eşantionare de 44100 Hz.

#### **Domeniul frecven**ţ**ei - Transformata Fourier**

Teorema Fourier afirmă că „orice semnal periodic este compus dintr-o suprapunere de unde sinusoidale pure, cu amplitudini şi faze alese corespunzător, ale căror frecvenţe sunt armonice ale frecvenţei fundamentale a semnalului”. Aşadar, cu ajutorul transformatei Fourier (Ecuaţia 2.1) putem reprezenta semnalele într-un nou domeniu, cel al frecvenţei, prin amplitudinea şi faza fiecărei unde sinusoidale ce îl compune.

*Ecuaţia 2.1 Ecuaţia Transformatei Fourier*

Această reprezentare este preferată în domeniul digital al prelucrării semnalelor, deoarece pot fi analizate mult mai uşor frecvenţele prezente, iar apoi supuse la diferite operaţii precum filtrare, mărirea sau scăderea anumitor frecvenţe, recunoaşterea tonurilor etc.

Un avantaj major al analizei în domeniul frecvenţei este înlăturarea zgomotului. Uitându-ne la o formă de undă, nu ne putem da seama cum a fost alterat un semnal, însă în general zgomotele au frecvenţe scăzute, astfel că le putem înlătura uşor folosind Transformata Fourier.

Totuşi, un dezavantaj este pierderea informaţiilor despre timp, astfel că nu putem şti în ce moment a apărut o anumită frecvenţă în secvenţa de sunete.

#### **Spectrograma**

În lumea reală însă, majoritatea semnalelor cu care lucrăm sau pe care vrem să le analizăm sunt neperiodice, astfel că frecvenţele conţinute de un semnal variază în timp. Soluţia pentru această problemă este aplicarea Transformatei Fourier pe segmente din semnal, fiind numită Transformata Fourier de Timp Scurt (Short-Time Fourier Transform). Segmentele au lungime fixă aleasă, şi se pot suprapune mai mult sau mai puţin. Rezultatul acestui proces se numeşte spectrogramă şi este un mod de a evidenţia schimbarea amplitudinii în timp a unui semnal la frecvenţe diferite.

Grafic, axa oX a spectrogramei este reprezentată de timp, mai precis de segmentele de timp pe care a fost aplicată STFT. Numărul de puncte de pe axă este dat de formula *,* unde este numărul de eşantioane din melodie, iar este pasul dintre două ferestre consecutive pe care se aplică Transformata Fourier.

Diferenţa dintre puncte în secunde este dată de formula ,unde este frecvenţa de eşantionare a melodiei. Aşadar, un punct de pe axă reprezintă începutul unei ferestre, iar diferenţa dintre două puncte consecutive este un „bin” de timp.

Axa oY reprezintă frecvenţe în Hz, mai exact reprezintă „bin”-uri de frecvenţă. Numărul de puncte de pe axa oY este dat de formula , unde este numărul de eşantioane folosite în Transformata Fourier. Fiecărui „bin” de frecvenţă îi corespunde un interval de Hz, iar punctele de pe axă sunt date de centrul „bin”-urilor.

Valorile aflate pe grafic la intersecţiile dintre timpi şi frecvenţe, sau culorile, reprezintă amplitudinea sau puterea semnalului. In general, axa oX este transformată în scară logaritmică, iar dimensiunea culorilor în decibeli.

#### **Scara** m**el**

În articolul ‘Understanding the Mel Spectrogram’ [14]este prezentată concluzia conform căreia „studiile au arătat că oamenii nu percep frecvenţele pe o scară liniară. Suntem mai buni la a detecta diferenţe între frecvenţele scăzute decât între cele ridicate. Spre exemplu, putem distinge uşor diferenţa dintre 500 Hz şi 1000 Hz, dar vom distinge cu greu diferenţa dintre 10000 Hz şi 10500 Hz, deşi distanţa dintre cele doua perechi este aceeaşi”.

Transformarea în scara mel (Ecuaţia 2.2) presupune reprezentarea frecvenţelor astfel încât diferenţe egale între unităţi să sune egal unui ascultător, fiind o îmbunătăţire majoră a reprezentării sub formă de spectrogramă.

1

*Ecuaţia 2.2 Transformarea a f Hz în m mels [41]*

Constantele prezente în formulă au fost alese experimental şi există mai multe variaţii ale lor. Teoretic, sub pragul de 700 Hz unităţile mel ar trebui să aibă o creştere liniară cu scara frecvenţelor. Deasupra sa, diferenţa între cele două ar trebui să fie logaritmică. Constanta 2595 este aleasă astfel încât, prin aplicarea formulei, 1000 mel să fie corespondentul a 1000 Hz.

## 2.2 Reţele neuronale artificiale

### 2.2.1 Definiţie

Reţelele neuronale sunt un subiect popular şi mult discutat, stând la baza paradigmei de programare numita deep learning. Prima definiţie a reţelelor neuronale din dicţionarul Oxford Languages enunţă că ele reprezintă „orice grup de neuroni care conduce impulsuri într-o manieră coordonată, ca ansamblurile de celule ale creierului ce înregistrează un stimul vizual” [27].

O a doua definiţie se referă la reţelele neuronale în contextul calculatoarelor, ele fiind „un sistem hardware sau software în care conexiunile ponderate dintre nodurile de date sunt rafinate pentru a produce rezultate cât mai precise în procesarea informaţiilor, de exemplu în recunoaşterea de tipare sau rezolvarea de probleme, cu scopul computaţiei algoritmice care necesită intervenţie umană minimă”. Putem deduce aşadar că ele sunt inspirate din sisteme nervoase centrale şi sunt organizate sub forma unor grafuri orientate.

### 2.2.2 Structura

Fiind organizate drept grafuri orientate, reţelele neuronale artificiale sunt alcătuite, deci, din noduri şi arcuri, iar informaţia trece prin ele într-un singur sens. Articolul ‘Artificial Neural Network for Machine Learning– Structure & Layers’ [34] descrie nodurile ca fiind corespondente ale neuronilor din sistemele nervoase, pe când arcurile sunt corespondente ale dendritelor şi sinapselor. În plus, faţă de reţelele neuronale anatomice, fiecare arc are asociată o valoare sau pondere. Reţeaua se îmbunătăţeşte prin ajustarea acestora, producând rezultate din ce în ce mai precise.

În mod clasic, reţelele neuronale artificiale cuprind straturi de neuroni interconectate, fiecare strat avand un rol specific (Figura 2.3) . În general, ele sunt împărţite în trei categorii:

#### **Straturi de intrare**

În scopurile pentru care sunt folosite reţelele neuronale este de obicei necesară o bază de date drept suport. În general, aceasta trebuie preprocesată prin diverse transformări pentru a ajunge la o formă anume sau pentru a extrage anumite trăsături din obiectele sale.

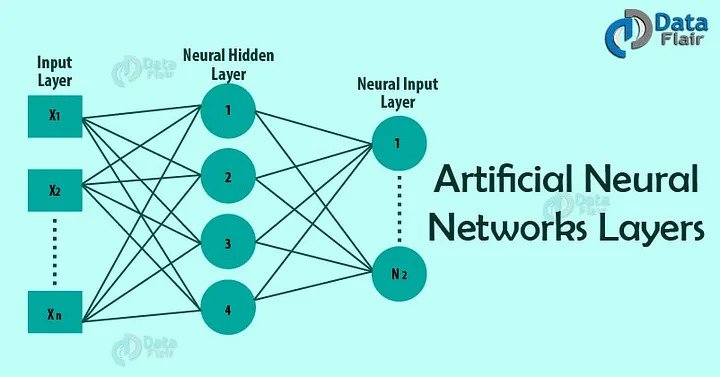
Reţelele neuronale prezintă un strat de intrare, care este responsabil de primirea datelor după preprocesare şi introducerea lor în reţea. Neuronii acestui strat transmit mai departe valorile primite neuronilor din următorul strat, în general ascuns, cu care sunt conectaţi.

#### **Straturi ascunse**

Majoritatea reţelelor neuronale cuprind mai multe straturi ascunse. Fiecare neuron din straturile ascunse primeşte unul sau mai multe intrări de la stratul precedent, aplică transformări acestora, corespunzător ponderilor arcelor şi adună rezultatele ce formează input-ul pentru următorul strat neuronal cu care este conectat. Numărul neuronilor dintr-un strat este egal cu numărul de ieşiri ale acestuia.

#### **Straturi de ie**ş**ire**

Reţelele neuronale cuprind un singur strat de ieşire care primeşte date de la ultimul strat ascuns, sau de la stratul de intrare dacă nu există straturi ascunse, şi reprezintă finalul reţelei. Numărul de neuroni al acestui strat este egal cu numărul de rezultate dorite. Spre exemplu, dacă cerinţa este de a clasifica intrări în cinci clase, startul de ieşire va avea 5 neuroni, unde fiecare neuron n va prezice probabilitatea ca input-ul reţelei să aparţină clasei cu numărul n.



*Figura 2.3 Straturile unei reţele neuronale artificiale [34]*

### 2.2.3 Straturi neuronale

#### Linear sau Dense

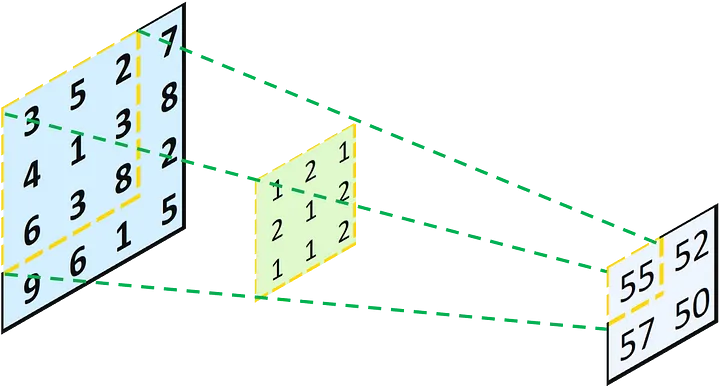
Straturile Linear sunt componenta de baza a reţelelor neuronale, ce, aşa cum le spune numele, aplică o transformare liniara (Ecuaţia 2.4) asupra datelor primite ca input. Scopul lor este de a învăţa ponderile modelului.

*Ecuaţia 2.4 Ecuaţia unei transformări liniare*

#### Convoluţional

Straturile convoluţionale sunt unitatea de baza a reţelelor neuronale convoluţionale. Ele conţin un număr de nuclee de o dimensiune dată, sub formă de matrici, ce reprezintă setul de parametrii antrenabili. Primind date de la straturile anterioare sau direct din baza de date, ele se suprapun peste porţiuni din input şi realizează înmulţirea element cu element cu acestea, apoi rezultatele sunt adunate formând o singură valoare. Astfel, pentru un nucleu de dimensiune 3x3 suprapus peste prima porţiune din input, se vor realiza 9 înmulţiri, apoi rezultatele vor forma o sumă (Figura 2.5).

Fiecare nucleu va produce în final o „hartă” doi-dimensională de valori, iar totalitatea acestora va reprezenta output-ul stratului convoluţional. Înălţimea şi lăţimea nucleelor sunt în general mult mai mici decat ale input-ului.



*Figura 2.5 O convoluţie aplicată pe o parte de dimensiune 3x3 din input cu un nucleu 3x3 [13]*

Pentru a defini mişcarea nucleelor asupra input-ului, pe lângă dimensiunea acestora, sunt stabilite două valori:

* Pasul de deplasare, sau stride - La început, nucleul este plasat în partea din stanga-sus a obiectului pe care este aplicată convoluţia. Se va realiza înmulţirea matriceală, iar apoi nucleul se va deplasa în dreapta, respectiv în stânga-jos dacă se va ajunge la „finalul” obiectului. În această repoziţionare, nucleu poate face unul sau mai mulţi paşi, acest număr fiind dat ca parametru stratului convoluţional.
* Umplerea, sau padding - Înainte de aplicarea convoluţiilor, uneori este de preferat adăugarea unor „borduri” matricei de input pentru gestionarea dimensiunilor hărţii de valori rezultată sau pentru conservarea a diverse trăsături.

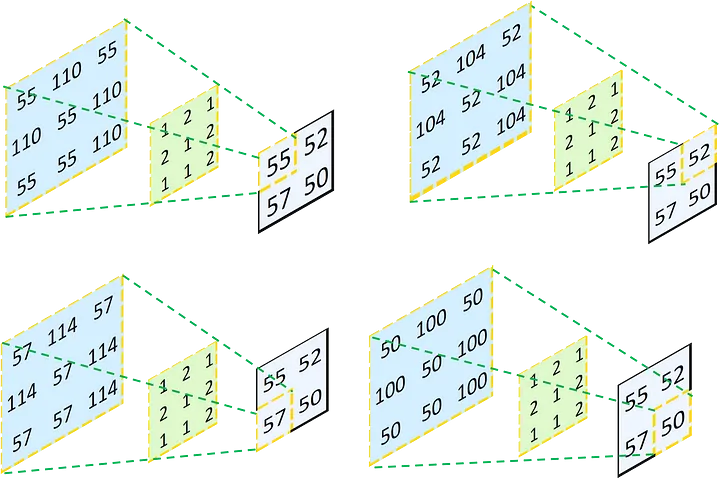
Înălţimea şi lăţimea obiectului obţinut în urma aplicării straturilor convoluţionale sunt identice sau mai mici decât cele iniţiale, capturând anumite trăsături de bază ale sale.

#### 

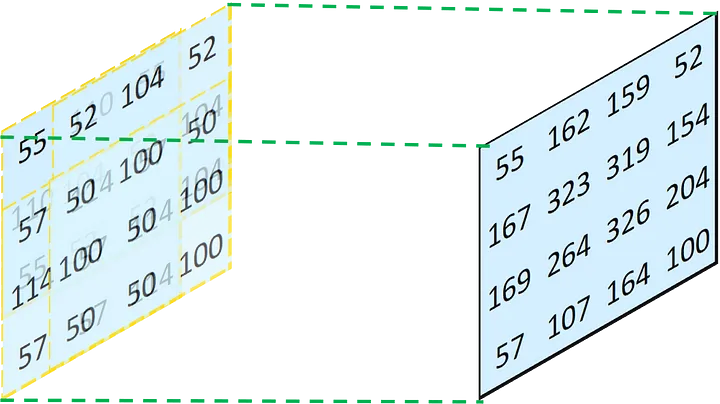
#### Convoluţional transpus

Straturile de convoluţie transpusă funcţionează tot pe baza nucleelor, a parametrilor învăţabili şi a valorilor stride şi padding ca în cazul convoluţiilor simple, însă pot fi privite ca fiind inversul acestora.

În loc ca nucleele să fie înmulţite cu porţiuni din input de aceeaşi dimensiune, acestea se vor înmulţi cu câte o valoare din input, formând matrici de dimensiunea nucleului (Figura 2.6). Apoi, acestea vor fi la rândul lor suprapuse, respectând poziţia elementelor cu care au fost formate şi cei doi parametrii, iar elementele lor vor fi adunate între ele (Figura 2.7). Rezultatul va fi tot o hartă doi-dimensională de valori, însă dimensiunile acesteia vor fi identice sau mai mari decât cele ale obiectului de intrare.



*Figura 2.6 Multiplicarea fiecărui element din input cu fiecare valoare din nucleu (de la dreapta la stânga) [13]*

**

*Figura 2.7 Combinarea celor 4 straturi rezultate şi însumarea valorilor care se suprapun [13]*

Sunt folosite pentru a captura trăsături ale obiectelor, dar şi pentru proprietatea lor de a le redimensiona.

Straturile convoluţionale şi convoluţionale transpuse sunt folosite datorită avantajelor lor, trei dintre ele fiind descrise în ‘Convolutional Neural Networks, Explained’ [21], şi anume:

* *Interacţiunea rară (sparse interaction)*

În loc să conecteze fiecare neuron la toţi neuronii din stratul anterior, ca în reţelele neuronale simple, straturile convoluţionale conectează fiecare neuron la doar o parte din datele de intrare ale stratului. Motivul acestei metode este susceptibilitatea că pixelii învecinaţi împart trăsături comune relevante. Astfel, sunt capturate dependenţe spaţiale şi este redusă memoria necesară reţinerii parametrilor.

* *Partajarea parametrilor*

Pentru crearea hărţilor de valori, neuronii stratului convoluţional sunt constrânşi să folosească acelaşi set de ponderi în mai multe regiuni. Astfel, se reduce semnificativ numărul de parametrii ai reţelei, iar aceasta devine mai eficientă şi mai uşor de antrenat. De asemenea, permite modelului să generalizeze anumite trăsături.

* *Invarianţa la translatare*

Pentru că nucleele folosesc aceleaşi ponderi, modelul este capabil să detecteze caracteristici similare situate în diferite locaţii. De asemenea, dacă apare o schimbare în input,

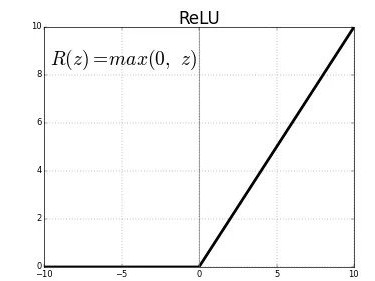
aceasta se va reflecta în output în acelaşi mod.

## 2.3 Funcţii de activare

În problemele pentru care sunt folosite reţele neuronale vrem în general ca valoarea finală calculată să se încadreze într-un anumit interval numeric. Din cauza multitudinii de parametrii pe care îi conţin straturile de neuroni şi a operaţiilor aplicate, s-ar putea ca output-ul final să nu se încadreze între limitele dorite. Soluţia este folosirea funcţiilor de activare, care normalizează datele pe care le primesc în anumite intervale precum [-1, 1] sau [0, 1]. Ele sunt aplicate în general pe fiecare dintre ieşirile straturilor folosite, putând fi utilizate funcţii diferite pentru paşi diferiţi.

### 2.3.1 ReLU (Rectified Linear Unit)

Funcţia ReLU (Figura 2.8) transformă valorile negative în 0, iar pe cele pozitive le lăsa neschimbate. Astfel, intervalul valorilor rezultate va fi [0, ∞).



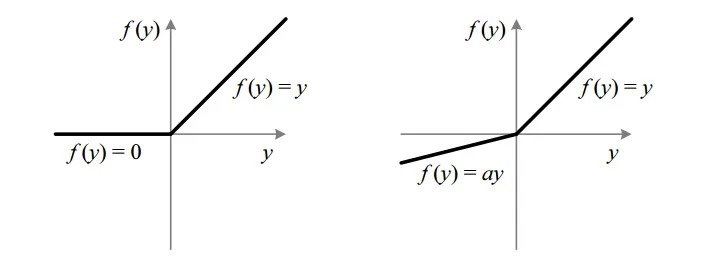
*Figura 2.8 Formula şi graficul funcţiei de activare ReLU [35]*

Transformarea în 0 a valorilor negative ar putea influenţa negativ modelul, deoarece acesta nu va putea învăţa cum să le mapeze corect.

### 

### 2.3.2 LeakyReLU (Leaky Rectified Linear Unit)

În încercarea de a rezolva problema funcţiei ReLU, Leaky ReLU (Figura 2.9) transformă valorile negative prin înmulţirea lor cu constanta 0,01. Astfel, intervalul valorilor rezultate va fi (-∞, ∞).

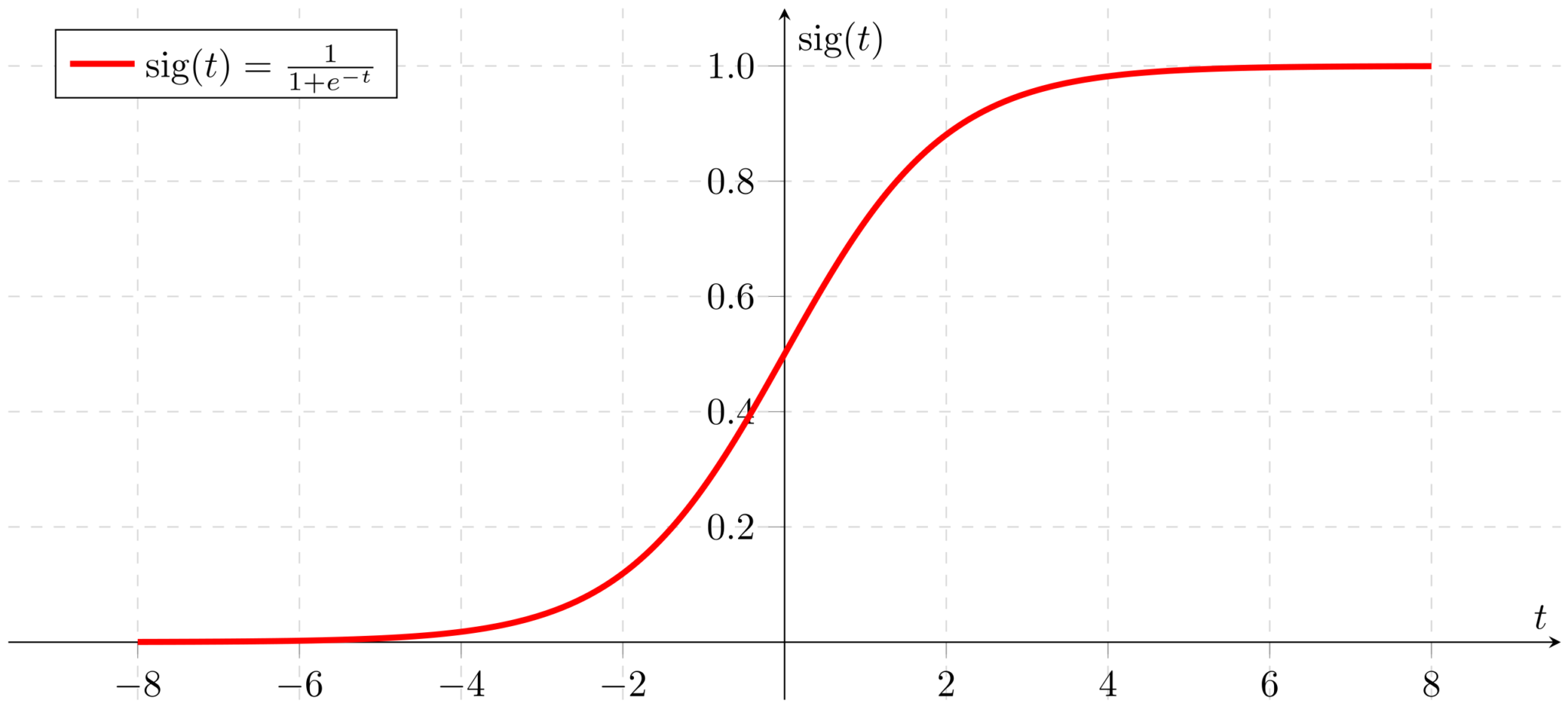


*Figura 2.9 ReLU vs Leaky ReLU [35]*

### 

### 2.3.3 Sigmoid sau funcţia de activare logistică

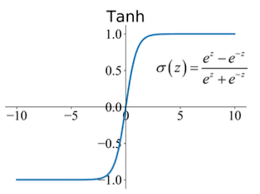
Intervalul de ieşire al funcţiei Sigmoid (Figura 2.10) este (0, 1) , aceasta fiind folosită deseori pentru probleme în care rezultatul trebuie să fie o probabilitate.



*Figura 2.10 Formula şi graficul funcţiei de activare Sigmoid [8]*

### 2.3.4 Tanh sau funcţia de activare tangentă hiperbolică

Spre deosebire de funcţia Sigmoid, valorile lui Tanh (Figura 2.11) sunt în intervalul (-1, 1). Astfel, valorile negative vor fi mapate în valori negative. Este folosită în general pentru probleme de clasificare binară.



*Figura 2.11 Formula şi graficul funcţiei de activare Tanh [28]*

## 2.4 Alte tipuri de straturi

### 2.4.1 BatchNorm

Straturile batch normalization sau BN sunt o metodă de a face antrenarea reţelelor neuronale mai rapidă şi mai stabilă. Ele normalizează datele din straturile ascunse folosind media (Ecuaţia 2.12.1) şi varinţa (Ecuaţia 2.12.2) setului curent de obiecte. Astfel, fiecare ieşire a unui neuron va avea o distribuţie standard normală în cadrul setului curent de date.

*Ecuaţiile 2.12.1 şi 2.12.2 Media si varianţa setului de date*

Conform articolului ‘Batch normalization in 3 levels of understanding’ [10] ce descrie modul de funcţionare al acestui tip de strat şi analizează mai multe experimente, două dintre marile avantaje ale folosirii batch normalization sunt:

* Convergenţă mai rapidă şi mai bună (cu acurateţe mai mare);
* Permite folosirea unei rate de învăţare mai mare fără a compromite convergenţa.

### 2.4.2 Flatten (aplatizare)

„Straturile Flatten transformă intrări multidimensionale în ieşiri uni-dimensionale şi sunt în general folosite pentru trecerea de la straturi convoluţionale la straturi liniare” [37].

### 2.4.3 GaussianNoise

Stratul GaussianNoise este folosit pentru a adăuga un nivel de „zgomot” unui obiect, în diverse scopuri precum creşterea dificultăţii de recunoaştere a sa, spre exemplu în cazul în care una din componentele reţelei se antrenează mult mai rapid decat o altă componentă. „Zgomotul” va avea o distribuţie normală cu media 0 şi deviaţie standard (Ecuaţia 2.13) dată. Pe durata antrenării, cele două variabile se pot modifica.

*Ecuaţia 2.13 Formula deviaţiei standard .*

GaussianNoise nu este un strat clasic din domeniul machine learning, ci este o implementare adaptată, inspirată din ‘Improved Techniques for Training GANs’ [38].

## 2.5 Funcţii obiectiv

Funcţiile obiectiv sunt folosite în problemele bazate pe inteligenţă artificială pentru a măsura performanţa modelelor şi progresul lor de-a lungul a diverse faze. Mai precis, ele măsoară o diferenţă folosind diverse metrici între valoarea obţinută şi cea dorită. Cu cât rezultatul funcţiei este mai mic, cu atât modelul se apropie mai mult de scopul propus.

În problemele de învăţare artificială, după fiecare pas al modelului este calculată valoarea funcţiei obiectiv care este trimisă drept „feedback” reţelei. Aceasta îşi ajustează ulterior parametri bazat pe scorul primit.

### 2.5.1 Funcţia obiectiv de identitate

Funcţia obiectiv de identitate se aplică în cazurile când, după o serie de transformări, rezultatul trebuie să fie cât mai similar sau chiar identic cu intrarea. Se foloseşte în probleme precum eliminarea zgomotului din imagini sau clarificarea imaginilor, transmiţând modelului că nu trebuie să devieze prea mult de la conţinutul original.

Pentru calcularea acesteia este în general folosită norma L1 (Ecuaţia 2.14), numită şi media erorii absolute.

.

*Ecuaţia 2.14 Norma L1 sau media erorii absolute*

### 

### 2.5.2 Funcţia obiectiv de conţinut

Funcţia obiectiv de conţinut are scopul de a conserva conţinutul semantic al unui obiect, asemănător cu funcţia obiectiv de identitate. Spre deosebire de aceasta, se foloseşte în probleme precum transferul de stil neural, unde ieşirea trebuie să păstreze conţinutul intrării adoptând însă alt stil. Această funcţie încurajează modelul să captureze structura de baza şi obiectele prezente în imaginea de intrare.

Pentru calculul funcţiei obiectiv de conţinut se foloseşte în general media erorii pătrate - MSE (Ecuaţia 2.15).

.

*Ecuaţia 2.15 Media erorii pătrate*

### 2.5.3 Funcţia obiectiv prin potrivirea caracteristicilor

Funcţia obiectiv prin potrivirea caracteristicilor are ca scop stabilizarea procesului de antrenare a reţelelor generative adversariale, unde generatorul încearcă să producă obiecte cât mai realiste.

Calculul său se realizează prin compararea caracteristicilor extrase din obiecte reale cu caracteristici extrase din obiecte generate. Este folosit discriminantul pentru a captura „hărţi” ale trăsăturilor, apoi este măsurată diferenţa dintre acestea, în general folosind media erorii pătrate. Pentru că vrem să măsurăm distanţa între trăsături care ţin mai mult de stil decât de conţinut, “hărţile” sunt extrase din momente intermediare ale procesării realizate de discriminant.

### 2.5.4 Funcţia obiectiv adversarială

Funcţia obiectiv adversarială este funcţia clasică utilizată în contextul reţelelor generative adversariale fiind cunoscută şi sub numele de minmax (Ecuaţia 2.16).

.

*Ecuaţia 2.16 Funcţia minmax*

Denumirea sa are la bază ideea generală a reţelelor adversariale, şi anume că una dintre componente vrea să minimizeze cât mai repede valoarea funcţiei, iar cealaltă vrea să o maximizeze mereu.

### 2.5.5 Funcţia obiectiv contrastivă

Funcţia contrastivă (Ecuaţia 2.17) este folosită în problemele de reprezentare a datelor, încurajând obiecte similare, din aceeaşi distribuţie, să fie aproape unul de altul în spaţiul de reprezentare, şi împingând obiecte disimilare departe unul de altul.

,

*Ecuaţia 2.17 Funcţia contrastivă, unde d este distanţa dintre cele doua intrari,*

*m este un punct fixat, iar y are valoare binară şi reprezintă*

*similaritatea (0) sau disimilaritatea (1) obiectelor date.*

Astfel, pentru doua intrări aparţinând aceleiaşi distribuţii, primul termen încurajează distanţa să fie mică. În caz contrar, cel de-al doilea termen împinge distanţa cât mai aproape sau chiar peste valoarea .

### 2.5.6 Funcţia obiectiv triplă

Funcţia obiectiv triplă (Ecuaţia 2.18) primeşte trei valori de intrare: două puncte „asemănătoare”, a şi p, şi un punct disimilar celor două, n. Scopul său este să împingă punctul negativ n cât mai departe de celelalte două, care trebuie la rândul lor să păstreze o distanţă mică între ele. Funcţia este folosită drept obiectiv în contexte similare funcţiei contrastive.

.

*Ecuaţia 2.18 Funcţia obiectiv triplă*

### 2.5.7 Funcţia obiectiv siameză

Funcţia obiectiv siameză (Ecuaţia 2.19) este o combinaţie a celor două funcţii prezentate anterior şi se foloseşte în special în cazul reţelelor siameze.

*Ecuaţia 2.19 Funcţia siameză unde d este distanţa dintre două*

*reprezentări pentru care se calculează valoarea funcţiei.*

Pentru distanţe mici, se apropie de , iar pentru distanţe mari se apropie de 0. Aşadar, spre deosebire de obiectivele prezentate mai sus ale căror intervale de valori sunt pozitive, funcţia siameză va rezulta doar valori negative.

### 2.5.8 Funcţia obiectiv de similaritate cosinus

Funcţia obiectiv de similaritate cosinus (Ecuaţia 2.20) încurajează similaritatea dintre doi vectori daţi.

*Ecuaţia 2.20 Funcţia de similaritate cosinus*

Această funcţie este folosită tot în scopul reprezentării datelor. Intervalul de valori al rezultatelor este [-1, 1], unde 1 semnifică vectori cu aceeaşi direcţie sau identici, 0 semnifică vectori ortogonali sau necomparabili, iar -1 semnifică vectori cu direcţii opuse sau deloc similari.

## 2.6 Rata de învăţare

Rata de învăţare a algoritmilor de machine learning este rata sau pasul cu care aceştia îşi actualizează parametrii în timpul procesului de antrenare, cu scopul de a minimiza funcţia obiectiv.

Modelele de machine learning conţin în general două tipuri de parametri:

* Hiperparametri - variabile ale căror valoare este stabilită apriori procesului de învăţare, pentru a stabili modul în care algoritmii învaţă şi pentru a regla performanţa acestora;
* Parametri învăţabili - parametrii pe care algoritmii îi estimează în timpul procesului de învăţare, bazat pe datele primite.

După fiecare iteraţie sunt calculaţi gradienţii modelului, care reprezintă direcţia şi magnitudinea celei mai abrupte creşteri sau scăderi a funcţiei obiectiv. Rata de învăţare determină ponderea din gradient cu care parametrii sunt actualizaţi. Scopul actualizării parametrilor este de a „muta” valorile acestora în direcţia celei mai abrupte coborâri pentru a ajunge la valorile optime care să minimizeze funcţia obiectiv.

În cazul unei rate prea mici, ajustările vor fi minime, iar procesul de învăţare poate dura foarte mult timp şi poate rămâne blocat în soluţii suboptime.

O rată de învăţare ridicată determină ajustări semnificative, însă ar putea exista riscul ca parametrii să depăşească valorile optime, ceea ce poate duce la instabilitate şi divergenţă în timpul procesului de antrenare.

Alegerea unei rate adecvate de învăţare poate fi dificilă şi depinde de diverşi factori, precum problema pe care încercăm să o rezolvăm, setul de date primit, arhitectura modelului şi optimizatorii folosiţi. De obicei, rata de învăţare este determinată prin experimentare cu diferite valori ale sale.

## 2.7 Optimizatori

Optimizatorii sunt algoritmi folosiţi pentru a aplica actualizarea parametrilor, având o rată de învăţare dată. Există diferite moduri prin care optimizatorii folosesc rata de învăţare. În unele cazuri este menţinută aceeaşi valoare a sa pe tot parcursul procesului de antrenare, alteori aceasta este ajustată dinamic bazat pe diferite metrici sau euristici.

### Optimizatorul Adam (Adaptive Moment Estimation)

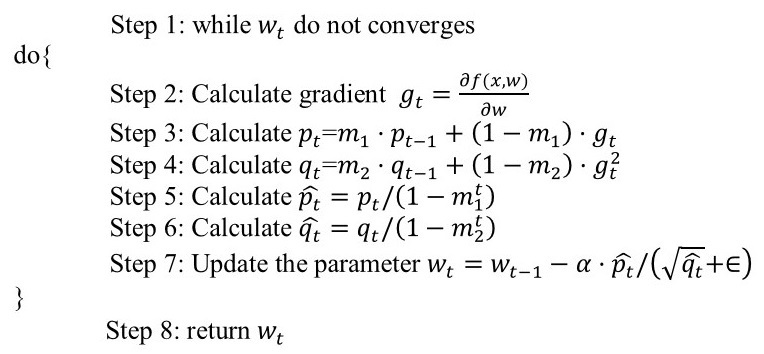
Algoritmul Adam (Figura 2.21) este unul dintre cei mai folosiţi optimizatori în machine learning, şi este optimizatorul folosit în general pentru reţelele generative adversariale, descris atât în ‘Generative Adversarial Nets’ [9], cât şi în ‘Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks’ [1].

Conform ‘Intuition of Adam Optimizer’ [5], Adam „combină ideea optimizatorilor bazaţi pe momentum cu algoritmul RMSP”. Tactica sa principală este de a menţine o rată de învăţare separată pentru fiecare parametru. Ţine cont atât de primul moment (media) cât şi de al doilea moment (varianţa necentrată) a gradienţilor folosind medii exponenţiale mobile. Acest lucru îi permite să ajusteze adaptiv ratele de învăţare pentru diverşi parametrii bazat pe gradienţi anteriori.

Conform aceluiaşi articol, „momentum este un algoritm folosit pentru accelerarea coborârii pe gradient prin calcularea mediei ponderate exponenţiale a derivatelor. Folosirea mediilor face algoritmul să conveargă către minim într-o manieră mai rapidă.”

RMSP „este un algoritm de învăţare adaptivă care încearcă să îmbunătăţească algoritmul AdaGrad. În loc de a calcula suma cumulativă a pătratelor gradienţilor ca în AdaGrad, calculează media exponenţială mobilă”, conform articolului menţionat anterior.

Un avantaj major al algoritmului Adam, descris în ‘ML | ADAM (Adaptive Moment Estimation) Optimization’ [6], este faptul că „ajustarea parametrilor este total invariantă redimensionării gradientului, algoritmul convergând chiar dacă funcţia obiectiv se schimbă în timp. Dezavantajul acestei tehnici este că necesită calcularea derivatelor de grad doi ceea ce rezultă în costuri ridicate”.



*Figura 2.21 Algoritmul Adam [6]*

Explicit, la pasul 1 se verifică dacă modelul a convers sau nu, fiind ponderile la momentul . Dacă răspunsul este negativ, la pasul 2 sunt calculaţi gradienţii , de la pasul actual, unde este funcţia obiectiv.

Paşii 3 şi 4 calculează primul moment , respectiv al doilea moment , la momentul de timp , relativ la valorile obtinute. Iniţial, acestea au valoarea 0. si sunt constante, parametrii ai mediilor mobile, în general având valoarea 0,9.

Pentru că atât , cât şi au iniţial valoarea 0, şi pentru că şi au valori apropiate de 1, şi au tendinţa să se apropie de 0. Optimizatorul corectează această problemă prin două formule, calculate la paşii 5 şi 6, ce ajută de asemenea la controlul ponderilor atunci când minimul global este atins, pentru a nu oscila foarte departe de el.

Pasul 7 descrie ecuaţia generală a optimizatorului de ajustare a parametrilor, unde este rata de învăţare, iar epsilon este o constantă mică, în general , folosită pentru a evita împărţirea la 0.

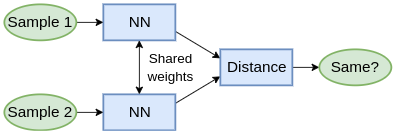
Algoritmul este reluat până când modelul converge.

## 2.8 Reţeaua Siameză (Siamese Network)

O reţea siameză (Figura 2.22) primeşte ca input două obiecte, în general imagini. Are o singură componentă, un codificator ce transformă cele doua input-uri în câte un vector latent de dimensiune fixă dată. Procesul de transformare al imaginilor se face „în paralel”, de aici şi denumirea de reţea „siameză”, însemnând că la fiecare pas sunt aplicate aceleaşi ponderi pentru a calcula cei doi vectori.

Scopul reţelei este de a „măsura” distanţa dintre cele doua input-uri, astfel că dacă acestea aparţin unui acelaşi domeniu, distanţa dintre vectorii corespunzători trebuie să fie mică, iar dacă provin din doua clase diferite, distanţa trebuie sa fie mare.

Conform articolului ‘A New Way to look at GANs’ [17], reţelele siameze sunt folosite în cazuri de „one-shot learning”, adică au abilitatea de a învăţa caracteristici ale obiectelor cu doar un exemplu. Majoritatea clasificatorilor au nevoie de antrenare pe seturi mari de date pentru a obţine performanţe bune. Secretul performanţei acestui tip de reţele stă în funcţia sa obiectiv. În funcţie de cerinţe şi implementare, două funcţii utilizate des drept obiective pentru reţelele siameze sunt funcţia contrastivă şi funcţia triplă prezentate în subcapitolul *2.5 Funcţii obiectiv*.



*Figura 2.22 Arhitectura unei Reţele Siameze [26]*

## 2.9 Reţele Generative Adversariale (Generative Adversarial Networks)

Reţelele generative adversariale sunt o metodă de generare de obiecte ce bazându-se pe concepte de deep learning. Sunt metode de învăţare nesupervizată ce implică învăţarea automată de trăsături sau tipare dintr-un set de date cu scopul de a recrea ulterior noi exemple cât mai indistinctibile faţă de acesta.

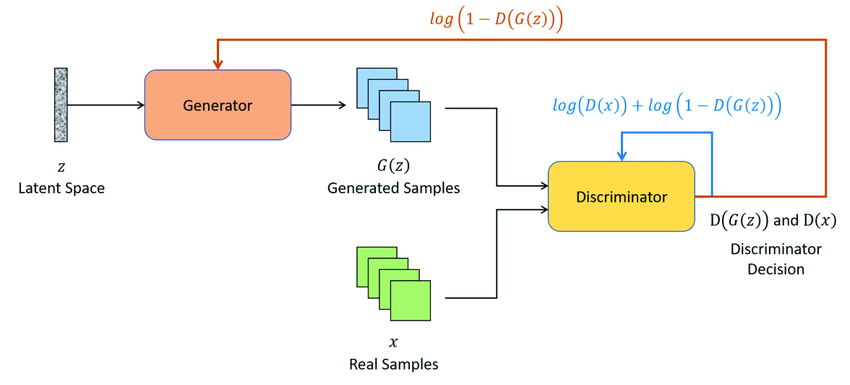
Arhitectura (Figura 2.23) este compusă din două componente, două modele diferite, numite în literatură generator şi discriminant, ce se antrenează reciproc astfel:

* Generatorul primeşte ca input un vector latent z cu valori dintr-o distribuţie normală, pe care încearcă să îl transforme într-un obiect de tipul şi dimensiunile celor din setul de date;
* Discriminantul încearcă să clasifice exemplele primite ca fiind originale (aparţinând setului de date iniţial) sau prelucrate (generate de generator);
* Pe parcursul etapei de antrenare, generatorul încearcă să creeze exemple cât mai autentice pentru a “păcăli” discriminantul;
* Pe de altă parte, discriminantul se antrenează pentru a distinge din ce în ce mai bine setul de date autentic faţă de obiectele generate;
* O reţea de tip GAN se presupune a fi “bine antrenată” atunci când generatorul ajunge să creeze exemple atât de autentice încât discriminantul ghiceşte cu o probabilitate de 0,5 dacă acestea sunt reale sau false; în acest caz generatorul converge;
* Aşadar, generatorul şi discriminantul joacă un joc de tip minmax.

Din lucrarea lui Goodfellow intitulată “Generative Adversarial Nets” [9] aflăm că funcţia de pierdere a unui model GAN este chiar funcţia minmax descrisă anterior.

Discriminantul va analiza atât obiecte reale cât şi obiecte prelucrate de generator, încercând să maximizeze funcţia , unde x vor fi exemplele reale, iar cele generate, şi este probabilitatea întoarsă de discriminant.

Generatorul va încerca să minimizeze probabilitatea ca discriminantul să prezică că exemplul generat este fals, deci va încerca să minimizeze funcţia



*Figura 2.23 Arhitectura unei Retele Generative Adversariale [33]*

O variaţie a funcţiei obiectiv folosită în practică schimbă scopul generatorului, şi anume acesta va încerca să maximizeze probabilitatea ca discriminantul să prezică că exemplele generate sunt reale, deci va maximiza

În general, cea mai simplă variantă de reţea generativă adversarială foloseşte atât pentru generator cât şi pentru discriminant straturi dense împreună cu ReLU sau LeakyReLU ca funcţii de activare.

Reţelele de tip GAN sunt specifice problemelor de generare de obiecte noi şi realiste şi au fost folosite în sarcini precum generarea de artă, crearea de imagini sau videouri sintetice pentru diferite scopuri. Pornind de la ideea de bază a GAN-urilor, au fost create diverse derivate ale sale, adaptate pe noi scopuri.

Articolul ‘Generative Adversarial Network (GAN)’ [4]rezumă foarte cuprinzător câteva dintre avantajele şi dezavantajele pe care le prezintă reţelele generative adversariale.

Avantaje:

* Posibilitatea de a genera obiecte noi;
* Realismul obiectelor generate;
* Procesul de învăţare este nesupervizat;
* Versatilitate.

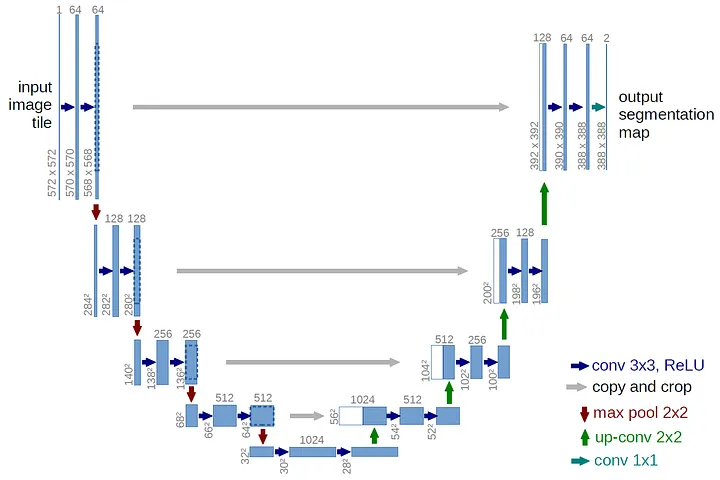
Dezavantaje:

* Instabilitatea procesului de antrenare: există un risc ridicat de instabilitate sau imposibilitate de a converge;
* Este necesară o putere computaţională ridicată;
* Overfitting: reţelele pot face overfit pe setul de antrenare, producând apoi numai obiecte mult prea similare acestora.

## 2.10 Arhitectura UNET

Apărută în 2015, “arhitectura UNET este o arhitectură de reţea sub formă de U de tip codor-decodor, care constă în patru blocuri de codificare şi patru blocuri de decodificare ce sunt conectate între ele printr-o trecere” [24] (Figura 2.24). În partea de codificare sunt înjumătăţite dimensiunile spaţiale şi dublat numărul de filtre la fiecare bloc. Invers, în partea de decodare, este înjumătăţit numărul de canale şi dublat numărul de dimensiuni spaţiale.

În general, UNET este folosită în probleme de computer vision, pentru segmentarea imaginilor sau extragerea anumitor trăsături.



*Figura 2.24 Arhitectura UNET [24]*

### Codorul

Prin arhitectura sa, codorul extrage trăsături din imagini realizând o abstractizare a acestora. Cum este descris şi în articolul ‘What is UNET?’ [24], straturile sale clasice sunt:

* Convolutionale cu nucleu de dimensiune 3x3;
* Urmate de funcţia de activare ReLU pentru a introduce non-liniaritate datelor;
* Max-pooling de dimensiune 2x2 pentru înjumătăţirea dimensiunilor spaţiale.

### Trecerea

Straturile de trecere conectează codorul de decodor. Arhitectura clasică conţine două straturi convoluţionale cu nucleu de dimensiune 3x3, fiecare dintre ele fiind urmată de funcţia de activare ReLU.

### Decodorul

Decodorul foloseşte reprezentarea abstractă a imaginilor pentru a genera o „mască semantică segmentată”. Conform articolului menţionat [24], arhitectura sa consta în:

* Straturi de convoluţie transpusă cu nucleu de dimensiune 2x2;
* Straturi de convoluţie 3x3, urmate de funcţia de activare ReLU;
* O convoluţie de dimensiune 1x1, urmată de o activare Sigmoid, ce va constitui clasificarea pixelilor.

Arhitectura UNET este „construită pentru a învăţa dintr-un număr mai mic de date de antrenare”, fiind capabilă de abstractizare.

# 

# Capitolul 3

# Algoritmi

## 3.1 Setul de date

**GTZAN - Clasificarea genurilor muzicale**

Am folosit setul de date *GTZAN - Clasificarea genurilor muzicale [12]* ce conţine câte 100 de melodii din 10 genuri muzicale diferite, având lungime de 30 de secunde. Am considerat că pentru început, setul de date este potrivit pentru a crea „filtrele” de gen muzical.

Pentru că acurateţea recunoaşterii muzicale şi a schimbării de stil depinde direct de „puritatea” setului de date am prelucrat melodiile în spectrograme, având avantajele menţionate în Capitolul 2, subcapitolul *2.1.2 Metode de reprezentare a sunetului - Spectrograma*, în special pentru eliminarea zgomotului. Am aplicat apoi scara mel, fiind reprezentativă pentru cum percep oamenii frecvenţe, descrisă în acelaşi subcapitol. În continuare mă voi referi la mel spectrograme drept spectrograme.

## 3.2 Crearea spectrogramelor

* Audio-urile folosite au o durată de aproximativ 30 de secunde şi sunt eşantionate la frecvenţa 22050 Hz, deci vor avea aproximativ 661500 de eşantioane;
* Am creat spectrograma întregii melodii, pentru fiecare melodie, ce va avea forma unde *t* este numărul de „ferestre” pe care s-a aplicat Transformata Fourier, iar *mels* este numărul filtrelor mel aplicate în crearea spectrogramei;
* Având putere computaţională mică şi audio-uri de dimensiuni destul de mari, ce implicit generează spectrograme largi, am ales să lucrez în final pentru algoritmul de schimbare de gen muzical cu „bucăţi” din spectrograme de dimensiune 32x32. Melodiile sunt *Mono*, deci imaginile rezultate vor avea un singur canal (vor fi grayscale);
* Am salvat toate spectrogramele rezultate în format .JPEG, grayscale, formând astfel setul de date;
* Algoritmul de recunoaştere muzicală va avea ca input spectrogramele întregi.

## 3.3 Algoritmul de schimbare de gen muzical

### 3.3.1 Introducere în problema transferului de stiluri

Intuiţia pentru acest tip de algoritm este de a analiza mai multe obiecte (imagini, melodii, video-uri) ce aparţin aceluiaşi stil, apoi de a transfera o parte din trăsături unui alt obiect ce aparţine unui domeniu diferit. În general, când vine vorba de “transfer de stiluri”, aceasta fiind denumirea folosită în lucrările de specialitate, subiectul este abordat şi rezolvat în diverse moduri pentru imagini, în diferite contexte cum ar fi schimbarea diferitelor trăsături ale feţei, a schemei cromatice, sau a stilului unei picturi. În domeniul muzical însă, subiectul este rar dezbătut şi nu se cunoaşte o metodă sigură sau exactă, deoarece formatul audio presupune mult mai mulţi parametrii de luat în considerare faţă de o imagine.

Un algoritm ce rezolvă această problemă ar putea fi folosit atât la schimbarea genului muzical, cât şi la schimbarea vocii unui povestitor într-o înregistrare, sau a tonalităţii unei voci asociate unui dispozitiv tip GPS. În continuare voi prezenta o serie de algoritmi de machine learning ce ar putea rezolva această cerinţă, pe care i-am folosit mai departe în implementarea aplicaţiei.

### 3.3.2 Neural Style Transfer

În lucrarea ‘A Neural Algorithm of Artistic Style’ [15] din 2015, este descris un algoritm de deep learning ce are scopul de a crea noi imagini prin combinarea conţinutului unei imagini cu stilul alteia.

Ideea principală este de a utiliza o reţea neuronală preantrenată pentru a extrage conţinutul, respectiv stilul din cele doua imagini furnizate ca input. Apoi, reţeaua va furniza imaginea dorită.

Reţeaua poate fi folosită în contexte precum transformarea unei picturi în stilul altui pictor, adăugarea de diferite texturi în imagini, schimbarea tonalităţilor cromatice din imagini. Un exemplu concret în care este folosită sunt filtrele de imagini de pe reţelele de socializare, ce pot face imaginile sa pară retro, vintage, sau să pară că sunt făcute cu aparate polaroid.

În scopul transferului de genuri muzicale între melodii însă, algoritmul nu este aplicabil deoarece spectrogramele au acelaşi „stil”, sunt imagini grayscale. Diferenţa dintre genurile muzicale este făcută de instrumentele folosite şi de frecvenţele atinse, fapt ce se va reflecta în spectrograme prin diferenţe între intensitatea culorii pixelilor, în anumite zone specifice. Aşadar, un model mai complex este necesar.

Modelele studiate şi prezentate de mine sunt algoritmi de învăţare nesupervizată, întrucât doresc modificarea unui obiect deja existent, deci învăţarea unor trăsături, nu clasificarea sa. Astfel, unii dintre cei mai potriviţi algoritmi pentru această problema se bazează pe Reţele Generative Adversariale (GAN).

### 3.3.3 Deep Convolutional GAN - DCGAN

Deep Convolutional GANs sunt reţele generative adversariale care folosesc straturi convoluţionale în loc de cele dense. Astfel, conform lucrării originale ‘Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks’ [1], cele doua modele sunt formate din:

-> Generator:

* straturi convoluţionale;
* straturi BatchNorm;
* LeakyReLU ca funcţie de activare.

-> Discriminant:

* straturi de convoluţie transpusă;
* straturi BatchNorm;
* ReLU ca funcţie de activare.

Vectorul latent z este transformat după formatul datelor de input cu ajutorul straturilor de convoluţie care permit redimensionarea sa. Funcţia obiectiv este cea folosită şi în cazul GAN-urilor simple şi anume funcţia adversarială. Conform lucrării [1], pentru cele două componente se folosesc optimizatori Adam, cu rata de invaţare 0,0002 şi beta1 = 0,5.

Avantajele DCGAN faţă de GAN provin din folosirea straturilor convoluţionale şi vor fi deci cele discutate în Capitolul 2, subcapitolul *2.2.3 Tipuri de straturi neuronale*.

În vederea îndeplinirii scopului de transfer de gen muzical, va trebui ca generatorul să primească drept input o spectrogramă, ci nu un vector latent. În mod tradiţional, setul de date pentru transferul “image-to-image” este format din imagini din stilul A şi corespondentul lor din stilul B. Totuşi, acest lucru este o mare limitare, pentru că acest tip de seturi de date sunt greu de găsit sau creat.

Setul de date pe care lucrez, prezentat la începutul acestui capitol, subcapitolul *3.1 Setul de date,* nu este împerecheat, adică generatorul nu va primi exemple corelate din stilul A şi stilul B, ci va primi un set de date din A şi un alt set de date din B, fără o legatură între elementele sale. Outputul va fi tot o spectrogramă, de aceleaşi dimensiuni ca cea de intrare.

În acest caz, generatorul va avea două scopuri:

* Cel iniţial, prezentat în subcapitolul *2.9 Reţele Generative Adversariale*, de a învăţa distribuţia sau pattern-ul setului de antrenare şi de a îl aplica pe intrarea primită;
* Pentru că inputul va fi o spectrogramă, deci un obiect cu conţinut nealeator, noul scop este de a păstra o corespondenţă de conţinut între imaginea de input x şi imaginea modificată G(X); mai exact, reţeaua trebuie să înveţe să aplice un stil pe un anume conţinut, astfel încât ieşirea să conţină aceleaşi elemente ca intrarea.

Pentru a înţelege mai bine cel de-al doilea punct voi prezenta un exemplu. Presupunem că avem o spectrogramă a unei melodii pop ce conţine o voce şi mai multe instrumente, şi vrem să o transformăm într-o spectrogramă rock care evidenţiază chitara şi tobele.

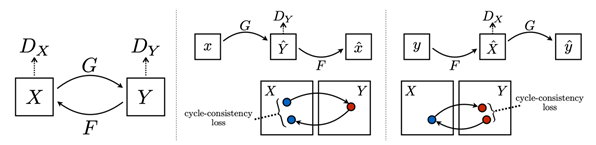
Generatorul trebuie să înveţe să conserve elementele cheie ale spectrogramei pop, adică vocea şi instrumentele muzicale, în timp ce încorporează şi elementele rock, adică chitara şi tobele.

În cele ce urmează mă voi referi la generator şi discriminant ca având structura prezentată în lucrarea originală despre reţelele generative adversariale convoluţionale [1].

### 3.3.4 CycleGAN

CycleGAN (Figura 3.1) este unul dintre primele modele folosite pentru realizarea transferului de stiluri pe date neîmperecheate. Spre deosebire de modelele GAN anterioare, input-ul său pentru toate componentele va fi constituit din obiecte din setul de date, nemaifiind folosit vectorul latent. În plus, CycleGAN este compus din două generatoare şi doi discriminanţi, astfel:

* Primul generator primeşte obiecte dintr-un domeniu A şi le transformă în stilul domeniului B;
* Al doilea generator primeşte obiecte din domeniul B şi le transformă în stilul domeniului A;
* Cei doi discriminanţi au rolul de a clasifica obiectele ca aparţinând domeniului B, respectiv A, şi de a da feedback primului, respectiv celui de-al doilea generator.



*Figura 3.1 Arhitectura CycleGAN [36]*

Spre deosebire de DCGAN, este adăugat un plus la funcţia de pierdere. CycleGAN se bazează pe „cycle consistency”, însemnând că pentru o intrare x dată primului generator, ieşirea corespunzătoare poate fi transformată de cel de-al doilea generator, iar rezultatul y ar trebui să fie similar lui x. Plusul vine din măsurarea diferenţei dintre x şi y folosind o funcţie obiectiv de identitate, unde diferenţa se va face între y şi x.

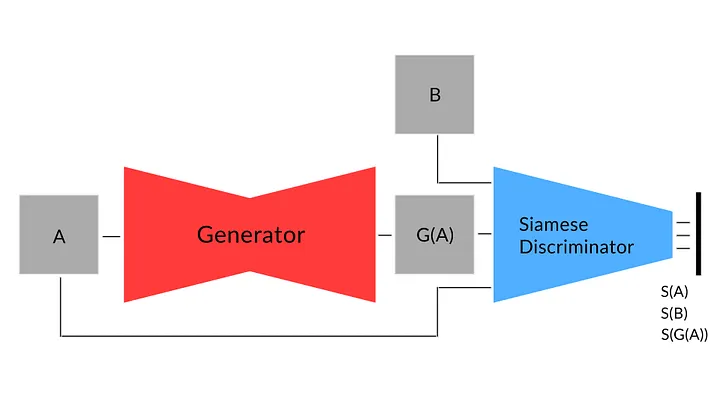
Totuşi, antrenarea unui model GAN este foarte scumpă din punct de vedere computaţional, iar CycleGAN este un model “dublu”, deci un dezavantaj mare al său este atât timpul de execuţie, cât şi memoria utilizată.

Un alt dezavantaj este dificultatea de optimizare a parametrilor, din cauza numărului mare al acestora. Pentru a îi aduce la valori benefice este necesar un set de date foarte mare.

În plus, CycleGAN se bazează pe diferenţa pixel-to-pixel între imagini, deci este un model destul de rigid. În cazul a două genuri muzicale foarte diferite între ele va fi greu de lucrat cu acest model în scopul transferului de gen muzical şi ar putea da rezultate nesatisfăcătoare din cauza „cycle consistency”.

### 3.4.5 Siamese Discriminator GAN (SDGAN)

O reţea SDGAN (Figura 3.2) foloseşte pe post de discriminant o reţea siameză, ce primeşte ca input perechi de obiecte reale şi generate şi are drept output vectori latenţi, la fel ca în cazul unei reţele siameze simple.



*Figura 3.2 Arhitectura Siamese Discriminator GAN [18]*

În algoritmi precum cei de recunoaştere facială, numărul de clase este egal cu numărul de feţe ce trebuiesc recunoscute. Aşadar, reţeaua încearcă să grupeze imagini ce conţin feţe cu trăsături asemănătoare cât mai aproape una de alta. Putem privi fiecare imagine ca un punct în spaţiul reţelei, numărul de dimensiuni de reprezentare fiind egal cu lungimea vectorilor generaţi.

În cazul unui GAN, singurele două clase sunt „real” sau „fals”, deci reţeaua SDGAN va încerca să grupeze toate imaginile ce provin din setul de antrenare cât mai aproape unele de altele, şi va aşeza grupul imaginilor generate mai departe.

Scopul generatorului, însă, este de a minimiza distanţa dintre cele două grupuri, ajungând să se amestece într-un singur grup. Input-ul generatorului va fi de aceasta data un anumit tip de obiect.

Pentru a măsura performanţa reţelei şi a antrena componentele sale se folosesc diferite combinaţii de funcţii obiectiv, atât dintre cele specifice reţelelor generative adversariale, cât şi dintre cele folosite în cazul reţelelor simeze simple. Scopul final al reţelei SDGAN va rămâne acelaşi ca scopul unei reţele GAN simplă.

Plecând de la ideea funcţiei obiectiv triple, pentru că în cazul transferului de gen muzical vor exista doar două clase, putem alege punctul ancoră fixat aleator. Pentru simplificarea calculelor poate fi alesa originea drept ancoră, care va reprezenta deci un centru relativ al spectrogramelor autentice.

Mai mult, preluând şi ideea funcţiei obiectiv contrastivă, putem alege o margine fixată m. Astfel, scopul va fi nu numai de a depărta imaginile neautentice cât mai mult de origine, ci şi de a le poziţiona pe cât posibil după această margine.

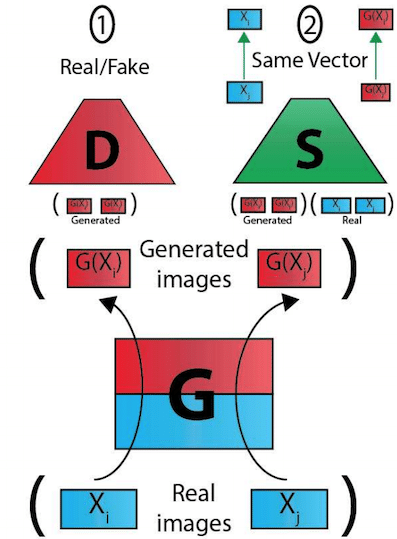
Se deduce că scopul generatorului va fi de a apropia toate imaginile de centru, “înăuntrul” marginii.

În plus, fiindcă inputul este reprezentat de spectrograme, vom vrea să conservăm conţinutul acestora, astfel că pentru două obiecte din domeniul A, A1 şi A2, şi transformările lor date de generator în domeniul B, AB1 şi AB2, vectorii corespunzători lor, generaţi de discriminantul siamez, vor trebui să aibă valori astfel încât distanţele dintre A1 şi A2, respectiv AB1 şi AB2 să fie asemănătoare. Pentru acest scop se poate folosi similaritatea cosinus dintre A1 - A2 şi AB1 - AB2.

### 3.4.6 TraVeLGAN (Transformation Vector Learning GAN)

Reţeaua TraVeLGAN (Figura 3.3) este asemănătoare cu SDGAN, diferenţa constând în implementarea separată a discriminantul de reţeaua siameză. Cele trei componente vor funcţiona astfel:

* Generatorul primeşte ca input obiecte atât dintr-un domeniu sursă, cât şi dintr-un domeniu ţintă, şi generează transformările acestora în funcţie de parametrii săi la momentul respectiv. Rolul său este acelaşi, anume de a se îmbunătăţi pentru a produce obiecte cât mai realiste din domeniul ţintă. În plus, trebuie să conserve conţinutul acestora;
* Discriminantul primeşte ca input atât obiecte originale, cât şi obiecte produse de generator şi prezice probabilităţile ca acestea să fie reale sau false;
* Reţeaua siameză primeşte aceleaşi intrări ca discriminantul, şi produce pentru fiecare obiect un vector latent corespunzător. Rolul său este de a îi spune generatorului cum să conserve conţinutul obiectelor.



*Figura 3.3 Arhitectura TraVeLGAN [20]*

În arhitectura clasică a unui discriminant ce foloseşte straturi convoluţionale, acestea sunt urmate de straturi de normalizare. Acest lucru se aplică şi pentru reţelele GAN siameze. Aşadar, în timpul transformării obiectelor în vectori latenţi, acestea vor fi normalizate cel puţin o dată, restrângând spaţiul de reprezentare.

În cazul TraVeLGAN, pentru că reţeaua siameză primeşte direct inputuri originale sau prelucrate de generator, acestea nu vor mai fi normalizate. Astfel, avantajul este că pentru două obiecte A1 şi A2 din batch-uri diferite, generatorul va putea învăţa, şi încă destul de rapid, să reflecte distanţa dintre acestea în transformările lor AB1 şi AB2.

În lucrarea ‘TraVeLGAN: Image-to-image Translation by Transformation Vector Learning’ [20], sunt descrise mai multe funcţii obiectiv, ce este recomandat a fi utilizate împreună. Fiecare dintre acestea are un rol, plecând de la scopurile generatorului, discriminantului şi reţelei siameze. Deoarece arhitectura celor trei componente poate varia, iar scopul în care sunt aplicate poate fi diferit de la problemă la problemă, nu există o combinaţie precisă a funcţiilor care să producă rezultate bune în orice situaţie. Patru dintre acestea, descrise în Capitolul 2, subcapitolul *2.5 Funcţii obiectiv,* sunt funcţia obiectiv de identitate, funcţia obiectiv de conţinut, funcţia obiectiv adversarială şi funcţia obiectiv de consistenţă a ciclului, folosită pentru structurile de tip CycleGAN.

## 3.3 Algoritmul de recunoaştere muzicală

### 3.3.1 Introducere

Intuitiv, pentru a potrivi o melodie nou înregistrată cu o melodie dintr-o bază de date, ar trebui ca în baza de date să nu existe suprapuneri între intrările sale. Aşadar, pentru fiecare obiect adăugat ar trebui creat un cod hash unic.

Mai mult, aplicaţia va înregistra doar câteva secunde dintr-o melodie, pe care va încerca să le potrivească în diferite intervale de timp din diferite melodii, deci acurateţea algoritmului va depinde de secvenţa înregistrată din melodie, astfel că o porţiune mai “variată” din melodie s-ar putea să producă potriviri mai bune.

Aşadar, în baza de date trebuie evitate pe cât posibil şi suprapuneri de segmente din melodii şi trebuie acordată o importanţă egală tuturor momentelor unei melodii.

### 3.3.2 „Constelaţia”

Pentru că am ales să lucrez cu spectrograme care pun în evidenţă frecvenţele melodiilor, mă voi folosi de particularităţile lor pentru a crea valorile hash. Am preluat ideea originală a aplicaţiei Shazam, descrisă in ‘How Shazam Works - An explanation in Python’ [22], de a crea o „constelaţie” a fiecărei melodii, formată din cele mai proeminente vârfuri de frecvenţă, adică frecvenţele care ies în evidenţă prin amplitudinea lor, relativ la frecvenţele apropiate. Am realizat acest lucru prin parcurgerea „bin”-urilor de timp ale fiecărei spectrograme, din care am preluat amplitudinile aferente.

De asemenea, în urma acestei parcurgeri am creat un vector ce conţine proeminenţele vârfurilor selectate. Numărul de vârfuri ce îndeplinesc condiţiile cerute poate fi mai mare sau mai mic decât am vrea, aşa că am folosit o argpartitiepentru a selecta setul de varfuri final, de o dimensiune dată. Am preluat frecvenţele corespunzătoare acestor vârfuri şi le-am adăugat în lista *constellation* împreună cu „bin”-ul de timp corespunzător lor.

### 3.3.3 Dicţionarul de hash-uri

Totuşi, pentru că nu sunt reţinute prea multe informaţii despre domeniul timpului, va fi greu ca aplicaţia să potrivească perfect frecvenţele preluate din cele 10 secunde înregistrate cu frecvenţele din constelaţie. Multe cântece, în special din acelaşi gen muzical, folosesc aceleaşii frecvenţe sau combinaţii de frecvenţe pentru părţi specifice ale melodiei (intro, refren etc.), aşa că doar cu o secvenţă de 10 secunde va fi greu de creat o potrivire exactă, pentru că melodii diferite pot conţine un subset asemănător de frecvenţe pe o anumită secvenţă.

Soluţia propusă pentru această problemă este crearea mai multor hash-uri pentru aceeaşi melodie, combinând frecvenţele din constelaţie aflate la anumite distanţe una de alta. Aşadar, vor fi căutate combinaţii de frecvenţe în melodii, nu doar frecvenţe unitare. Pentru că spectrograma este creată prin aplicarea Transformatei Fourier pe ferestre de timp din melodie, se ştie din ce „bin” de timp face parte fiecare frecvenţă.

Având două frecvenţe şi distanţa dintre ele măsurată în ferestre de timp, hash-ul final va fi un număr pe 32 de biţi, primi 12 biţi reprezentând diferenţa de timp, următorii 10 prima frecvenţă, iar ultimii 10 cea de-a doua frecvenţă.

La crearea listei *constellation,* elementele sunt inserate astfel încât să fie sortate crescător în funcţie de timp*.* Aşadar, dacă alegem o pereche (timp, frecvenţă) din *constellation*, putem foarte uşor să accesăm următoarele n puncte în timp, pentru a crea hash-ul combinat. Pentru scopul actual, am ales n = 100.

Lista constelaţiei este parcursă prin două bucle imbricate în care sunt preluate două perechi asemănătoare celei de mai sus cu distanţa în „bin”-uri dintre ele mai mică sau egală ca n.

Pentru a reprezenta cele doua frecvenţe pe câte 10 biţi, acestea sunt împărţite la o frecvenţă mai mare decât cele din melodiile din setul de date, mai exact 22050, obţinând valori subunitare ce sunt înmulţite cu . Astfel, este sigur că valorile vor fi în intervalul . Mai departe, pentru reprezentarea pe biţi este calculată partea întreagă din acestea.

La acest pas au fost deja create toate valorile ce compun hash-ul final, aşa că sunt concatenate prin operaţia pe biţi „sau”, pentru a le potrivi în „bin”-urile de biţi corespunzătoare.

Ultimul pas este adăugarea cheii *hash* în dicţionarul *hashes* cu valoarea fiind perechea corespunzătoare formată din timpul preluat de la prima frecvenţă şi id-ul melodiei pentru care au fost realizate aceste operaţii.

### 3.3.4 Baza de date finală

Procedeul descris este aplicat pentru toate melodiile din baza de date, obţinându-se mai multe hash-uri pentru fiecare, iar baza de date finală este constituită dintr-un dicţionar în care cheia reprezintă hash-ul, iar valoarea conţine un tuplu de forma (fereastră de timp, cântec), în care prima valoarea este indicele ferestrei unde se găseşte prima frecvenţă din hash-ul respectiv, iar a doua valoare este un id asociat melodiei. Deoarece s-ar putea crea acelaşi hash pentru două melodii diferite, este reţinută ca valoare o listă de tupluri.

### 3.3.5 Procesul de recunoaştere

După înregistrarea a 10 secunde dintr-o melodie oarecare, pentru a o potrivi cu o intrare din baza de date se aplică acelaşi algoritm, adică i se creaza o „constelaţie”, apoi un dicţionar de hash-uri. Este parcurs dicţionarul şi sunt căutate potriviri în baza de date. Pentru că pot rezulta mai multe potrivi cu aceeaşi melodie, este reţinut un dicţionar în care se numără de câte ori hash-ul mostrei date este identic cu hash-ul unei melodii x din baza de date.

În final, melodia care are valoarea cea mai mare în dicţionar este presupusa melodie din care face parte mostra de 10 secunde. Se afişează un top al potrivirilor în ordine descrescătoare a valorii din dicţionar.

# 

# Capitolul 4

# Tehnologii utilizate în implementarea algoritmilor

## 4.1 Python

Conform documentaţiei originale [40], „Python este un limbaj de programare high-level, interpretat, orientat pe obiecte, cu semantici dinamice”. Este un limbaj atractiv datorită uşurinţei de a scrie cod, a structurilor integrate de date şi a multitudinii de librării pe care le suportă.

Toate implementările descrise în această lucrare au fost realizate în limbajul python, versiunea 3.8.10.

## 4.2 PyTorch

Creat în 2017, PyTorch este un framework optimizat utilizat pentru procese de deep learning ce lucrează cu obiecte de tip tensor şi poate rula atat pe CPU cât şi pe GPU. API-ul din python este împărţit în mai multe categorii de funcţionalităţi precum torch.nn, torch.nn.functional, torch.Tensor, torch.fft.

Alte doua framework-uri populare folosite pentru deep learning sunt TensorFlow şi Keras.

### 4.2.1 PyTorch vs TensorFlow

Atât PyTorch cât şi TensorFlow oferă soluţii pentru construirea modelelor, dar diferă prin implementare şi funcţionalităţi. Cea de-a doua conţine o varietate mai mare de opţiuni, pe când PyTorch este specifică python şi este orientată pe obiecte.

Deşi TensorFlow este mai populară, cercetătorii aleg PyTorch cand vine vorba de „flexibilitate, capabilităţi de debugging sau durată scurtă de antrenare” [11]. Acelaşi articol citat descrie PyTorch ca fiind mai avantajos în cazul paralelismului datelor, „bazându-se pe suport nativ pentru execuţii asincrone in python”.

Orice algoritmi bazaţi pe PyTorch pot fi implementaţi şi în TensorFlow, diferenţa fiind setarea manuală a anumitor parametrii sau funcţionalităţi. Aşadar, PyTorch este mai uşor de utilizat şi conţine toate implementările necesare unui algoritm de deep learning.

### 4.2.2 PyTorch vs Keras

Conform articolului citat anterior [11], Keras este preferat de dezvoltatori datorită uşurinţei de a construi modele, pe când PyTorch este preferat de matematicieni şi cercetători. Alte avantaje ale celui din urmă sunt rapiditatea şi opţiunile pentru debugging, ce nu sunt la fel de performante în Keras. Totodată, Keras funcţionează mai bine pe seturi mici de date, lucru care ar constitui un dezavantaj în cazul lucrării prezentate.

Având avantajele prezentate mai sus, am ales să folosesc PyTorch, versiunea 2.0.0, în implementările modelelor de deep learning prezentate în lucrare, fiind un framework foarte bine documentat, rapid şi uşor de utilizat.

## 4.3 Librării Python

### Torch

Descris în documentaţia PyTorch [32], „pachetul torch conţine structuri de date pentru tensori multidimensionali şi operaţii matematice ce pot fi aplicate pe aceştia, precum şi utilităţi pentru serializare eficientă a tensorilor”.

Am folosit funcţii precum:

* *torch.transpose* şi *torch.squeeze*, pentru a trece de la formatul datelor folosit în modele la formatul în care acestea pot fi afişate grafic;
* clasele *Dataset* şi *Dataloader* din torch.utils.data, pentru a stoca setul de date şi a îl organiza pentru o parcurgere cât mai uşoară;
* clasa *Module* din torch.nn, pentru a construi noi clase, modele sau straturi pentru acestea;
* clasa *Sequential* din torch.nn, pentru a construi o structură a modelelor;
* *torch.nn.init.constant\_* şi *torch.nn.init\_normal\_* pentru iniţializarea ponderilor şi a bias-urilor;
* straturi ale reţelelor implementate, precum *Conv2d*, *Flatten*, *BatchNorm*, *ReLU* etc.;
* diferite funcţii din torch.nn.functional precum *l1\_norm*, *normalzie*, *square*;
* funcţiile obiectiv *BCELoss* şi *CosineEmbeddingLoss*;
* implementarea optimizatorului *Adam*.

### Torchvision

Torchvision este librăria din PyTorch folosită în special pentru computer vision, ce conţine operaţii de creare şi transformare a seturilor de date formate din imagini şi oferă soluţii de construire a arhitecturilor modelelor ce le prelucrează.

Din modulul torchvision am folosit:

* clasele *Compose*, *ToTensor*, *Normalize* din submodulul transforms, pentru a preprocesa setul de date;
* *make\_grid* din submodulul utils, pentru a afişa spectrogramele într-un anumit format.

### Torch.optim

Torch.optim este pachetul din PyTorch ce conţine implementări ale diverşi optimizatori, precum optimizatorul Adam.

Torch.optim.lr\_scheduler oferă diferite metode de reducere dinamică a ratei de învăţare pe parcursul antrenării, luând în considerare diferite criterii. Clasa *ReduceLROnPlateau* reduce rata de învăţare când o metrică dată a încetat să se îmbunătăţească, după un anumit număr de epoci. În implementare clasa a fost folosită împreuna cu optimizatorii Adam atât pentru generator, cât şi pentru discriminant.

### Librosa

„Librosa este un pachet python pentru analiză muzicală şi audio”, conform documentaţiei sale [16]. Funcţionalităţile sale includ prelucrarea elementelor audio şi extragerea a diverse caracteristici ale acestora, spre exemplu prin aplicarea Transformatei Fourier sau a derivatelor acesteia.

Am utilizat versiunea 0.10.0 a modulului. Am folosit funcţia *load* pentru a încărca melodii din memorie şi funcţia *melspectrogram* din submodulul feature pentru a crea mel spectrogramele lor.

### NumPy

După cum aflăm în documentaţia sa [25], NumPy este pachetul de bază din python folosit pentru implementari ştiinţifice, ce realizează transformarea datelor într-un obiect de tip tablou pe care pot fi aplicate diverse prelucrări precum operaţii aritmetice, sortări, extragere de statistici, transformări în diverse structuri etc.

Am folosit versiunea 1.24.3 a modulului numpy şi funcţii precum:

* *array*, pentru transformarea unui obiect într-un tablou tip numpy;
* *save* şi *load*, pentru a salva şi încărca obiecte în şi din memorie;
* *fft.fftfreq*, pentru crearea unui tablou cu valorile „bin”-urilor de frecvenţă rezultate în urma aplicării Transformatei Fourier;
* *argpartition*, pentru crearea unei partiţii a unui tablou fiind dat un parametru k, astfel încât pentru al k-lea element fixat, toate elementele dinaintea lui vor fi mai mici ca el, iar toate elementele situate după vor avea valori mai mari.

### Matplotlib

Matplotlib este un pachet din python utilizat în realizarea de grafice pe baza datelor, vizualizarea imaginilor sau crearea de diverse animaţii ştiinţifice. În anumiţi algoritmi, am folosit interfaţa sa Pyplot, specializată în plotarea graficelor, din versiunea 3.7.1.

### IPython.display

„Display este un API public pentru unelte de afişare din IPython” [23]. Clasa *Audio* se ocupă de crearea unui obiect de tip audio şi afişarea sa în notebook-ul de unde este apelată. Versiunea folosită în implementări este 8.12.2.

### Scipy.signal

Pachetul scipy.signal este creat în scopul prelucrării semnalelor, cuprinzând diverse funcţionalităţi pentru filtrări, funcţii de tip fereastră, forme de unde, analiză spectrală.

Am folosit funcţia *find\_peaks,* din versiunea 1.10.1, pentru găsirea celor mai proeminente vârfuri de frecvenţe din spectrogramele date.

### Tqdm

Tqdm este o librărie din python utilizata pentru a urmări progresul a diverse procese precum parcurgerea unei instrucţiuni de tip buclă, încărcarea unei pagini, descărcarea unui fişier.

### Python Imaging Library

„Python Imaging Library adaugă capacităţi de prelucrare a imaginilor interpretorului de python” [30]. Clasa sa *Image* este folosită pentru a reprezenta datele de tip PIL sub formă de imagini. Din această clasă am folosit funcţiile *open* - pentru încărcarea imaginilor în program, *new* - pentru formarea unui nou obiect de tip imagine, şi *fromarray* - pentru transformarea unui tablou într-o imagine. Versiunea folosită a modulului este 9.5.0.

### Pathlib

Pathlib este un sistem de reprezentare a fişierelor orientat pe obiecte adaptat pentru diverse sisteme de operare. Clasa sa principala *Path,* din versiunea 1.0.1, a fost folosită pentru a lista sau găsi diverse fişiere din calculator.

### Joblib

Libraria joblib este utilă în uşurarea procesului de pipelining în python. Ea prezintă două avantaje majore, menţionate în articolul ‘Using joblib to speed up your Python pipelines’ [31]:

* capacitatea de a folosi memoria cache pentru a evita recalcularea unor paşi;
* execută paralelizare pentru a folosi în întregime resursele CPU/GPU.

În implementare am folosit funcţionalitatea sa de a salva şi încărca parametrii modelelor folosite, din versiunea 1.2.0.

### Re

Utilitatea librăriei re este de a permite folosirea expresiilor regulate în diferite contexte. În implementare, am folosit expresii regulate pentru sortarea unor path-uri în ordinea crescătoare a numerelor pe care le conţineau în nume.

### Pickle

„Modulul pickle implementează protocoale binare pentru serializarea şi deserializarea structurii obiectelor din python”, conform documentaţiei, versiunea 4.0 [29]. „‘Pickling’ este procesul prin care ierarhia unui obiect din python este convertită într-un stream de bytes, iar ‘unpickling’ este operaţia inversă”. Aceste operaţii sunt utile în încărcarea şi salvarea datelor, producând structuri ce ocupă mai puţină memorie decât în mod normal.

### Os

Librăria os din python are scopul de a permite interacţiunea cu sistemul de operare. Funcţia sa *scandir* este folosită pentru scanarea obiectelor prezente într-un folder dat ca input şi crearea unui iterator pentru a le putea parcurge şi a realiza diverse operaţii asupra lor.

Utilitatea os în lucrarea prezentă este de a scana un director pentru a forma o listă cu numele subdirectoarelor acestuia.

### Glob

Definiţia modelului glob afirmă că acesta „găseşte toate căile corespunzătoare unui model specificat conform regulilor utilizate de shell-ul Unix, chiar dacă rezultatele sunt returnate într-o ordine arbitrară” [7]. Astfel, este utilă în selectarea doar a anumitor obiecte dintr-un director dat.

### Typing

Funcţionalitatea principală a modului typing descrisă în documentaţia sa este de a „oferi suport în timpul rulării pentru sugestia tipurilor de date” [39]. Conţine tipuri de date precum Dict, List, Tuple.

## 4.4 Particularităţi hardware

În vederea realizării acestei lucrări, componenta hardware disponibilă pentru rularea tuturor algoritmilor ce vor fi prezentaţi în continuare dispune de 16 GB de memorie RAM şi 8 unităţi centrale computaţionale (CPU). Astfel, modelele de machine learning ar putea să nu îşi fi atins performanţa maximă, iar aplicaţia de recunoaştere muzicală nu putut fi scalată, operând doar pe un set mic de date.

# Capitolul 5

# Rezultate obţinute şi comparaţii

## 5.1 Transferul de gen muzical

În vederea transferului de gen muzical am implementat câteva din metodele discutate mai sus împreună cu anumite îmbunătăţiri. Pentru că nu există o metodă exactă de folosire a funcţiilor obiectiv în contextul reţelelor de tip GAN, am experimentat mai multe combinaţii ale acestor funcţii. În scopul acestei lucrări, le-am folosit pentru a îmbunătăţi generatorul şi discriminantul.

În afara cazului în care o structură este menţionată explicit, componentele reţelelor au fost construite astfel:

#### Generatorul

Arhitectura generatorului este de tip UNET, codorul şi decodorul având însă doar câte două blocuri.

Fiecare dintre blocurile codorului conţine un strat de convoluţie cu nucleu de dimensiune 4x4, stride 2 şi padding 1, un strat BatchNorm şi funcţia de activare LeakyReLU.

Blocurile decodorului conţin câte un strat de convoluţie transpusă cu aceiaşi hiperparametrii menţionaţi anterior. Primul bloc este urmat de un strat BatchNorm şi funcţia de activare LeakyReLU, pe când cel de-al doilea bloc este succedat doar de funcţia Tanh.

Având această structură, ieşirile vor avea aceeaşi dimensiune ca intrările.

#### Discriminantul

Discriminantul are în componenţa sa 3 blocuri ce conţin câte un strat de convoluţie cu nucleu de dimensiune 4x4, stride 2, padding 1, urmate de BatchNorm şi LeakyReLU. Al patrulea bloc conţine o convoluţie cu acelaşi nucleu, dar stride 1 şi padding 0, şi funcţia de activare Sigmoid.

Având această structură, pentru o intrare de dimensiuni 1x32x32, ieşirea va avea forma 1x1x1.

#### Reţeaua siameză

Acolo unde este folosită, reţeaua siameză cuprinde 3 straturi de convoluţie cu parametrii:

1. Nucleu de 4x4, stride 2, padding 1;
2. Nucleu 3x3, stride 2x1, padding 1x1;
3. Nucleu 3x3, stride 1x2, padding 1x1.

Fiecare strat este urmat de BatchNorm şi LeakyReLU. În final, este folosit un strat Flatten şi un strat Linear cu dimensiunea output-ului 128.

Ponderile generatorului şi ale discriminantului pentru toate straturile de convoluţie şi straturile de normalizare au fost iniţializate cu valori aleatoare dintr-o distribuţie normală cu medie 0 şi deviaţie standard 0.02, recomandarea venind din lucrarea ‘Generative Adversarial Nets’ [9].

În toate implementările, rata de învăţare a generatorului a fost de 0.0002, iar a discriminantului de 0.000001. Ambele componente au folosit optimizatorul Adam şi o metodă de ajustare dinamică a ratei de învăţare, prin funcţia ReduceLROnPlateau din modulul torch.optim.

Toate tipurile de straturi menţionate, precum şi rata de învăţare şi optimizatorul au fost detaliate în Capitolul 2 Preliminarii.

Seturile de date notate în continuare A şi B vor conţine spectrograme de melodii pop, respectiv rock.

În urma fiecărei încercări am formulat câteva concluzii care m-au ajutat să îmbunătăţesc modelul în următoarele etape. În cele ce urmează voi prezenta încercările făcute.

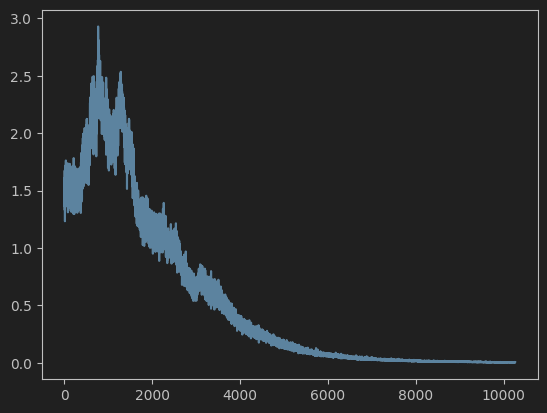
### 5.1.1 DCGAN

Iniţial, am implementat cel mai simplu algoritm prezentat pentru transferul de gen muzical, şi anume reţeaua generativă adversarială cu straturi convoluţionale. Am folosit funcţia obiectiv adversarială, folosită în paper-ul lui Goodfellow [9], implementând-o prin clasa BCELoss din modulul torch.nn. Astfel, valorile acesteia vor fi în intervalul [0, ∞), iar cele două componente trebuie să conveargă spre 0.

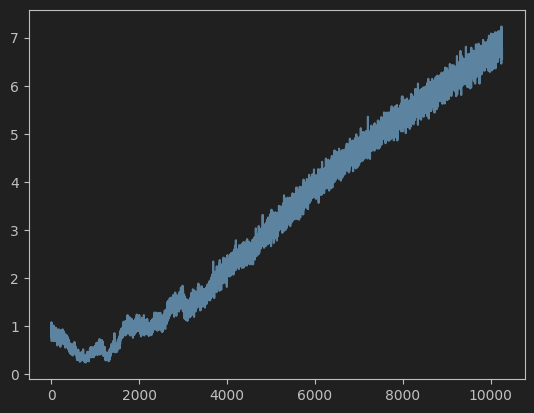
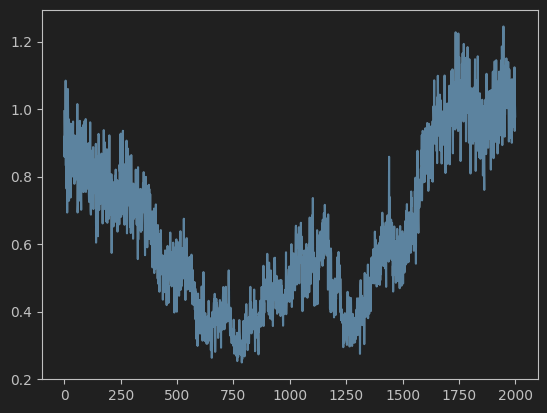
Pentru generator, am folosit structura descrisă anterior, însă cu 4 straturi convolutionale şi 4 straturi de convoluţie transpusă. Input-ul generatorului este constituit din „zgomot” format prin generare de numere aleatoare sub forma unui tablou de dimensiuni 1x32x32.

Input-ul discriminantului are forma 1x32x32 şi este constituit din exemple reale, din setul de date utilizat, cât şi din rezultate ale generatorului aplicat pe zgomot.

Modelul a fost antrenat 20 de epoci, pe 513 batch-uri de dimensiune 32. După prelucrarea unui batch, am reţinut valorile erorilor pentru generator şi discriminant.

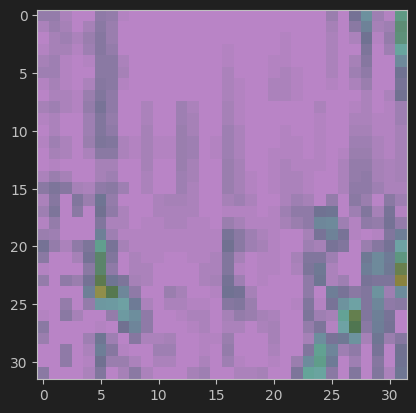
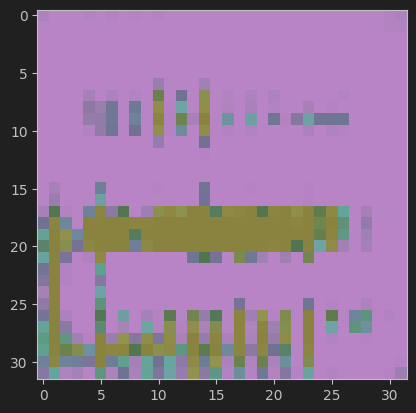


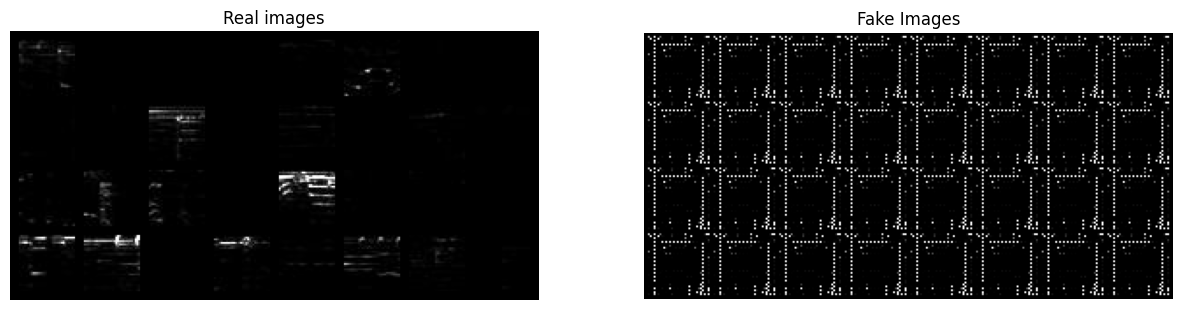
*Figura 5.1 Graficul funcţiei obiectiv a discriminantului*

*Figura 5.2 Graficul funcţiei obiectiv a generatorului*

Din Figura 5.1 se poate observa că discriminantul converge după aproximativ 7000 de iteraţii. Pe de altă parte, din Figura 5.2, observăm că eroarea generatorului se apropie de 0 după 750 de iteraţii, apoi după 1250 de iteraţii, adică aproximativ 2 epoci, dar nu converge niciodată.



*Figura 5.3 Exemple de spectrograme înainte şi după aplicarea transformării generatorului -*

*se observă pierderea conţinutului lor.*

Totuşi, am aplicat transformarea generatorului pe spectrograme din alt gen muzical şi se poate observa din Figura 5.3 că nu a fost conservat conţinutul. Mai mult, generatorul a prelucrat toate imaginile dintr-un batch în acelaşi mod.

DCGAN în aceasta formă este, deci, un model nesatisfăcător pentru transferul de gen muzical. Generatorul nu este suficient de puternic, iar conţinutul nu este conservat.

### 5.1.2 SDGAN

O a doua metodă implementată este GAN-ul cu discriminant ce se comportă ca o reţea siameză, output-ul său nemaifiind o probabilitate, ci un vector latent. Am preluat ideea din articolul ‘Style Transfer with GANs on HD Images’ [18], însă am păstrat doar structura de bază, modificând funcţiile obiectiv şi neadoptând alte schimbări menţionate.

Pentru că output-ul discriminantului trebuie să fie un vector, am modificat parametrii convoluţiilor astfel încât forma finală să fie 1x8x4, deci lungimea vectorului va fi 32.

Am lucrat cu două seturi de date, unul conţinând spectrograme dintr-un stil A, iar cel de-al doilea din stilul B, scopul fiind transformarea celor din A în genul B. Astfel, input-ul generatorului a fost de această dată constituit din exemple din setul A, cu forma 1x32x32.

Faţă de DCGAN, am adăugat modelului o funcţie contrastivă, implementată prin CosineEmbeddingLoss [2] (Ecuaţia 5.4) din modulul torch.nn:

*Ecuaţia 5.4 Funcţia CosineEmbeddingLoss din modulul torch.nn*

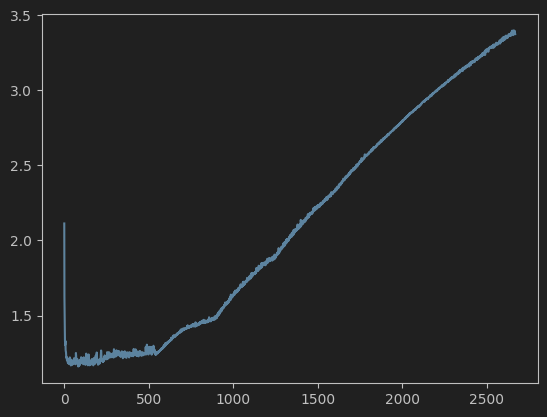
Funcţia urmăreşte acelaşi principiu ca şi funcţia obiectiv contrastivă, folosită în contextul reţelelor siameze. Pentru y = 1 în cazul imaginilor din aceeaşi distribuţie şi y=-1 altfel, şi o margine de 0.2, pierderea contrastivă apropie valoarea sa de 0 în cazul predicţiilor bune, şi o măreşte altfel.

În plus, pentru că este folosit cosinusul în măsurarea distanţelor, funcţia este folosită pentru a măsura similaritatea dintre cele două intrări. Într-un caz ideal, dacă cei doi tensori provin din aceeaşi distribuţie, valoarea cosinusului se va apropia de 1. Altfel, vor tinde să fie ortogonali, deci cosinusul va fi apropiat de 0.

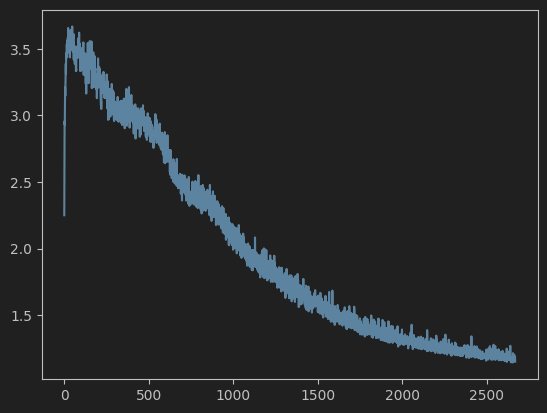
Am experimentat implementarea generatorului atât cu structura DCGAN-ului - cu 8 straturi convoluţionale, cât şi cu doar 4 straturi convoluţionale. Deoarece diferenţele nu au fost observabile, voi prezenta în continuare rezultate ale implementărilor de generatoare cu 4 straturi convoluţionale.

#### Arhitectura clasică

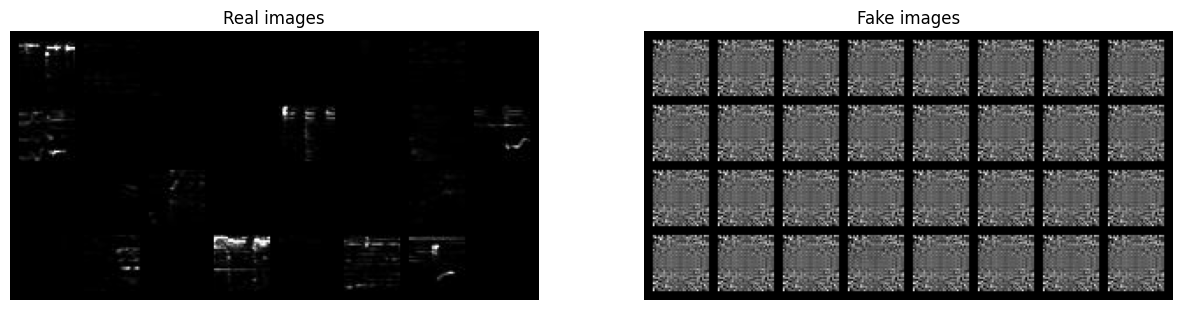
Cu structura descrisă în paragraful anterior, generatorul având 4 straturi convoluţionale în total, din Figura 5.6 se observă că discriminantul converge eventual, după 2500 de iteraţii. Totuşi, ca în cazul DCGAN, Figura 5.5 arată că generatorul nu converge. Conţinutul nu se conservă iar spectrogramele produse pot fi considerate „zgomot” (Figura 5.7).



*Figura 5.5 Graficul funcţiei obiectiv a generatorului*



*Figura 5.6 Graficul funcţiei obiectiv a discriminantului*

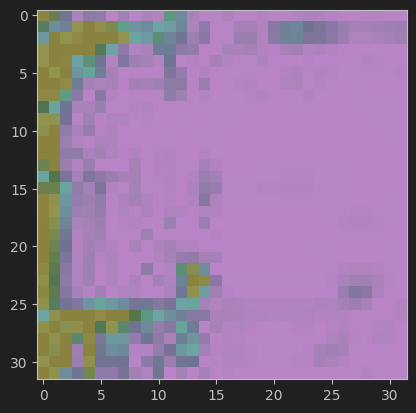
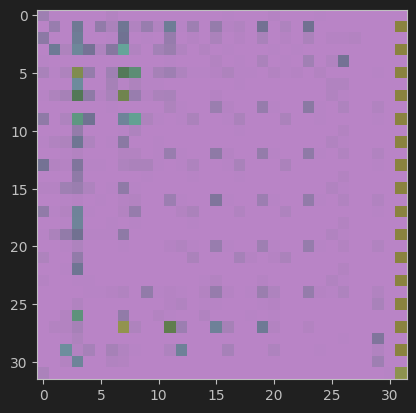


*Figura 5.7 Exemple de spectrograme înainte şi după aplicarea transformării generatorului -*

*se observă pierderea conţinutului lor*

#### Adăugarea stratului GaussianNoise

O altă încercare de îmbunătăţire a acestei reţele a fost adăugarea stratului GaussianNoise, descris în Capitolul 2, subcapitolul *2.4 Alte tipuri de straturi,* înainte de fiecare strat convoluţional al discriminantului pentru a face diferenţierea mai dificilă între imagini. Graficele funcţiilor obiectiv ale celor două componente au fost asemănătoare cu cele din urma încercării anterioare, discriminantul convergând, iar generatorul nu. Totuşi, am observat o oarecare conservare a conţinutului spectrogramelor (Figura 5.8).

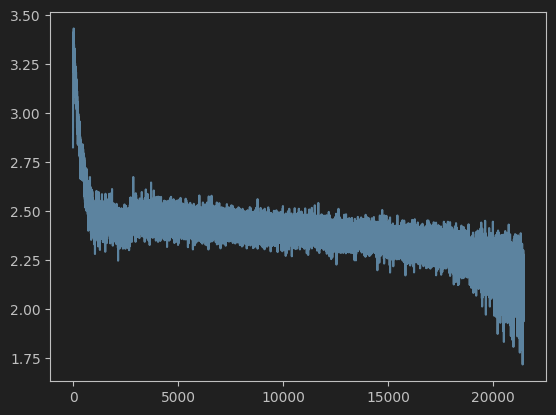
*Figura 5.8 O spectrogramă din setul A înainte şi după aplicarea transformării generatorului.*

#### Difuzia etichetelor

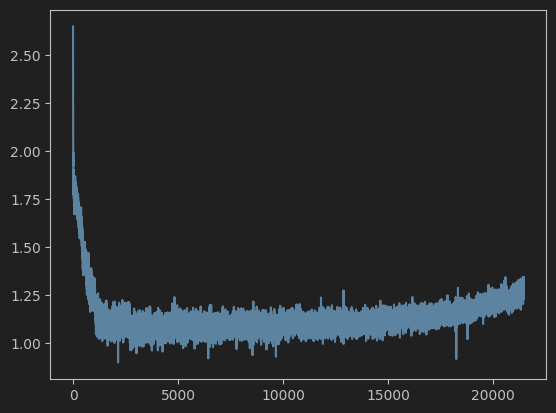
Pentru că nu am obţinut rezultate satisfăcătoare nici cu cea de-a doua încercare, am modificat valorile etichetelor faţă de care este calculată funcţia obiectiv adversarială, şi anume:

* Pentru fiecare etichetă „reală”, am ales o valoare aleatoare din intervalul 0.7 - 1.2;
* Pentru fiecare etichetă „falsă” am ales o valoare aleatoare din intervalul 0.0 - 0.3.

Tehnica este inspirată din ‘Towards Principled Methods for Training Generative Adversarial Networks’ [19]şi are la bază acelaşi principiu ca şi tehnica de adăugare a „zgomotului”, mai precis introducerea „confuziei” în structura discriminantului, pentru a îi încetini procesul de învăţare.



*Figura 5.9 Graficul funcţiei obiectiv a discriminantului*



*Figura 5.10 Graficul funcţiei obiectiv a generatorului*

Modelul a fost antrenat 40 de epoci şi funcţia de difuzie a etichetelor a fost o îmbunătăţire clară faţă de modelele anterioare. Deşi generatorul nu converge (Figura 5.10), valorile funcţiei sale obiectiv sunt mai apropiate de 0. Într-adevăr, spre finalul procesului acestea cresc, generatorul are tendinţă de divergenţă, iar discriminantul se îndreaptă spre convergenţă (Figura 5.9). Rezultatele sunt încă nesatisfăcătoare.

### 5.1.3 TraVeLGAN

În urma rezultatelor nesatisfăcătoare produse de variantele antrenate ale SDGAN, am decis implementarea reţelei siameze separat de discriminant, ceea ce a rezultat într-o arhitectură de tip Transformation Learning Vector GAN. Structurile celor trei componente sunt cele descrise la începutul subcapitolului *5.1 Transferul de gen muzical*.

#### Arhitectura de baza

Faţă de modelele anterioare, au fost modificate funcţiile obiectiv folosite. Am păstrat funcţia adversarială, fiind de bază în structurile bazate pe reţele generative adversariale. Pe lângă aceasta, am adăugat funcţia de identitate - specifică TraVeLGAN, şi o combinaţie liniară a funcţiei cosinus cu funcţia siameză - pentru a conserva conţinutul spectrogramelor. Această combinaţie necesită însă câte două obiecte din aceeaşi distribuţie pentru a fi calculate. Pentru a rezolva această problemă, am segmentat fiecare spectrogramă în două părţi, de dimensiune 32x16, pe care am aplicat transformările generatorului şi reţelei siameze. Apoi, pentru a le da ca input discriminantului, le-am asamblat în poziţia lor iniţială.

Pentru ca funcţiile obiectiv folosite să aibă un impact mai mare şi mai rapid asupra reţelei, am creat pentru generator o combinaţie liniară a acestora dată de Ecuaţia 5.11.

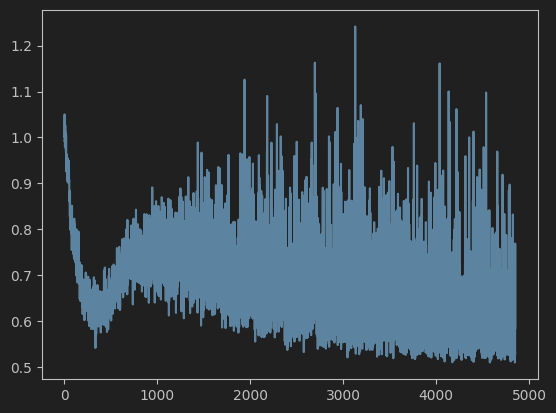
.

*Ecuaţia 5.11 Funcţia obiectiv a generatorului din implementarea TraVeLGAN*

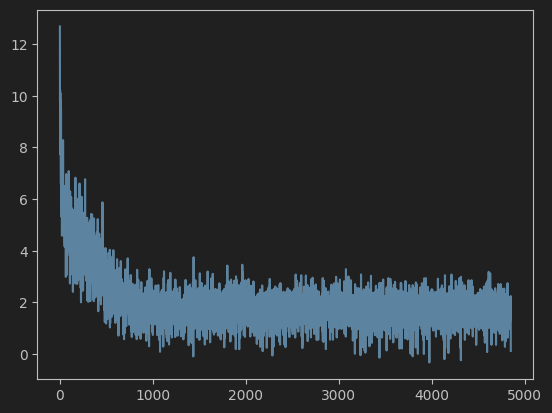
Pentru discriminant am păstrat doar valoarea funcţiei adversariale, atât pe input-uri originale, cât şi pe input-uri transformate.

Modelul a fost antrenat timp de 9 epoci, pe 538 batch-uri de dimensiune 32 din stilurile A şi B, acesta fiind maximul de iteraţii atins de componenta hardware.

Rezultatele obţinute au fost mai satisfăcătoare faţă de modelele încercate anterior, funcţia obiectiv a generatorului stabilizându-se sub un anumit prag, cum se poate observa în Figura 5.13. Totodată, valorile funcţiei discriminantului, prezentate în Figura 5.12, oscilează foarte mult după un anumit număr de epoci, ceea ce înseamnă că acesta a fost „păcălit”. După un număr mai mare de iteraţii, generatorul ar fi putut converge.



*Figura 5.12 Graficul funcţiei obiectiv a discriminantului*



*Figura 5.13 Graficul funcţiei obiectiv a generatorului*

#### 

*Figura 5.14 O spectrograma din setul A înainte şi după aplicarea transformării generatorului.*

Spre deosebire de modelele anterioare, îmbunătăţirea se poate observa şi vizual. Se observă din Figura 5.14 că de această dată generatorul a conservat conţinutul spectrogramelor, adăugând şi propria modificare.

#### Adăugarea feature matching

Am antrenat modelul descris anterior împreună cu o nouă funcţie obiectiv pentru generator şi anume cea de potrivire a caracteristicilor, descrisă în Capitolul 2, subcapitolul *2.5 Funcţii obiectiv*.

Cu ajutorul acestei funcţii sunt reţinute trăsături extrase de discriminant într-un moment intermediar formării predicţiei atât pentru spectrogramele originale cât şi pentru cele transformate. Momentul este ales astfel încât să fie capturate pattern-uri cât mai reprezentative. De asemenea, la extragerea acestor trăsături nu este adăugat stratul de zgomot.

Intrucât scopul generatorului este de a „păcăli” discriminantul, diferenţa dintre trăsăturile extrase în cele două cazuri ar trebui să fie cât mai mică.

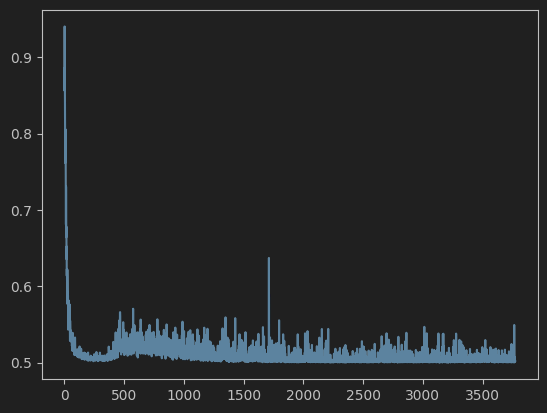
Astfel, am adăugat discriminantului o variantă a sa fără straturile GaussianNoise, iar hărţile de trăsături preluate din aceasta au fost cele din urma straturilor convoluţionale 2 şi 3. De asemenea, am mărit rata sa de învăţare la 0,00001.

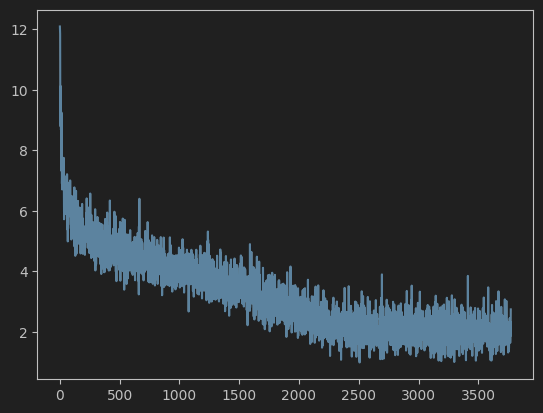
Funcţia obiectiv finală a generatorului, adăugând ponderi componentelor sale este dată de Ecuaţia 5.15.

*Ecuaţia 5.15 Funcţia obiectiv a generatorului din implementarea TraVeLGAN cu feature matching*

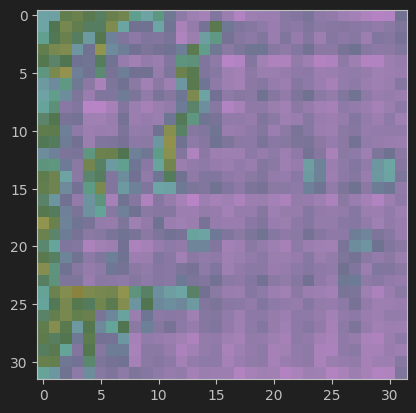
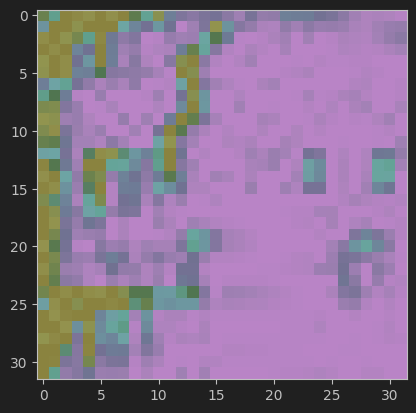
Modelul a fost antrenat timp de 7 epoci, pe 538 batch-uri de dimensiune 32 din stilurile A şi B, acesta fiind maximul de iteraţii atins de componenta hardware.

Cum se observă din Figura 5.17, valorile funcţiei generatorului sunt din ce în ce mai mici, apropiindu-se de convergenţă, deci modelul este o rezolvare promiţătoare a transferului de gen muzical. De asemenea, se observă din Figura 5.18 conservarea conţinutului spectrogramelor. Se poate observa că discriminantul converge (Figura 5.16).

*Figura 5.16 Graficul funcţiei obiectiv a discriminantului*

**

*Figura 5.17 Graficul funcţiei obiectiv a generatorului*



*Figura 5.18 O spectrograma din setul A înainte şi după aplicarea transformării generatorului.*

## 5.2 Recunoaşterea muzicală

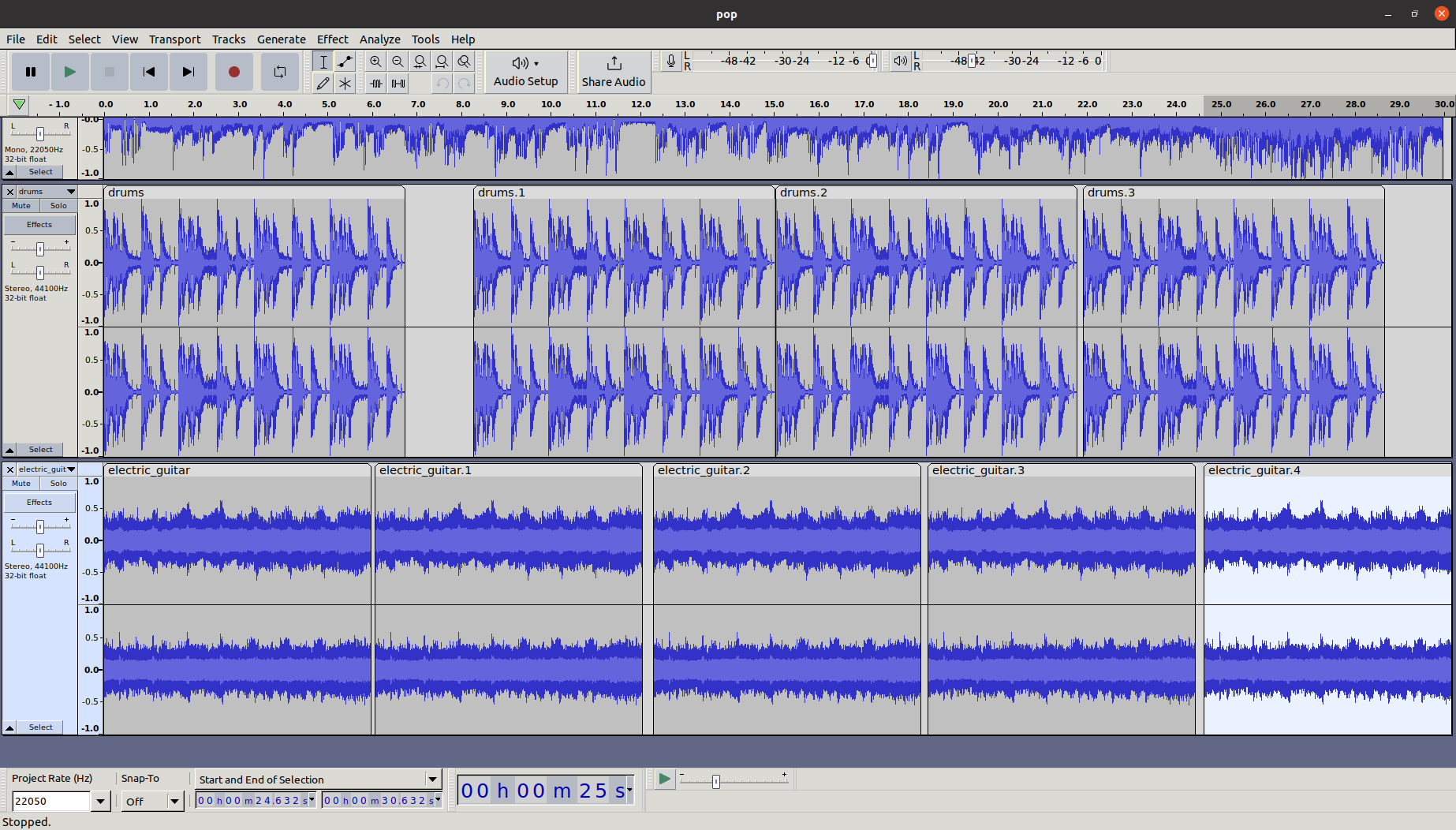
Am implementat algoritmul de recunoaştere muzicală aşa cum este descris în Capitolul 3, subcapitolul *3.3 Algoritmul de schimbare de gen muzical*.

În urma rezultatelor obţinute pentru problema de transfer de gen muzical, am salvat ambele generatoare ale modelelor TraVeLGAN pentru a le testa în algoritmul de recunoaştere muzicală. În continuare, mă voi referi la cel al cărei funcţie obiectiv cuprinde şi potrivirea trăsăturilor drept al doilea model, iar la celălalt drept primul model.

Pentru că generatoarele pot să transforme melodii în genul muzical rock, şi pentru că genul pop sună în general diferit de acesta, am ales să lucrez cu acestea două drept genuri de bază. Din fiecare am avut la dispoziţie câte 100 de melodii de câte 30 de secunde.

Am ales o melodie pop la întâmplare pe care o voi numi melodia 0 din genul pop. Pentru a testa cât mai realist aplicaţia, am căutat metode de a o face să sune cât mai rock, fără a o prelucra în spectrograme. Astfel, am preluat două snippet-uri de câte 7 secunde, unul cu tobe1, iar celălalt cu chitară electrică2 pentru a le adăuga melodiei.

Am folosit aplicaţia Audacity3 pentru a suprapune cele trei audio-uri şi pentru a multiplica segmentele cu instrumente muzicale. Totuşi, am ales poziţionarea lor într-o ordine aleatoare, astfel încât să se regăsească cât mai mult în melodie. Organizarea finală poate fi observată în Figura 5.19.



*Figura 5.19 Suprapunerea melodiei 0 din genul pop cu snippet-urile de tobe şi chitară electrică*

Deoarece scopul aplicaţiei Find your music este să recunoască melodii după segmente de doar 10 secunde din acestea, am tăiat melodia rezultată în trei segmente aleatoare pe care le-am salvat drept *pop+drums+guitar\_cut0.1.wav, pop+drums+guitar\_cut0.2.wav* şi *pop+drums+guitar\_cut0.3.wav.* Acestea sunt audio-urile pe care am testat aplicaţia.

Pentru a acoperi cât mai multe cazuri posibile, am alcătuit trei seturi de date formate din:

* Setul 1 - spectrogramele celor 100 de melodii pop şi ale celor 100 de melodii rock, plus spectrograma melodiei 0 pop transformată atât cu modelul 1, cât şi cu modelul 2;
* Setul 2 - toate spectrogramele din setul 1 plus spectrogramele melodiilor pop prelucrate în melodii rock cu modelul 1;
* Setul 3 - toate spectrogramele din setul 2 plus spectrogramele celor 100 de melodii blues transformate în melodii rock cu modelul 2.

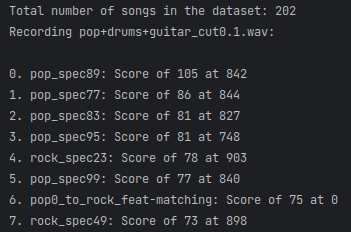
Am numit spectrograma melodiei 0 transformată cu modelul 1 *pop0\_to\_rock*, iar pe cea transformată cu modelul 2 *pop0\_to\_rock\_feat-matching.* Restul spectrogramelor respectă notaţia *[gen\_muzical]\_spec[numărul\_melodiei].*

În continuare voi prezenta rezultatele obţinute pe cele trei seturi de date.

### **pop+drums+guitar\_cut0.1**

#### Setul 1

Transformarea modelului 2 a ocupat locul 6 din 202 în topul potrivirilor (Figura 5.20), pe când cea a modelului 1 a ocupat locul 39, iar spectrograma originală locul 24.

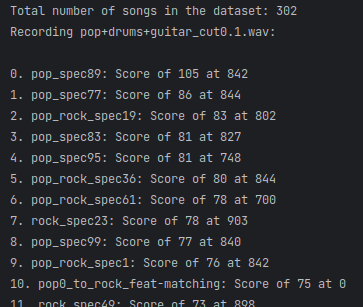


*Figura 5.20 Topul potrivirilor muzicale pe segmentul 1, setul 1*

#### 

#### Setul 2

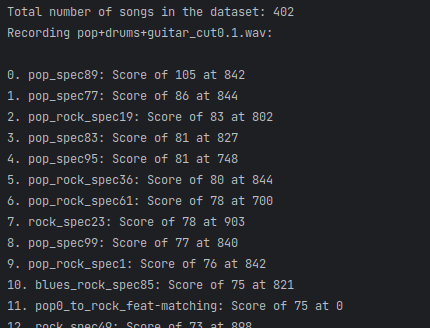
Pe al doilea set de date modelul 2 este clasat pe poziţia 10 (Figura 5.21), modelul 1 pe poziţia 66, iar spectrograma originală pe poziţia 40.



*Figura 5.21 Topul potrivirilor muzicale pe segmentul 1, setul 2*

#### Setul 3

Pe al treilea set de date modelul 2 este clasat pe poziţia 11 (Figura 5.22), modelul 1 pe poziţia 83, iar spectrograma originală pe poziţia 46.



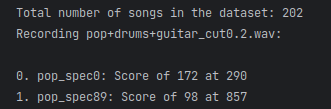
*Figura 5.22 Topul potrivirilor muzicale pe segmentul 1, setul 3*

În concluzie, rezultatele obţinute pe primul segment sunt foarte bune pentru modelul 2, relativ la faptul că spectrograma prelucrată este deasupra celei originale considerabil, iar pe setul de 402 melodii este pe locul 11.

### pop+drums+guitar\_cut0.2

#### Setul 1

De această dată, spectrograma originală a fost clasată prima, iar celelalte două pe locurile 31, respectiv 53 (Figura 5.23).

*Figura 5.23 Topul potrivirilor muzicale pe segmentul 2, setul 1*

#### Setul 2

Pe al doilea set de date spectrograma originală îşi menţine locul 1, pe când modelul 2 este clasat pe poziţia 46, modelul 1 pe poziţia 79.

#### Setul 3

Pe al treilea set de date spectrograma originală rămâne tot pe locul 1, pe când modelele 1 şi 2 au scăzut în clasament.

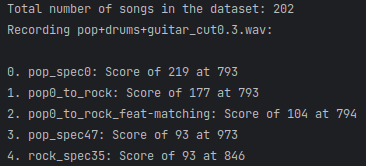
În concluzie, rezultatele obţinute pe al doilea segment nu sunt deloc satisfăcătoare pentru algoritmii de transfer muzical. Problema ar putea fi modul de poziţionare aleator al celor două snippet-uri, sau doar componentele melodiei pe acel segment.

Totuşi, rezultatele sunt un plus pentru algoritmul de recunoaştere muzicală, deoarece melodia originală s-a situat mereu pe primul loc, indiferent de dimensiunea setului de date.

### pop+drums+guitar\_cut0.3

#### Setul 1

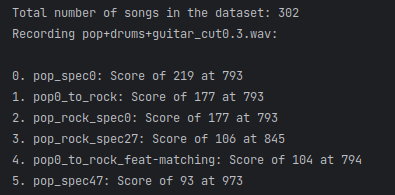
Cele trei spectrograme au ocupat primele locuri, însă cea originală este prima (Figura 5.24).



*Figura 5.24 Topul potrivirilor muzicale pe segmentul 3, setul 1*

#### Setul 2

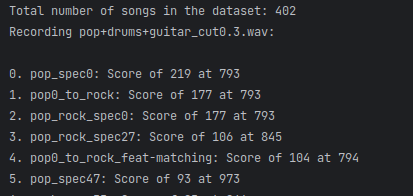
Clasamentul este acelaşi ca la setul 2 (Figura 5.25).



*Figura 5.25 Topul potrivirilor muzicale pe segmentul 3, setul 2*

#### Setul 3

Clasamentul este din nou acelaşi (Figura 5.26).



*Figura 5.26 Topul potrivirilor muzicale pe segmentul 3, setul 3*

În concluzie, spectrograma originală este clasată tot înaintea celorlalte două, ca în cazul segmentului 2, însă îmbunătăţirea semnificativă este faptul că cele trei ocupă mereu primele 3 poziţii, deci algoritmul de recunoaştere funcţionează ideal în acest caz.

Sumarizând rezultatele obţinute, este clar observabil că al doilea model TraVeLGAN a avut cele mai bune rezultate, chiar satisfăcătoare relativ la dimensiunea setului de date. Desigur, există loc şi metode de îmbunătăţire.

Algoritmul de recunoaştere în particular a avut rezultate bune, spectrogramele provenind din melodia 0 fiind mereu primele sau aproape de vârful clasamentului.

Prelucrarea melodiei 0 în genul rock a reprezentat de asemenea un factor important al acestor concluzii. Cu o prelucrare profesionistă, adecvată, cu sens, consider că rezultatele ar fi putut fi bune.

# Capitolul 6

# Concluzii

Schimbarea de gen muzical este o problemă dificilă şi prea puţin experimentată. Există multe metode teoretice, însă acestea ar putea funcţiona mai bine sau mai rău, în funcţie de datele alese, de genul ţintă sau alţi factori de care trebuie să ţină cont reţelele neuronale.

Recunoaşterea muzicală presupune un algoritm mai simplu, de la care se pot face modificări cu rezultat pozitiv imediat, depinzând însă de arhitectura hardware avută la dispoziţie.

#### Direcţii viitoare

Întrucât se poate observa că unele componente ale reţelelor neuronale nu ajung să conveargă, este evident că performanţa algoritmul de transfer muzical poate fi îmbunătăţită. S-ar putea realiza acest lucru prin creşterea setului de date, a complexităţii modelelor şi a timpului de antrenare. Întrucât acestea sunt aspecte care au legătură directă cu hardware-ul folosit, acest lucru nu a fost posibil în realizarea lucrării actuale.

Un pas important în dezvoltarea aplicaţiei ar fi testarea algoritmului de recunoaştere cu mult mai multe melodii şi cu multiple prelucrări, în mai multe genuri muzicale.

Pentru creşterea acurateţei de recunoaştere a cover-urilor sau a interpretărilor live, algoritmul de schimbare de gen muzical ar putea fi folosit pentru adăugarea de zgomot sau pentru schimbarea tonalităţilor, a timbrului solistului etc.

Algoritmul de recunoaştere muzicală poate fi scalat eventual pentru a opera pe o bază de date ce cuprinde toate melodiile existente, cu un spaţiu de stocare adecvat. De asemenea, aceasta ar trebui configurată încât adăugarea de noi melodii să fie uşor de realizat.

Pentru a putea fi comercializată, ar trebui adaugată aplicaţiei o interfaţă uşor de folosit, îmbinând componenta de back-end cu cea de front-end.

# 

# Bibliografie

[1] Alec Radford et al., „Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, în (2016), DOI: <https://doi.org/10.48550/arXiv.1511.06434>, URL: <https://arxiv.org/abs/1511.06434>, accesat: 01.06.2023.

[2] *CosineEmbeddingLoss - PyTorch 2.0 documentation*, URL: [https://pytorch.org/docs/stable](https://pytorch.org/docs/stable/generated/torch.nn.CosineEmbeddingLoss.html)

[/generated/torch.nn.CosineEmbeddingLoss.html](https://pytorch.org/docs/stable/generated/torch.nn.CosineEmbeddingLoss.html), accesat: 01.06.2023.

[3] Encyclopædia Britannica, *Sound | Properties, Types, & Facts | Britannica,* URL:<https://www.britannica.com/science/sound-physics>, accesat: 01.06.2023.

[4] GeeksforGeeks, *Generative Adversarial Network (GAN)*,URL: <https://www.geeksforgeeks.org/generative-adversarial-network-gan/>, accesat: 01.06.2023.

[5] GeeksforGeeks, *Intuition of Adam Optimizer*, URL: <https://www.geeksforgeeks.org/intuition-of-adam-optimizer/>, accesat: 01.06.2023.

[6] GeeksforGeeks, *ML | ADAM (Adaptive Moment Estimation) Optimization*, URL: <https://www.geeksforgeeks.org/adam-adaptive-moment-estimation-optimization-ml/>, accesat: 01.06.2023.

[7] *glob — Unix style pathname pattern expansion*, URL: <https://docs.python.org/3/library/glob.html>, accesat: 01.06.2023.

[8] Hemalatha Raghavan, „Activation Function: Sigmoid”, în (2023), URL: <https://cloud2data.com/activation-function-sigmoid/>, accesat: 01.06.2023.

[9] Ian J. Goodfellow et al., „Generative Adversarial Nets”, în (2014), DOI: <https://doi.org/10.48550/arXiv.1406.2661>, URL: <https://arxiv.org/abs/1406.2661>, accesat: 10.06.2023.

[10] Johann Huber, „Batch normalization in 3 levels of understanding”, în (2020), URL: <https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338#b93c>, accesat: 01.06.2023.

[11] John Terra, „Keras vs Tensorflow vs Pytorch: Key Differences Among Deep Learning”, în (2023), URL: <https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article>, accesat: 01.06.2023.

[12] Kaggle, *GTZAN Dataset - Music Genre Classification,* URL:<https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification>, accesat: 01.05.2023.

[13] Kuan Wei, „Understand Transposed Convolutions”, în (2020), URL: <https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-4f5d97b2967>, accesat: 01.06.2023.

[14] Leland Roberts, „Understanding the Mel Spectrogram”, în (2020), URL: <https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53>, accesat: 01.06.2023.

[15] Leon A. Gatys et al., „A Neural Algorithm of Artistic Style”, în (2015), DOI: <https://doi.org/10.48550/arXiv.1508.06576>, URL: <https://arxiv.org/abs/1508.06576>, accesat: 10.06.2023.

[16] *librosa,* DOI: 10.5281/zenodo.7746972, URL: <https://librosa.org/doc/latest/index.html>, accesat: 01.06.2023.

[17] Marco Pasini, „A New Way to look at GANs”, în (2019), URL: <https://towardsdatascience.com/a-new-way-to-look-at-gans-7c6b6e6e9737>, accesat: 01.06.2023.

[18] Marco Pasini, „Style Transfer with GANs on HD Images”, în (2019), URL: <https://towardsdatascience.com/style-transfer-with-gans-on-hd-images-88e8efcf3716>, accesat: 01.06.2023.

[19] Martin Arjovsky, Léon Bottou, „Towards Principled Methods for Training Generative Adversarial Networks”, în (2017), DOI: <https://doi.org/10.48550/arXiv.1701.04862> , URL: <https://arxiv.org/abs/1701.04862>, accesat: 10.06.2023.

[20] Matthew Amodio, Smita Krishnaswamy, „TraVeLGAN: Image-to-image Translation by Transformation Vector Learning”, în (2019), DOI: <https://doi.org/10.48550/arXiv.1902.09631>, URL: <https://arxiv.org/abs/1902.09631>, accesat: 10.06.2023.

[21] Mayank Mishra, „Convolutional Neural Networks, Explained”, în (2020), URL: <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>, accesat: 01.06.2023.

[22] Michael Strauss, „How Shazam Works - An explanation in Python”, în (2021), URL: <https://michaelstrauss.dev/shazam-in-python>, accesat: 15.05.2023.

[23] *Module: display - IPython 3.2.1 documentation*, URL: <https://ipython.org/ipython-doc/3/api/generated/IPython.display.html>, accesat: 01.06.2023.

[24] Nikhil Tomar, „What is UNET?”, în (2021), URL: <https://medium.com/analytics-vidhya/what-is-unet-157314c87634>, accesat: 01.06.2023.

[25] *NumPy documentation*, URL: <https://numpy.org/doc/stable/>, accesat: 01.06.2023.

[26] O’Reilly, *Siamese networks*, URL: <https://www.oreilly.com/library/view/advanced-deep-learning/9781789956177/d06332c0-0c39-413b-951e-51b95005eeda.xhtml>, accesat: 01.06.2023.

[27] Oxford Languages, *Neural network Definition and Meaning,* URL: <https://www.dictionary.com/browse/neural-network>, accesat: 01.06.2023.

[28] Papers With Code, *Tanh Activation,* URL: <https://paperswithcode.com/method/tanh-activation>, accesat: 01.06.2023.

[29] *pickle — Python object serialization*, URL: <https://docs.python.org/3/library/pickle.html>, accesat: 01.06.2023.

[30] *Pillow*, URL: <https://pypi.org/project/Pillow/>, accesat: 01.06.2023.

[31] Pratik Gandhi, „Using joblib to speed up your Python pipelines”, în (2020), URL: <https://towardsdatascience.com/using-joblib-to-speed-up-your-python-pipelines-dd97440c653d>, accesat: 01.06.2023.

[32] *PyTorch 2.0 documentation* - <https://pytorch.org/docs/stable/index.html>, accesat: 01.06.2023.

[33] ResearchGate, *Typical Generative Adversarial Networks (GAN) architecture*, URL: <https://www.researchgate.net/figure/Typical-Generative-Adversarial-Networks-GAN-architecture_fig2_349182009>, accesat: 01.06.2023.

[34] Rinu Gour, „Artificial Neural Network for Machine Learning– Structure & Layers”, în (2019), URL: <https://medium.com/javarevisited/artificial-neural-network-for-machine-learning-structure-layers-a031fcb279d7>, accesat: 10.06.2023.

[35] Sagar Sharma, „Activation Functions in Neural Networks”, în (2017), URL: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>, accesat: 01.06.2023.

[36] Sik-Ho Tsang, „Review — CycleGAN: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks (GAN)”, în (2021), URL: <https://sh-tsang.medium.com/review-cyclegan-unpaired-image-to-image-translation-using-cycle-consistent-adversarial-networks-1c2602805be2>, accesat: 01.06.2023.

[37] TensorSpace.js, *Flatten,* URL: <https://tensorspace.org/html/docs/layerFlatten.html>, accesat: 01.06.2023.

[38] Tim Salimans et al., „Improved Techniques for Training GANs”, în (2016), DOI:

<https://doi.org/10.48550/arXiv.1606.03498>, URL: <https://arxiv.org/abs/1606.03498>, accesat: 10.06.2023.

[39] *typing — Support for type hints*, URL: <https://docs.python.org/3/library/typing.html>, accesat: 01.06.2023.

[40] *What is Python? Executive Summary*, URL: <https://www.python.org/doc/essays/blurb/>, accesat: 01.06.2023.

[41] Wikipedia, *Mel scale,* URL: <https://en.wikipedia.org/wiki/Mel_scale>, accesat: 01.06.2023.

[42] Wikipedia, *Nyquist–Shannon sampling theorem*, URL: <https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem>, accesat: 01.06.2023.