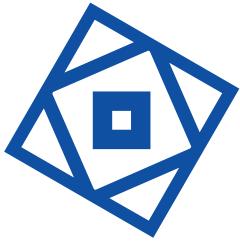


Universitatea  
Transilvania  
din Brașov  
FACULTATEA DE MATEMATICĂ  
ȘI INFORMATICĂ

# LUCRARE DE LICENȚĂ

**Absolvent:** Andrei-Daniel BOBEŞ  
**Coordonator:** Conf. dr. Lucian Mircea SASU  
**Co-supervizor:** Drd. Bogdan Adrian MUSAT

Brașov  
Iulie 2021



Universitatea  
Transilvania  
din Brașov  
FACULTATEA DE MATEMATICĂ  
ȘI INFORMATICĂ

## LUCRARE DE LICENȚĂ

Creșterea calității imaginilor prin  
super-rezoluție

**Absolvent:** Andrei-Daniel BOBEŞ

**Coordonator:** Conf. dr. Lucian Mircea SASU

**Co-supervizor:** Drd. Bogdan Adrian MUŞAT

Brașov  
Iulie 2021

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>4</b>
1.1	Scurtă prezentare a temei . . . . .	4
1.2	Motivația alegerii temei . . . . .	5
1.3	Contribuții personale . . . . .	5
1.4	Structura lucrării . . . . .	7
1.5	Contribuțiile colaboratorului și co-supervizorului asupra temei . .	7
<b>2</b>	<b>Descrierea subiectului</b>	<b>8</b>
2.1	Ce este super-rezoluția? . . . . .	8
2.2	Evoluția super-rezoluției folosind Deep Learning . . . . .	9
<b>3</b>	<b>Setul de date</b>	<b>14</b>
3.1	Obținerea setului de date . . . . .	14
3.2	Structura setului de date . . . . .	16
3.3	Prelucrarea setului de date . . . . .	16
3.3.1	Împărțirea setului de date . . . . .	17
<b>4</b>	<b>Brevier teoretic pentru rețelele neurale artificiale</b>	<b>19</b>
4.1	Introducere . . . . .	19
4.1.1	Inteligenta artificiala, Machine Learning și Deep Learning .	19
4.1.2	Tipuri de învățare . . . . .	21
4.1.3	Aplicabilitatea rețelelor neurale artificiale . . . . .	22
4.2	Concepțe de bază . . . . .	22
4.2.1	Neuronul artificial . . . . .	22
4.2.2	Structura unei rețele neurale artificiale . . . . .	23
4.2.3	Propagarea înainte . . . . .	24
4.2.4	Funcții de activare . . . . .	25
4.2.4.1	Funcția Sigmoid . . . . .	25
4.2.4.2	Funcția tangentă hiperbolică . . . . .	26
4.2.4.3	ReLU . . . . .	26
4.2.4.4	Funcția Leaky ReLU . . . . .	28

4.2.4.5	Funcția Softmax . . . . .	28
4.2.5	Funcția de cost . . . . .	29
4.2.6	Propagarea înapoi a erorii . . . . .	30
4.2.7	Strategii de optimizare . . . . .	31
4.2.7.1	Algoritmul de optimizare Adam . . . . .	31
4.2.8	Overfitting și Underfitting . . . . .	32
4.2.8.1	Overfitting . . . . .	33
4.2.8.2	Underfitting . . . . .	34
4.2.9	Regularizare . . . . .	34
4.2.9.1	Batch Normalization . . . . .	35
4.2.9.2	Weight Normalization . . . . .	36
4.2.9.3	Spectral Normalization . . . . .	37
4.2.10	Antrenarea . . . . .	37
4.2.10.1	Batch Size . . . . .	37
4.2.10.2	Numărul de epoci . . . . .	38
4.2.10.3	Rata de învățare . . . . .	38
4.2.11	Metrici de performanță . . . . .	38
4.2.11.1	Peak Signal-to-Noise Ratio . . . . .	39
4.2.11.2	Structural Similarity Index . . . . .	40
4.3	Rețele neurale artificiale de convoluție . . . . .	40
4.3.1	Strat convoluțional . . . . .	41
4.3.2	Strat de pooling . . . . .	42
4.4	Rețele reziduale recurente . . . . .	43
<b>5</b>	<b>Modele investigate</b>	<b>45</b>
5.1	Arhitecturi abordate . . . . .	46
5.1.1	Arhitectura SRGAN . . . . .	46
5.1.2	Arhitectura ESRGAN . . . . .	48
5.1.3	Arhitectura RDNSR . . . . .	49
5.2	Utilizarea unui model pre-antrenat pentru determinarea fluxului optic . . . . .	52
5.3	Utilizarea funcției de cost Ping-Pong . . . . .	53
5.4	Modificarea arhitecturii FRVSR . . . . .	55
5.4.1	Descrierea arhitecturii . . . . .	55
5.4.2	Modificări aduse structurii modelului . . . . .	57
5.5	Soluția finală . . . . .	60
<b>6</b>	<b>Tehnologii utilizate</b>	<b>62</b>
6.1	Limbajul de programare Python . . . . .	62
6.2	PyTorch . . . . .	63
6.2.1	TorchVision . . . . .	63

6.2.2	Module și clase abstracte . . . . .	63
6.3	Biblioteci . . . . .	64
6.3.1	NumPy . . . . .	64
6.3.2	Matplotlib . . . . .	64
6.3.3	OpenCV . . . . .	65
6.3.4	Flow-Vis . . . . .	65
6.3.5	youtube-dl și FFmpeg . . . . .	66
6.3.6	CUDA . . . . .	66
<b>7</b>	<b>Ghidul aplicației utilizator</b>	<b>67</b>
<b>8</b>	<b>Concluzii și direcții ulterioare de dezvoltare</b>	<b>71</b>
8.1	Experiența dobândită . . . . .	72
<b>Bibliografie</b>		<b>73</b>

# Capitolul 1

## Introducere

### 1.1 Scurtă prezentare a temei

Viziunea computațională<sup>1</sup> reprezintă un subiect actual și în plină expansiune, cu o diversitate amplă când vine vorba de problemele pe care încearcă să le rezolve. În ceea ce privește problemele de computer vision, acestea au o aparență simplă deoarece sunt rezolvate trivial de către un om, chiar și de un copil, însă rămân nerezolvate de calculator datorită limitării noastre în a înțelege viziunea biologică. Complexitatea percepției este dificil de pus sub o formă matematică.



Figura 1.1: Exemplu de obținere a unei imagini prin tehnici de super-rezoluție

Una dintre problemele clasice ale procesării de imagini este super-rezoluția. Problema adresată de această ramură este reconstruirea imaginilor cu o rezoluție ridicată din aceeași imagine la o rezoluție mică. Odată cu interesul către deep learning, în ultimii ani, super-rezoluția a primit o atenție semnificativă din partea comunității de cercetători.

<sup>1</sup>Din engleză: Computer vision

Această lucrare conține prezentarea unor implementări și tehnici, o abordare a soluționării problemei de super-rezoluție. În locul unei estimări pentru fiecare cadru în parte vom folosi o tehnică bazată pe folosirea unei rețele neurale recurente, ce va utiliza atât imaginile de rezoluție joasă, cât și ultima poză de rezoluție înaltă generată de arhitectură. Folosirea unei rețele recurente cât și a unei rețele de estimare a fluxului optic<sup>2</sup> aduce beneficii atât pe parcursul antrenamentului cât și pentru reconstrucția imaginilor.

## 1.2 Motivația alegerii temei

Obținerea unei imagini de o claritate înaltă este direct proporțională cu valoarea aparatelor prin intermediul cărora obținem aceste capturi. Prin abordarea acestei probleme prin intermediul deep learning-ului, obținem o soluție fiabilă pentru a rezolva aspectul de calitate-preț. Un bun exemplu în contextul vremurilor noastre este chiar smartphone-ul. Spre deosebire de o cameră performantă, accentul se pune pe un soft bun pentru a obține o imagine excelentă, ceea ce duce la un cost mai redus pentru utilizatorii ce vor investi într-un dispozitiv.

Alte soluții pentru a obține o imagine de rezoluție înaltă este prin intermediul software-urilor pentru calculator. Adobe a reușit în decursul acestui an să lanseze o nouă funcționalitate pentru unul dintre cele mai mari și folosite soft-uri de editare a imaginilor, Photoshop. Aceștia au adăugat, cu rezultate remarcabile, posibilitatea de redimensionare a imaginilor de la o rezoluție joasă la o rezoluție de 4 ori mai mare, îmbunătățind atât claritatea cât și calitatea imaginii.

Domeniul medicinei a avut de asemenea de beneficiat de pe urma tehniciilor de super-rezoluție prin procesarea de imagini medicale. Diagnosticarea unui pacient este o etapă crucială, uneori radiografiile acestora fiind neutilizabile din cauza rezoluției reduse. Super-rezoluția este însă folosită pentru a îmbunătăți imaginea folosită pentru o diagnosticare corectă.

Pe lângă domeniile menționate mai sus, lista poate fi extinsă în continuare observând o diversitate amplă a domeniilor ce au de câștigat de pe urma superrezoluției: astronomia, supravegherea prin camere video, televiziunea, microbiologia.

## 1.3 Contribuții personale

Punctul de plecare al acestei lucrări este reprezentată de propunerea oferită de Sajjadi et al. [14], ce reușesc să obțină rezultate foarte bune pentru problema de super-rezoluție aplicată pe videoclipuri.

---

<sup>2</sup>Din engleză: optical flow

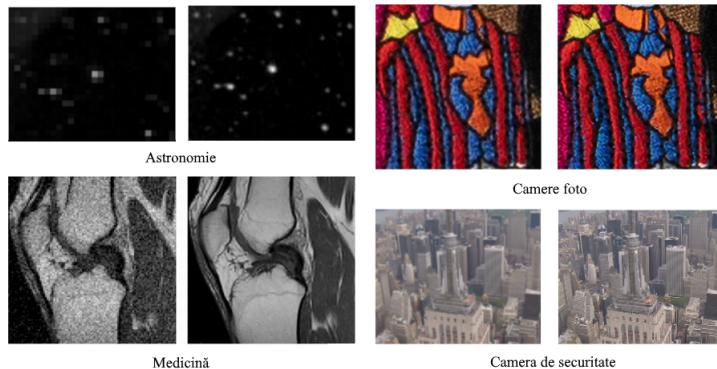


Figura 1.2: Super-rezoluție aplicată în diverse domenii.

Scopul acestei lucrări este unul experimental, prin realizarea unei implementări a modelului sugerat și aducerea de noi modificări asupra componentelor existente, pentru a observa comportamentele modelului atât la momentul antrenării cât și efectele asupra rezultatelor. Se dorește obținerea unui videoclip cu o claritate mai înaltă, prin trecea unui videoclip cu o rezoluție scăzută prin algoritmul de super-rezoluție.

Prima idee este cea de implementare a arhitecturii de bază propusă și procurarea setului de date necesar antrenării acesteia. Pasul de antrenare reprezintă învățarea caracteristicilor prezente în secvențele de cadre consecutive, scopul fiind reconstrucția cadrelor cu o rezoluție scăzută prin obținerea de detalii, pentru sporirea calității cadrelor.

O idee intuitivă este cea de schimbare a rețelelor din componența arhitecturii compuse. Prin înlocuirea cu rețele mai complexe sau cu modele pre-antrenate se dorește obținerea unor rezultate mai bune. Această idee conduce la crearea unor abordări noi, prin combinarea pașilor de construire a imaginilor de intrare, propusă de [14], cu rețele de super-rezoluție existente ce oferă rezultate remarcabile.

O altă idee este cea de introducere a unor componente noi în pasul de antrenare, prin adăugarea de noi funcții de cost orientate pentru probleme de superrezoluție. Astfel, prin preluarea și adaptarea acestor noi idei în lucrarea prezentată, rețeaua neurală artificială beneficiază de o creștere atât a calității video cât și a consistenței temporare pentru o secvență mai lungă de timp.

O ultima idee în dezvoltarea acestei aplicații constă în preluarea arhitecturi de bază propusă de [14], aducând modificări asupra structurii și a straturilor din componența sa. De asemenea prin intermediul observațiilor anterioare, vom introduce noi componente în structura arhitecturii ce sporesc calitatea videoclipurilor și ușurează procesul de antrenare.

## **1.4 Structura lucrării**

Lucrarea este structurată în următoarele capitole:

- **Capitolul 1** - scurtă introducere pentru familiarizarea cititorului cu tema aleasă;
- **Capitolul 2** - descriere amplă a subiectului ales, se va prezenta o scurtă introducere și evoluția în acest domeniu;
- **Capitolul 3** - prezentarea etapei de obținere a setului de date;
- **Capitolul 4** - prezentarea teoretică a elementelor interne din sfera rețelelor neurale artificiale folosite în lucrare;
- **Capitolul 5** - descrierea pe larg a modelelor propuse și a soluției finale;
- **Capitolul 6** - prezentarea tuturor tehnologiilor folosite în implementarea acestei lucrări;
- **Capitolul 7** - enunțarea concluziilor finale și posibilele direcții de implementare și dezvoltare a soluției curente.

## **1.5 Contribuțiile colaboratorului și co-supervizorului asupra temei**

Prezenta lucrare a fost realizată în colaborare cu S.C FotoNation by XPERI S.R.L, sub coordonarea conf. dr. Lucian-Mircea Sasu și co-supervizarea drd. Bogdan Mușat. Alegerea și construirea proiectului a fost realizată de comun acord cu părțile menționate, oferind suport din punct de vedere tehnic, cât și din punct de vedere al dobândirii cunoștințelor necesare pentru obținerea unei lucrări capabile.

## Capitolul 2

### Descrierea subiectului

#### 2.1 Ce este super-rezoluția?

Ideea care stă la baza rezolvării problemei de Super-Rezoluție (SR) este de a genera imagini "high resolution"<sup>1</sup> (HR), folosind imagini "low resolution"<sup>2</sup> (LR). O imagine HR oferă o densitate mai mare de pixeli, astfel aceasta va conține mai multe detalii din scena capturată.

Cadrele HR au o importanță semnificativă în computer vision, cu implicații directe în soluționarea altor sarcini precum analizarea unei imagini, "pattern recognition" sau "zooming".

Obținerea unor imagini HR nu este întotdeauna fezabilă, constrânsă atât de către prețul ridicat necesar aparaturii pentru capturare cât și de capabilitățile tehnologice limitate. O alternativă pentru evitarea acestor probleme sunt algoritmii de procesare a imaginii, ce oferă un cost redus cu rezultate satisfăcătoare.

Ramura super-rezoluției se împarte în două categorii:

- **Super-rezoluție pe o singură imagine (SISR<sup>3</sup>)** - obținerea unei fotografii HR dintr-o unică fotografie cu o claritate redusă;
- **Super-rezoluția aplicată pe video (VSR<sup>4</sup>)** - obținerea unui videoclip la o claritate ridicată, pornind de la cadrele unui videoclip cu o claritate redusă.

În timp ce pentru problema de SISR ne dorim reconstrucția detaliilor cu frecvență înaltă exclusiv din spațiul statistic, pentru VSR se pune problema relațiilor temporale dintre imaginile de intrare și cum acestea pot fi exploataate pentru a

---

<sup>1</sup>Imagini la o rezoluție înaltă

<sup>2</sup>Imagini la o rezoluție scăzută

<sup>3</sup>Din engleză: Single Image Super-Resolution

<sup>4</sup>Din engleză: Video Super-Resolution

îmbunătăți reconstrucția unui videoclip. Astfel este imperativă combinarea a cât mai multor cadre LR pentru a obține cel mai bun rezultat.

Problema pe care arhitecturile de tip Deep Learning încearcă să le rezolve o reprezentă aceea de înlocuire a algoritmilor tradiționali de super-rezoluție bazați pe metode de upscaling. Aceste metode clasice oferă rezultate ce duc lipsa detaliilor simple, totodată fără a putea elimina defecte sau artefacte de compresie.

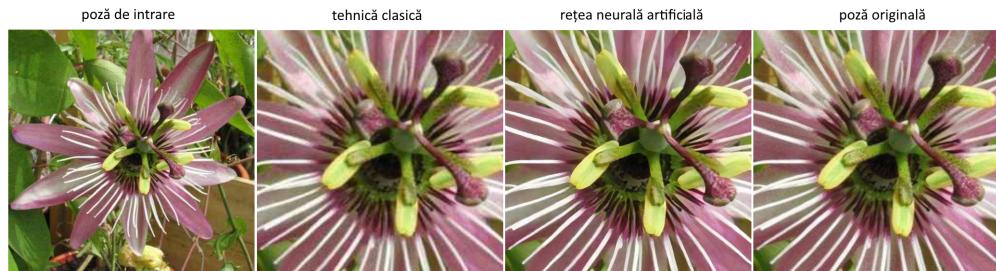


Figura 2.1: Diferența dintre tehniciile de super-rezoluție clasice și cele de tip Deep Learning.

În figura 2.1 putem observa, de la stânga spre dreapta, următoarele imagini:

- Imaginea LR pe care dorim să o rescalăm;
- Imaginea rezultată în urma aplicării unei tehnici clasice de super-rezoluție;
- Imaginea rezultată în urma aplicării unei tehnici de tip Deep Learning;
- Imaginea originală.

Obiectivul problemei de super-rezoluție este acela de a obține dintr-o fotografie LR un rezultat bun (chiar mai bun) comparativ cu imaginea target, cunoscută sub denumirea de ground truth.

## 2.2 Evoluția super-rezoluției folosind Deep Learning

O abordare clasică a problemei de super-rezoluție a fost folosirea unor metode simple de interpolare<sup>5</sup> precum cea biliniară, bicubică sau Lanczos [1]. În fiecare caz, pentru a determina valoarea unui pixel interpolat, trebuie să determinăm pixelul din imaginea de intrare căruia îi corespunde un pixel din imaginea de ieșire. Pentru determinarea valorii unui pixel din imaginea de ieșire se dorește

<sup>5</sup>Tehnică de bază din domeniul procesării de imagini, ce se ocupă de micșorarea imaginii prin intermediul unui filtru prestatibil și calcularea mediei pixelilor aflați sub acest filtru.

o aproximare a culorii și intensității pe baza valorii pixelilor din jurul acestuia. Cu cât numărul pixelilor luați în considerare este mai mare, cu atât efectul este mai precis, existând astfel un compromis între timpul de procesare și calitatea imaginii.

Însă pasul ce a redirecționat problemele de SR spre ramura Deep Learning-ul a fost Dong et al.[3]. Aceasta propune o metodă Deep Learning pentru rezolvarea problemei de SR, metoda constând în maparea end-to-end a imaginilor LR și HR. Folosind rețelele neurale de convoluție<sup>6</sup> (vezi secțiunea 4.3) se reușește obținerea unei imagini cu o claritate înaltă.

Trecerea de la metodele clasice la abordările de tip Deep Learning, a fost una rapidă datorată avantajelor oferite odată cu folosirea rețelelor inteligente:

- Obținerea unui rezultat rapid chiar și printr-o arhitectură simplistă;
- Oportunitatea de explorare a unor noi structuri și parametri, oferind libertatea de a experimenta posibile noi soluții.

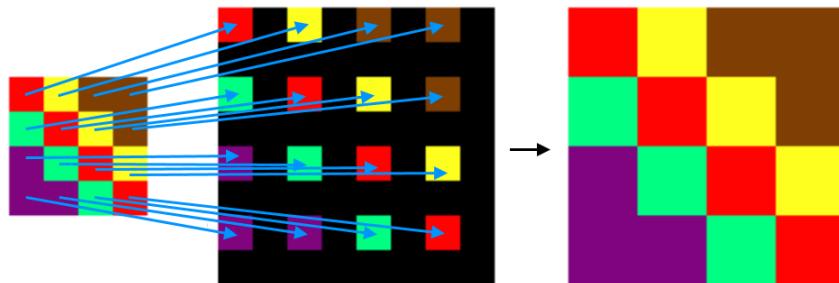


Figura 2.2: Operația de interpolare pornind de la o matrice de dimensiunile 4x4.

Găsirea unor soluții pentru VSR a dus la o abordare intuitivă, prin combinarea informațiilor din mai multe cadre LR și reconstruirea detaliilor lipsă din fiecare cadru individual, oferind un rezultat cu o claritate ridicată.

Interpretările clasice pentru VSR sunt în general expuse sub forma unor probleme de optimizare ce duc la un cost computațional ridicat de rezolvare. Majoritatea abordărilor Deep Learning folosite pentru această problemă împart, în general, sarcina de VSR în mai multe sub-sarcini, fiecare generând o singură imagine HR ce reprezintă ieșirea mai multor imagini de intrare LR.

---

<sup>6</sup>Din engleză: Convolutional Neural Networks (CNN)

Kappler et al. [7] propune o abordare în soluționarea problemei de VSR, utilizând rețele neurale de conoluție pentru extragerea informațiilor spațiale și temporare. Cadrele obținute dintr-un videoclip reprezintă intrările arhitecturii, aceste imagini fiind astfel o alternativă mai fiabilă în locul folosirii unui videoclip în procesul de antrenare. Se folosește de asemenea tehnica de “warping”<sup>7</sup> ce reprezintă deplasarea pixelilor cadrelor  $LR_{t-1}$  și  $LR_{t+1}$  către cadrul  $LR_t$  folosind o arhitectură de estimare a mișcării, concepută de Drulea și Nedevschi [8], concatenând astfel cele trei cadre și trecându-le printr-o rețea neurală de conoluție se obține estimarea cadrului  $I_{estt}$ .

Tao et al. [18] propun utilizarea unei intrări compusă din 7 cadre LR pentru estimarea unui singur cadrul HR. Odată cu obținerea mișcării din cadrele vecine ale unei imagini  $I_t^{LR}$ , se asociază cadrele astfel încât să se obțină o imagine la claritate înaltă. Ca pas final, se compune o arhitectură de tip encoder-decoder<sup>8</sup> ce folosește celule LSTM<sup>9</sup>. Astfel se demonstrează că folosirea cadrelor consecutive pentru obținerea informației compresate a mișcării reprezintă un pas crucial pentru obținerea unor videoclipuri cu o claritate înaltă.

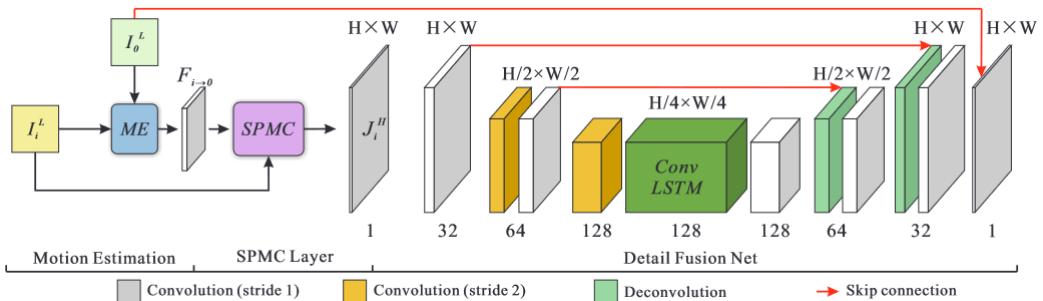


Figura 2.3: Schema modelului propusă de Tao et al. [2], ce are în componență: o porțiune cu rol în estimarea mișcării și compresarea informației prin metoda sub-pixel și arhitectură “Detail Fussion Net” bazată pe principiul encoder-decoder.

Lucas et al.[11] propune utilizare unei rețele adversarial generative, rețele cunoscute ce oferă rezultate remarcabile în problemele de generare de imagini. Astfel se introduc 2 rețele noi: un generator optimizat pentru problema de VSR (VSRResNet) și o nouă arhitectură pentru discriminator ce va avea ca rol ghidarea generatorului pentru a obține rezultate cât mai reale.

<sup>7</sup>Tehnică de deplasare a pozițiilor pixelilor dintr-o imagine de intrare către o imagine de ieșire, fără a modifica valorile acestora.

<sup>8</sup>Arhitectură a rețelei concepută din partea ce encoder, cu rolul în determinarea de informații esențiale și partea de decoder cu rol în decodificarea caracteristicilor extrase de către encoder.

<sup>9</sup>Din engleză: Long short-term memory

Arhitectura VSResNet propusă este compusă dintr-o serie de operații de conoluție pe 64 de canal și filtru<sup>10</sup> de 3x3 aplicat unei serii de 5 poze de intrare ( $LR_{t-2}, LR_{t-1}, LR_t, LR_{t+1}$  și  $LR_{t+2}$ , unde  $t$  reprezintă momentul din videoclip). Hărțile de caracteristici obținute vor fi concatenate pe dimensiunea canalelor și trecute mai departe prin 15 blocuri reziduale<sup>11</sup>. Pasul final constă în convertirea matricei de ponderi într-o imagine HR, acest lucru fiind realizat de o convoluție cu intrare de 64 de canale și ieșire pe 3 canale (RGB<sup>12</sup>).

Imaginea HR rezultată împreună cu imaginea ground truth, vor fi trecute pe rând prin discriminator, acesta returnând o probabilitate, ce denotă probabilitatea ca intrare oferită să reprezinte o poză reală. Discriminatorul este compus din perechi de tip conoluție-batch norm-funcție de activare, un strat complet conectat și funcția de activare sigmoidă (vezi secțiunea 4.2.4.1) ce va returna probabilitatea dorită.

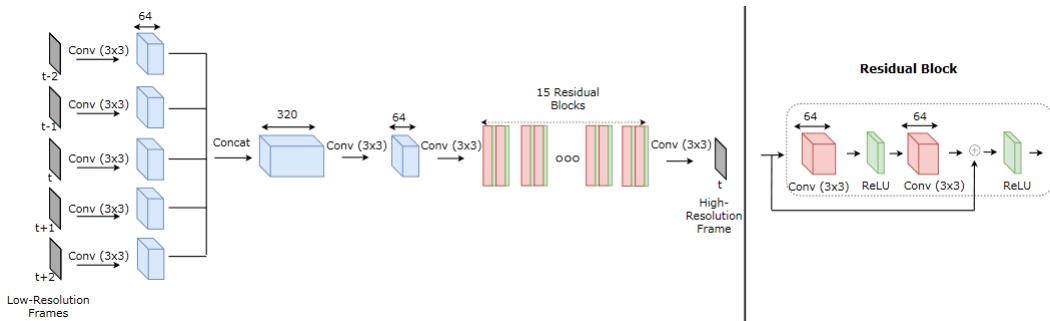


Figura 2.4: Arhitectura VSResNet pentru generator.

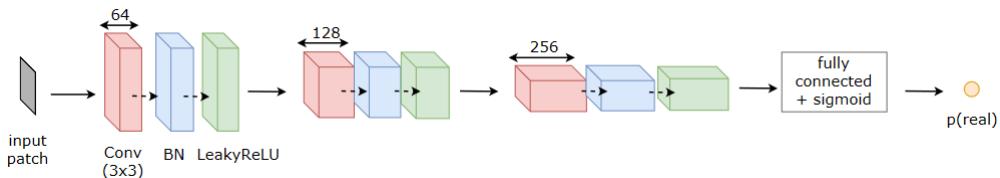


Figura 2.5: Arhitectura propusă pentru discriminator.

Arhitectura lui Sajjadi et al. [14] este cea cercetată în decursul acestei lucrări prin utilizarea, într-o manieră recurrentă, a cadrelor dintr-un videoclip LR, pentru obținerea unui videoclip HR (Frame-Recurrent Video Super-Resolution - FRVSR).

<sup>10</sup>Din engleză: kernel

<sup>11</sup>Din engleză: residual block

<sup>12</sup>Din engleză: Red, Green, Blue - reprezintă un format specifica imaginilor color, ce au în componentă 3 canale pentru culorile roșu, verde și albastru

În loc de estimarea individuală a fiecărui cadru din videoclip, această arhitectură reprezintă o abordare recurrentă ce folosește atât cadrele ce urmează a fi estimate, cât și cadrele ulterior estimate de către arhitectură.

Folosirea acestei maniere aduce un cost computațional mai redus, deoarece cadrele sunt procesate o singură dată. Mai mult de atât, informația obținută din cadrele generate anterior pot fi propagate în cadrele ce urmează a fi generate. Reutilizarea cadrelor are un rol important în estimarea următoarelor intrări, prin oferirea de detalii fine și păstrând consistență temporală pe parcursul videoclipului.

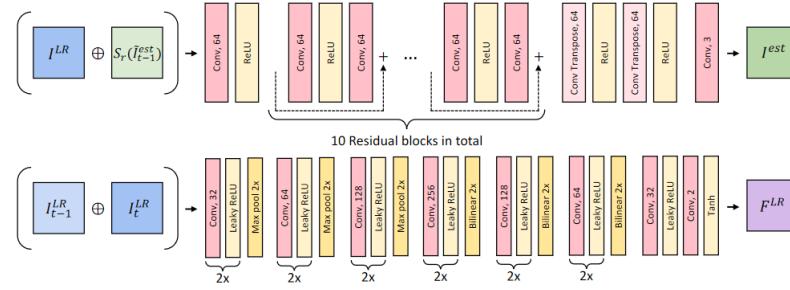


Figura 2.6: (Sus) Modelul propus pentru estimarea imaginilor cu o claritate înaltă. (Jos) Arhitectura de estimare fluxului optic dintre 2 imagini. Semnul  $\oplus$  denotă concatenarea imaginilor pe dimensiunea canalelor.

Pentru informația spațială, se introduce arhitectura ce se ocupă de determinarea fluxului optic, rolul acestuia fiind de identificare a matricei de transformare, ce conține direcția și cantitatea cu care un pixel referință din imaginea LR la timpul  $a_{t-1}$  s-a deplasat în imaginea LR la timpul curent  $a_t$ . Cu ajutorul acestei matrice se va aplica o operație de deplasare a pozițiilor pixelilor din ultima poză estimată la poza curentă, ce va aduce cadrul  $g_{t-1}$  la timpul curent. Informația obținută din urma acestei operații va reprezenta intrarea rețelei FRVSR, poza estimat (HR) urmând a fi refolosită în următoarea iterație, acest procedeu repetându-se pentru toate secvențele, câte  $n$  cadre, dintr-un videoclip.

# Capitolul 3

## Setul de date

Factorul decisiv pentru obținerea de rezultate bune sunt reprezentate de cantitatea și calitatea de date oferite în momentul antrenării unei arhitecturi. Prin urmare, înainte de a experimenta cu diferiți algoritmi, a fost necesar executarea unui proces de obținere și prelucrare a datelor, astfel încât să fie creat un set de date valid pentru problema de super-rezoluție aplicată pe videoclipuri.

În cazul problemei prezentate, este necesară construirea unui set de date ce conține în componență să atât imagini LR considerate imagini de intrare, cât și imagini HR considerate imagini reale. Pentru obținerea unui set de date corect și complet, trebuie satisfăcute următoarele caracteristici:

- **diversitatea datelor** - ne dorim crearea unui set de date obținut din cadrele unor videoclipuri ce conțin acțiuni lente, rapide, fundaluri statice dar și dinamice;
- **calitatea datelor** - imaginile LR și HR vor fi obținute din videoclipuri la o claritate înaltă, cu un număr cât mai ridicat de detaliu; se vor evita cele ce ar putea pune în dificultate procesul de antrenare;
- **cantitatea datelor** - obținerea unei cantități cât mai mare de date, folosite pentru procesele de antrenare, validare și testare rețelei neurale artificiale.

### 3.1 Obținerea setului de date

Problemele de SR și VSR duc lipsa unui număr mare de seturi de date publice, cele existente fiind puse cap la cap de cercetători prin obținerea de videoclipuri de pe diverse platforme multimedia precum Youtube sau Vimeo.

Un efort considerabil în implementarea acestei lucrări a fost direcționat către crearea unui set de date format din 30 de videoclipuri, ce reprezintă înregistrări

ale dronelor sau a camerelor auto de bord. Din aceste filmări au fost extrase mai multe secvențe de circa 5-6 secunde, din care au fost obținute 110 cadre consecutive per secvență, fiind ulterior salvate individual în directoare.

Tip	Dimensiune	Tip videoclip	Tip cadre	Acțiuni
Filmări drone	15 videoclipuri (1650 de cadre)	MP4	PNG	Acțiune lentă
Filmări camere de bord	15 videoclipuri (1650 de cadre)	MP4	PNG	Acțiune rapidă

Tabela 3.1: Videoclipuri obținute pentru setul de date. Tipul videoclipului reprezintă formatul video al fișierului. Tipul cadrelor prezintă formatul imaginilor. Acțiunea lentă/rapidă semnifică viteza de desfășurare a acțiunilor din videoclipuri.

Videoclipurile obținute prin intermediul dronelor reprezintă filmări din diverse orașe. Motivele ce au stat la baza alegerii acestor tipuri de cadre sunt acțiunea lentă și abundența detaliilor. Acestea vor juca rolul imaginilor folosite pentru antrenarea rețelei artificiale ce va transforma cadrele LR în cadre HR. În schimb videoclipurile obținute prin intermediul camerelor de bord, oferă acțiuni dinamice orientate pentru antrenarea modelului de estimare a fluxului optic. Pentru păstrarea unui echilibru pe parcursul antrenării au fost folosite un număr egal de videoclipuri din fiecare tip de filmare.

Cadrele au fost obținute prin intermediul bibliotecilor enunțate în secțiunea 5.3.5: youtube-dl pentru obținerea videoclipurilor de pe platforma Youtube și FFmpeg pentru împărțirea filmării în cadre.

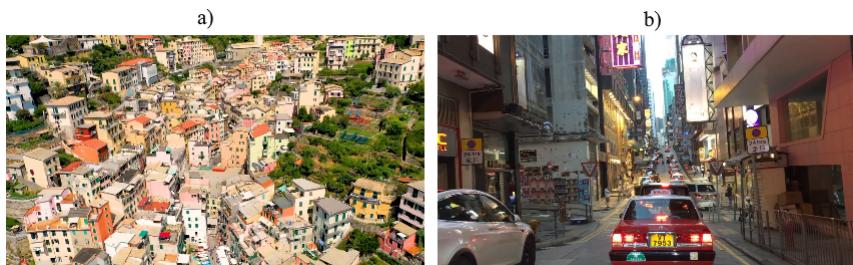


Figura 3.1: Exemple de cadre din setul de date creat: (a) imagini capturate cu ajutorul dronei, (b) imagini capturate cu ajutorul unei camere de bord.

## 3.2 Structura setului de date

Structura este formată din 3 subdirectoare, directorul "LR", "HR" și "scene". În directorul "scene" vor fi introduse foldere cu numele "scene\_xxxx" unde secvența "xxxx" reprezintă numărul directorului (ex: "scene\_0001"). Acest folder va conține cadrele videoclipurilor alese pentru antrenarea arhitecturii, în setul de date propus pentru fiecare filmare vom avea 110 cadre cu formatul "PNG"<sup>1</sup> și dimensiunea de 1280x720.

Prin procesarea acestora, se vor obține secvențe de 10 cadre LR și HR, cadrele LR vor avea extensia "PNG" cu dimensiunea de 32x32 și vor fi salvate în subdirectoare individuale, în directorul intitulat "LR". Cadrele HR vor avea extensie "PNG" cu dimensiunea 128x128 și vor fi salvate în subdirectoare individuale, în directorul denumit "HR". Aceste imagini reprezintă porțiuni decupate din cadrele originale.

Obținerea imaginilor și prelucrarea acestora înaintea pasului de antrenare aduce beneficii în timpul antrenării, oferind eficiență din punct de vedere al timpului dar și al memoriei necesare. Există desigur un compromis prin alocarea de resurse în memoria locală pentru păstrarea setului de date pe perioada de experimentare.

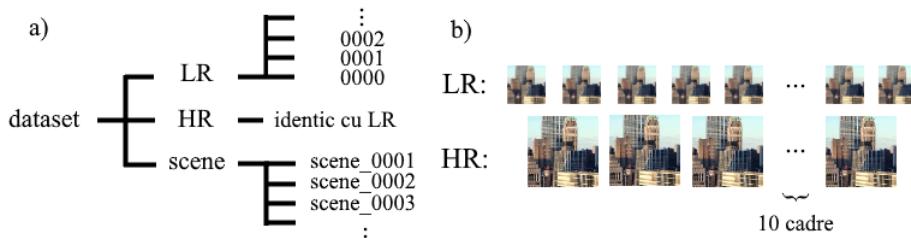


Figura 3.2: (a) Structura directorului ce conține setul de date (directorului principal intitulat "dataset"). Cele 3 subdirectoare ce conțin imaginile LR, HR și originale. (b) O secvență de imagini LR respectiv HR formată din 10 cadre consecutive obținute din cadrele originale.

## 3.3 Prelucrarea setului de date

În etapa de prelucrare a datelor se vor obține secvențele LR și HR din cadrele videoclipurilor propuse anterior.

<sup>1</sup>Portable Network Graphics - reprezintă un format de fișier specific imaginilor ce acceptă compresia datelor fără a exista pierderi.

Pentru crearea unei secvențe de imagini HR se vor folosi 10 cadre sursă consecutive ce vor fi micșorate prin înjumătățirea dimensiunilor, astfel cadrele vor fi aduse de la o rezoluție de 1280x720 la o rezoluție de 640x360. Se va genera aleatoriu o poziție încadrată în dimensiunea imaginii și se vor calcula 4 puncte ce reprezintă coordonatele colțurilor unui pătrat cu dimensiunea de 128x128. Prin decuparea acestui pătrat din imaginea sursă, folosind aceleași coordonate pentru toate cele 10 cadre, vom obține o secvență de tip HR. Acestea se vor salva într-un subdirector, în directorul denumit "HR".

Procesul de obținere a secvenței de imagini LR constă în refolosirea celor 10 cadre HR obținute în pasul anterior. Acestea vor fi redimensionate cu un factor de 4, obținând cadre cu rezoluția de 32x32. Sajjadi et al. [14] recomandă aplicarea unui blur gaussian pe cadrele LR, acest lucru îmbunătățind performanța antrenării oferind rezultate mai bune în pasul de generare de cadre SR. Pe cadrele obținute va urma aplicarea pasului de augmentare<sup>2</sup>, prin modificare valorilor de luminanță, contrast și saturatie, ulterior obținând cadrele LR finale. Aceste secvențe vor fi salvate în subdirectoare individuale, în directorul denumit "LR".

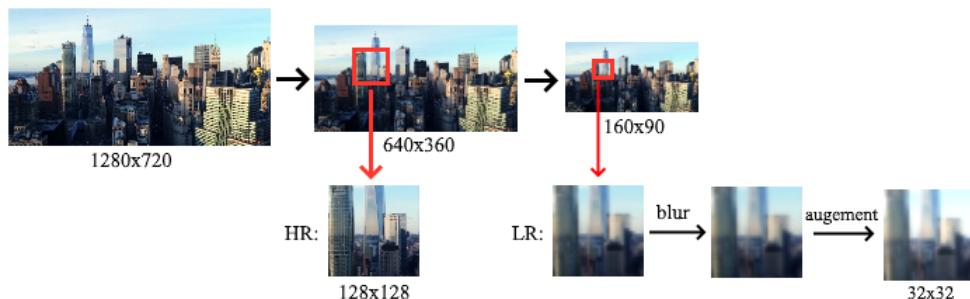


Figura 3.3: Pașii de obținere a secvențelor LR respectiv HR, din cadrele originale.

### 3.3.1 Împărțirea setului de date

În urma obținerii setului de date, acesta este împărțit în trei subseturi: setul de antrenare, de validare, respectiv setul de testare. Setul de antrenare este format din datele necesare algoritmului, sau modelului, pentru învățarea diferitelor caracteristici. Setul de validare are ca rol evaluarea corectă a modelului în timpul antrenamentului, etapă în care dezvoltatorii vor face reglarea hiperparametrilor. Pentru realizarea unei evaluări cât mai precisă și imparțială în urma antrenamentului, este necesar un set de testare.

<sup>2</sup>Tehnică de expandare a setului de date prin modificări aduse asupra imaginilor setului existent.

Această împărțire a datelor în cele trei subseturi enunțate, trebuie făcută cu grijă astfel încât să existe un raport corect între datele de antrenare și cele de validare.

Tipul de subset	Procent din setul integral	Numărul de cadre din setul integral
Antrenare	90%	2970 cadre
Validare	5%	165 cadre
Testare	5%	165 cadre

Tabela 3.2: Împărțirea setului de date.

# Capitolul 4

## Brevier teoretic pentru rețelele neurale artificiale

### 4.1 Introducere

#### 4.1.1 Inteligența artificială, Machine Learning și Deep Learning

Odată cu posibilitatea programării unui calculator, oamenii au început să-și adreseze întrebări precum “*cum?*” și “*când?*” aceste mașinării vor deveni inteligențe. Astăzi, inteligența artificială este o ramură activă pentru cercetători, întâlnită și utilizată atât pentru aplicațiile practice de tip soft, cât și în diverse dispozitive (numite și agenți inteligenți). Un dispozitiv are un comportament intelligent atunci când este capabil să interpreteze mediul înconjurător, urmând a efectua o acțiune pentru a duce la înndeplinirea unui obiectiv.

Automatizarea muncii bazate pe principiul de rutină, înțelegerea vorbirii, interpretarea imaginilor și diagnosticarea în medicină, sunt câteva ramuri pe care cercetătorii încearcă să le înțeleagă prin intermediul inteligenței artificiale. Deși simplu ca și concept, sarcinile intuitive sunt greu de definit și interpretat prin intermediu software, provocarea acestui domeniu constând în învățarea de către calculator a comportamentelor umane.

**Machine learning** este un subdomeniu al inteligenței artificiale ce se focalizează pe interpretarea și imitarea comportamentelor umane prin intermediul unor date de intrare. Acesta folosește raționamente matematice, statistică și algoritmi, pentru a găsi tipare în datele puse la dispoziție. Astfel machine learning prevede antrenarea algoritmilor prin intermediul datelor oferite, pentru a lua ulterior decizii. Orice informație ce poate fi stocată digital, poate fi de asemenea considerată dată de intrare pentru un algoritm de machine learning (ex: numere, text, imagini, fișiere video, fișiere audio etc.)

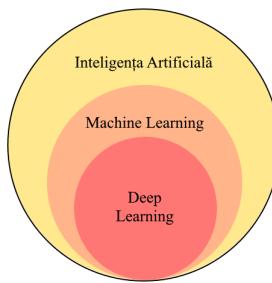


Figura 4.1: Relațiile dintre Inteligența Artificială, Machine Learning și Deep Learning

**Deep learning** este o ramură a machine learning-ului ce oferă posibilitatea calculatorului de a interpreta conceptele complexe prin înțelegerea și combinarea mai multor concepte simple.

Putem împărți o arhitectură simplă de tip deep learning în trei componente importante:

- **Straturi vizibile** - conțin variabile ce pot fi interpretate de către programator (trăsături ce sunt încă observabile din datele de intrare);
- **Straturi ascunse** - o serie de straturi în care algoritmul identifică propriile concepte prin care urmează să interpreteze datele de intrare;
- **Stratul de ieșire** - rezultatul obținut în urma interpretării.

În Figura 2.2 se poate observa cum o arhitectură deep learning poate determina sexul unei persoane prin învățarea conceptelor precum margini (edges), contururi, portiuni din fețe etc.

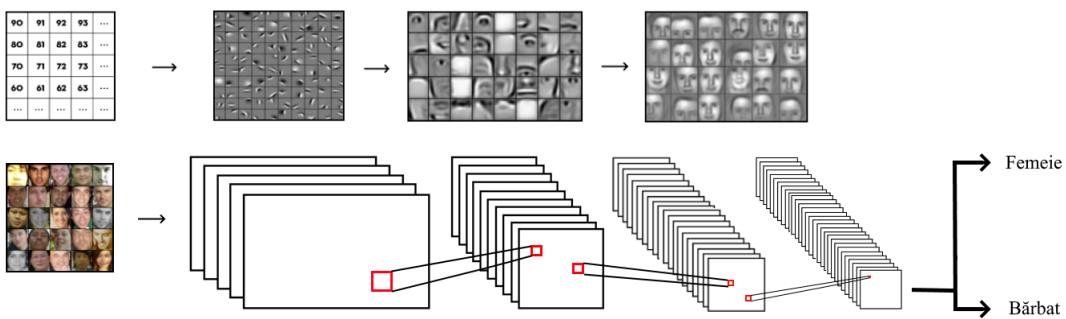


Figura 4.2: Clasificarea pozelor după gen prin intermediul unei arhitecturi Deep Learning.

#### 4.1.2 Tipuri de învățare

Procesul de învățare reprezintă acțiunea de procesare a informațiilor primite. Pornind de la informațiile provenite, sistemul intelligent are ca scop îndeplinirea, cu un rezultat cât mai bun, a sarcinii oferite. Există trei tipuri principale de învățare:

- **supervizată**
- **nesupervizată**
- **prin întârrire**

Pentru **învățarea supervizată**, un utilizator numit și "profesor" va avea datoria de a superviza algoritmul pentru a primi, prin intermediul unor date de intrare, estimări cât mai apropiate de adevăr. Acest tip de învățare este ușor de remarcat datorat datelor primite, având o formă "mostră-etichetă"<sup>1</sup>. Rezultatul urmărit este reprezentat de o estimare cât mai bună a unei valori de ieșire, pornind de la date noi de intrare. Procesul manual de construire a acestor seturi de date este unul laborios, constând în etichetarea corectă a datelor de intrare (această acțiune fiind recomandată de a fi realizată de către un specialist din domeniul de proveniență a datelor oferite).

**Învățarea nesupervizată** spre deosebire de cea supervizată, presupune identificarea unor tipare pornind de la datele de intrare, fără a fi specificate ieșirile corespunzătoare. Această sarcină este mai dificilă datoră lipsei etichetelor sau a unei critici asociate. În timp ce pentru învățarea supervizată etichetele ofereau algoritmului posibilitatea de a găsi relații între datele oferite, învățarea nesupervizată se axează pe identificarea unor structuri ascunse, percepute de algoritm într-o manieră abstractă, fără a fi nevoie o intervenție umană.

**Învățarea prin întârrire** presupune o interacțiune cu mediul înconjurător, cu scopul de a obține un feedback constant între acțiune și experiența dobândită. Utilizatorul astfel devine un critic în procesul de învățare, algoritmul fiind penalizat sau premiat de pe urma acțiunii săvârșite (necunoscând care sunt de fapt acțiunile dorite). Intuiția biologică din spatele acestui tip de învățare este una corectă, astfel algoritmul încercă să minimizeze din factorii penalizatori, ulterior maximizând numărul acțiunilor ce au dus la o premieră.

**Învățarea semi-supervizată** este specificată de asemenea de anumite surse, aceasta reprezentând un tip hibrid între învățarea supervizată și cea nesupervizată. Sistemului i se vor furniza informații despre ieșirile asociate, însă acestea vor fi incomplete. Scopul sistemului este de a genera date în locurile incomplete.

---

<sup>1</sup>Din engleză: sample-label

### 4.1.3 Aplicabilitatea rețelelor neurale artificiale

- **Clasificare** - pe baza unui set de date de forma intrare-ieșire, modelul va detecta asocieri între datele de intrare și etichete (acestea vor avea rol de clase), valorile estimate vor fi discrete și finite;
- **Regresie** - datele oferite vor fi identice cu cele ale problemei de clasificare, însă datele de ieșire vor fi reprezentate de valori numerice continue;
- **Clustering** - spre deosebire de problemele anterioare, datele oferite vor fi limitate doar la cele de intrare, scopul fiind de a identifica similarități, datele urmând să fie segregate în grupuri pe baza trăsăturilor abstracte identificate de către model;
- **Ranking** - acest tip de problemă are ca scop clasarea informațiilor dintr-o listă într-o anumită ordine, inducând un scor ordinal sau valori binare discrete (ex: "relevant/irrelevant");
- **Identificarea de asocieri** - pe baza unui set amplu de date, modelul va identifica diferențe asocieri complexe între datele oferite;

## 4.2 Concepte de bază

### 4.2.1 Neuronul artificial

Neuronul artificial denumit și **perceptron** (denumire introdusă de Frank Rosenblatt) este piesa ce stă la baza construirii unei rețele neurale artificiale. Algoritmul pentru perceptron reprezintă totodată o rețea neurală cu un singur strat, formată din patru componente importante: valorile de intrare, ponderi, bias și funcția de activare.

Perceptronul primește diferențe date venite din lumea reală,  $x_1, x_2, \dots$ , ce vor produce o singură valoare de ieșire binară. În exemplul prezentat în Fig 4.3, perceptronul are  $n$  intrări, numerotate cu 1,  $x_1, x_2, \dots, x_n$ . Rosenblatt propune o regulă simplă pentru obținerea valorii de ieșire, astfel introducând o listă de ponderi, numerotate cu  $w_0, w_1, w_2, \dots, w_n$ . Acestea reprezintă valori reale, fiecare stabilind importanța intrării asociate pentru valoarea de ieșire.

Neuronul de ieșire va returna o valoare, 0 sau 1, determinată de depășirea sau nedepășirea unui prag de către valoarea obținută de pe urma sumei  $\sum_0^i w_i x_i$ . La fel ca și ponderile, pragul va primi o valoare reală, ce reprezintă un alt parametru al perceptronului. În acest caz, pragul va purta rolul funcției de activare, astfel obținând următoarea formă algebraică:

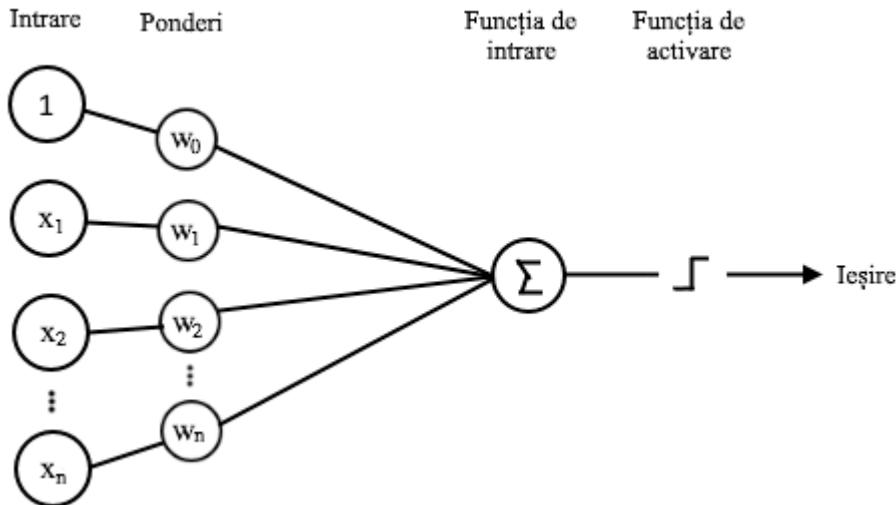


Figura 4.3: Structura unui neuron artificial.

$$iesire = \begin{cases} 0 & \text{dacă } \sum_0^j w_i x_i \leq \text{prag} \\ 1 & \text{dacă } \sum_0^j w_i x_i > \text{prag} \end{cases} \quad (4.1)$$

În general funcția de activare are o formă mai complexă, scopul fiind de a încadra rezultatul într-un anumit interval real. Acestea vor fi descrise pe larg în secțiunea 4.2.4.

#### 4.2.2 Structura unei rețele neurale artificiale

Prin dispunerea neuronilor pe verticală obținem diferite varietăți de rețele numite rețele neurale multistrat. O rețea multistrat se compune din cel puțin 3 straturi:

- **stratul de intrare** - valorile de intrare în rețea;
- **stratul ascuns** - pentru rețele neurale multistrat vor exista un minim de un strat ascuns compus din neuroni;
- **strat de ieșire** - compus din neuroni ce produc valorile estimate.

În arhitectura evidențiată în Fig 4.4, prima coloană de neuroni reprezintă primul strat ascuns, ce va avea rol în luarea unor decizii simple pe baza importanței datelor de intrare. În cel de-al doilea strat se remarcă luarea unui număr mai

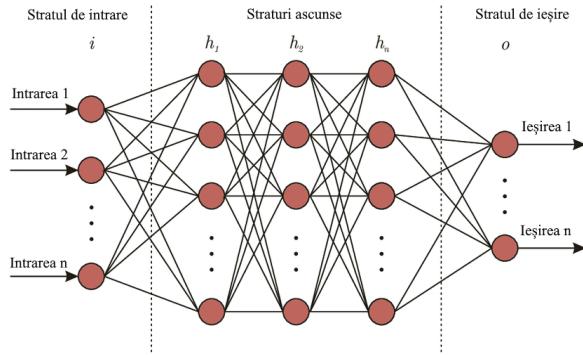


Figura 4.4: Structura unei rețele neurale artificiale multi-strat.

mare de decizii, bazate pe importanța ieșirilor anterioare, astfel obținem decizii mai complexe. Intuiem faptul că ieșirile ulterioare vor oferi decizii din ce în ce mai sofisticate spre deosebire de neuronii straturilor precedente. Astfel straturile ascunse reprezintă motorul computațional într-o arhitectură MLP<sup>2</sup>.

Stratul final este numit și strat de ieșire ce conține rezultatul problemei dorite (ex: predicția unei valori numerice sau clasificare). Observăm astfel că intrările dintr-un strat anterior vor fi propagate în stratul următor, această acțiune purtând numele de feedforward<sup>3</sup>.

#### 4.2.3 Propagarea înainte

Pasul de propagare înainte este un component important, reprezentând punctul de plecare în antrenarea unui algoritm. Din exterior remarcăm faptul că aceste rețele neurale artificiale sunt de fapt reprezentări complexe prin intermediul unor funcții matematice, ce încearcă să aproximeze o valoare dorită pe baza unei/unor variabile de intrare oarecare.

Putem exprima următoarea analogie:

$$y = f(x) \quad (4.2)$$

unde variabila "y" va lua rolul variabilei de ieșire, "x" a variabilelor de intrare, iar "f" funcția de aproximare utilizată în rețea neurală artificială. Pasul de propagare înainte reprezintă astfel aplicarea unei funcții asupra variabilelor de intrare.

Complexitatea rețelelor neurale artificiale va fi direct proporțională cu nivelul complexității problemelor care necesită a fi rezolvate. Această creștere a nivelului complexității se poate face în diferite moduri, în secțiunile ce urmează a fi

<sup>2</sup>Din engleză: Multi Layer Perceptron - Rețea neurală multi-strat

<sup>3</sup>propagare înainte

enunțate diferite abordări folosite în rezolvarea problemelor complexe cu ajutorul rețelelor neurale artificiale.

#### 4.2.4 Funcții de activare

Funcțiile de activare au ca scop introducerea modificărilor non-liniare în calcul, deoarece problemele din lumea reală nu pot fi modelate doar prin intermediul transformărilor liniare oferite de straturile cu neuroni simple. Astfel prin introducerea funcțiilor non-liniare (funcții cu un grad mai mare), graficul va fi reprezentat prin intermediul unei curbe (spre deosebire de o linie în cazul funcțiilor de gradul 1).

În continuare vom enunța unele dintre cele mai populare funcții de activare utilizate în cadrul rețelelor neurale artificiale:

- Funcția sigmoid;
- Funcția tangentă hiperbolică;
- Funcția ReLU;
- Funcția LeakyReLU;
- Funcția Softmax.

##### 4.2.4.1 Funcția Sigmoid

Funcția sigmoid este o funcție mărginită monotonă, caracterizată de graficul acestea, având formă asemănătoare cu litera "S". Această funcție are proprietatea de a uniformiza valorile în intervalul  $[0, 1]$ . Pentru problemele de estimare de probabilități condiționate valoarea obținută din urma funcției sigmoid va fi interpretată ca o probabilitate.

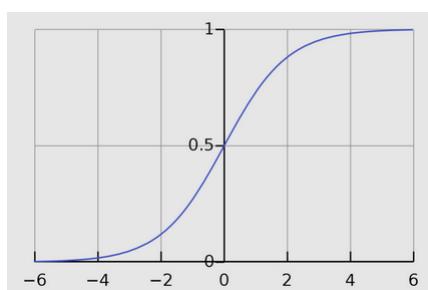


Figura 4.5: Graficul funcției sigmoid.

Formula funcției sigmoid este:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (4.3)$$

Derivata acestei funcții este:

$$f'(x) = f(x) \cdot (1 - f(x)) \quad (4.4)$$

#### 4.2.4.2 Funcția tangentă hiperbolică

Funcția tangentă hiperbolică, foarte asemănătoare cu funcția sigmoid, reprezintă una dintre funcțiile clasice folosite în componența rețelelor neurale artificiale. Această funcție uniformizează valorile în intervalul [-1,1], oferind avantaje precum continuitate, derivabilitate și aducând adâncime în marja de valori, spre deosebire de funcția prezentată precedent. În practică funcția tangentă hiperbolică oferă rezultate mai bune decât sigmoida logistică.

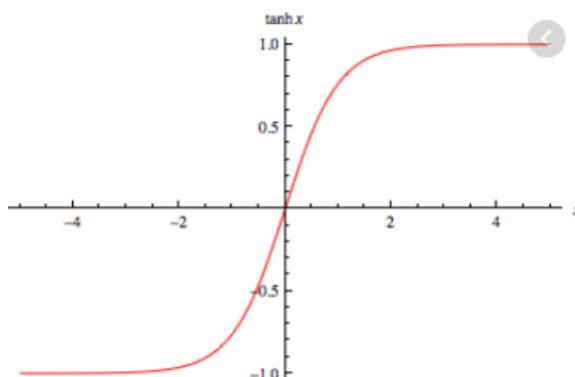


Figura 4.6: Graficul funcției tangentă hiperbolică.

Formula funcției tangentă hiperbolică este:

$$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \quad (4.5)$$

Derivata acestei funcții este:

$$f'(x) = 1 - \tanh^2(x) = 1 - f^2(x) \quad (4.6)$$

#### 4.2.4.3 ReLU

Funcția Rectified Linear Unit (ReLU), reprezintă cea mai populară și utilizată funcție de activare, cu o aplicabilitate pe o arie largă în domeniul rețelelor neurale artificiale. Spre deosebire de funcțiile prezentate anterior, în special comparativ

cu tangenta hiperbolică, s-a demonstrat faptul că ReLU reușește să aducă îmbunătățiri la rata de convergență, ajungând la rezultate de până la 6 ori mai bune, reușind în același timp să ofere simplitate și eficiență.

În general această funcție de activare este recomandată a fi folosită în interiorul straturilor ascunse ale unei rețele neurale artificiale, pentru straturile de ieșire existând funcții de activare specializate în funcție de tipul de problemă și tipul variabilelor de ieșire.

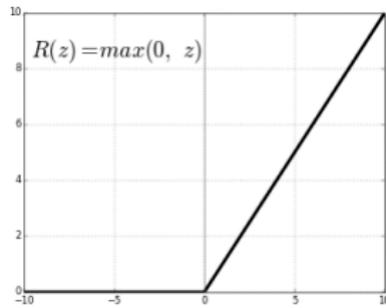


Figura 4.7: Graficul funcției ReLU.

Formula funcției ReLU este:

$$f(x) = \max(0, x) = \begin{cases} 0 & \text{dacă } x \leq 0 \\ x & \text{dacă } x > 0 \end{cases} \quad (4.7)$$

Ecuația funcției reprezintă valoarea maximă între valoarea 0 și valoarea curentă, astfel valorile negative fiind aduse la valoarea 0, după cum se poate observa în graficul din figura 3.7 .

Două aspecte importante ale funcției de activare ReLU sunt:

- **păstrarea interacțiunilor între variabilele de intrare** - determinarea unei valori prin intermediul acestei funcții de activare, va conduce la o dependență a variabilelor de intrare, astfel valoarea obținută poate fi 0 (în cazul în care suma tuturor variabilelor înmulțite cu valorile ponderilor asociate are o valoare negativă) sau va fi egală cu rezultatul sumei (în cazul în care aceeași sumă are o valoare pozitivă);
- **introducerea non-liniarității** - o funcție este considerată neliniară atunci când panta sa nu este constantă; indiferent de liniaritatea funcției pe o anumită porțiune, acesta oferă un caracter neliniar în ansamblu; în practică atunci când o rețea neurală artificială este antrenată, aceasta depinde de mulți parametri care influențează schimbarea pantei.

#### 4.2.4.4 Funcția Leaky ReLU

Funcția LeakyReLU reprezintă una dintre variațiile funcției ReLU, ceea ce înseamnă că majoritatea caracteristicilor vor aduce aceleași beneficii, însă cu următoarele mențiuni:

- Introduce o pantă diferită de valoare 0 pentru valorile negative;
- Valoarea de 0 este înlocuită cu o valoare apropiată de 0, ceea ce ajută la eliminarea problemei de anulare a gradienților;
- Pentru această funcție s-a semnalat o accelerare în procesul de antrenare.

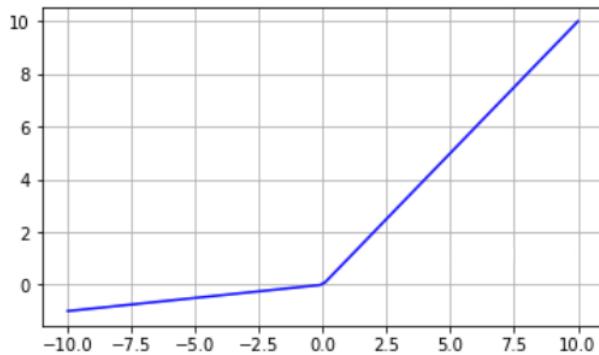


Figura 4.8: Graficul funcției LeakyReLU.

Formula funcției LeakyReLU este:

$$f(x) = \begin{cases} x & \text{dacă } x \geq 0 \\ \alpha \cdot x & \text{dacă } x < 0 \end{cases} \quad (4.8)$$

#### 4.2.4.5 Funcția Softmax

Funcția Softmax este utilă atunci când se dorește normalizarea unui vector cu valori oarecare într-o distribuție de probabilitate, având proprietatea ca toate valorile vectorului de ieșire să fi cuprinse în intervalul (0,1), iar prin însumarea acestor valori se va obține valoarea 1.

Funcția Softmax are formula:

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}} \quad (4.9)$$

unde  $x$  reprezintă vectorul de intrare,  $e^{x_i}$  reprezintă exponențiala vectorului de intrare,  $K$  reprezintă numărul claselor al clasificatorului, iar  $e^{x_j}$  exponențiala de scalare.

Din acest motiv, spre deosebire de funcția ReLU unde există limitări în utilizarea acesteia doar pe straturile ascunse (așa cum am precizat în secțiunea 4.23.2), funcția Softmax este un bun candidat în componența stratului de ieșire, fiind o funcție specializată în funcție de tipul de problemă ce se dorește a fi rezolvată.

#### 4.2.5 Funcția de cost

După cum am enunțat anterior, acest proces reprezintă optimizare a unor parametri încât nivelul aproximării sau aprecierea calității să fie realizate cu ajutorul funcțiilor de cost (numite și funcții de eroare). Astfel, rolul atribuit în procesul de învățare este de a estima cât de bine modelează funcția de optimizare relația dintre datele care intră și cele de ieșire.

În funcție de problema aleasă, există diverse funcții de cost, unele mai generice, fiind folosite pentru a rezolva o gamă mai largă de probleme, altele fiind construite special doar pentru anumite tipuri de probleme. Cele mai populare funcții de cost au următoarele caracteristici:

- **Mean Squared Error** - cunoscută sub numele de MSE, acestă funcție măsoară media pătratică a erorilor; obținerea valorii de eroare presupune calcularea mediei pătratelor diferențelor dintre valoarea de adevăr și valoarea estimată de către rețea;

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (4.10)$$

unde  $y_i$  și  $\hat{y}_i$  reprezintă valoarea de adevăr, respectiv valoarea estimată, iar  $n$  reprezintă numărul total de date.

- **$L_1$  Loss** - cunoscută și sub forma de minim absolut al derivației, este o funcție populară în domeniul Deep Learning, folosită în ramuri precum computer vision; această funcție măsoară suma diferențelor absolute dintre valoarea de adevăr și valoarea estimată de către rețea;

$$L1Loss = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (4.11)$$

- **Cross Entropy** - folosită cu precădere pentru probleme de clasificare, calculează performanța modelului de clasificare binară sau multiplă; valoarea obținută de pe urma acestei funcții de cost crește cu cât probabilitatea prezisă se depărtează față de valoarea de adevăr (obținerea valorilor apropiate de zero rezultă într-o estimare cât mai corectă).

Ecuăția funcției de cost Cross-Entropy pentru problema de clasificare binară:

$$CrossEntropy = -(y \cdot \log(p) + (1 - y) \cdot \log(1 - p)) \quad (4.12)$$

respectiv pentru problema de clasificare multiplă:

$$CrossEntropy = - \sum_{c=1}^M y_{o,c} \log(p_{o,c}) \quad (4.13)$$

unde  $y$  și  $p$  reprezintă valoarea de adevăr, respectiv valoarea prezisă de rețea, iar  $M$  reprezintă numărul de clase.

#### 4.2.6 Propagarea înapoi a erorii

Algoritmul de propagare înapoi a erorii (Rumelhart et al.[13]) permite informația de pe urma funcției de cost de a fi propagată invers prin rețea, scopul fiind obținerea gradientelor.

Calculul prin "chain rule"<sup>4</sup> (a nu se confunda cu metoda cu aceeași denumire din domeniul probabilităților) este o metodă de calcul a derivatelor unei funcții, compusă din alte funcții ale căror derive sunt cunoscute. Back-propagation este un algoritm ce folosește într-o manieră recursivă această metodă, operațiile fiind făcute în aşa fel încât eficiența să fie cât mai bună.

La baza acestui algoritm se află conceptul de derive parțiale pentru optimizarea parametrilor rețelei. Prin calcularea derivatelor parțiale în raport cu parametrii rețelei se obține cantitatea din eroarea totală cu care trebuie modificați acești parametri dar și direcția (pozitivă sau negativă). În urma acestei modificări se dorește minimizarea valorii obținute prin intermediul funcției de cost.

---

<sup>4</sup>Regula de derivare a funcțiilor compuse

#### 4.2.7 Strategii de optimizare

În procesul de antrenare există cazuri în care durata reprezintă un compromis în operațiunea de obținere a unui rezultat satisfăcător, deoarece durata de antrenare cât și cea de testare a rețelei, de către dezvoltator, poate dифeri în funcție de problema aleasă. Deși timpul nu reprezintă un impediment în procesul de antrenare al unui model, totuși este necesară o reducere a acestuia.

Performanța algoritmului de optimizare reprezintă un factor important când vine vorba de obținerea unui algoritm de antrenare performant. Prin folosirea unor tehnici speciale se pot combate probleme precum:

- o durată mare ce poate apărea în procesul de optimizare;
- majoritatea problemelor de optimizare sunt non-convexe, ceea ce duce la existența unor puncte multiple de minim, ce duc la îngreunarea procesului de optimizare deoarece algoritmul trebuie să determine dacă punctele respective reprezintă un minim local sau global.

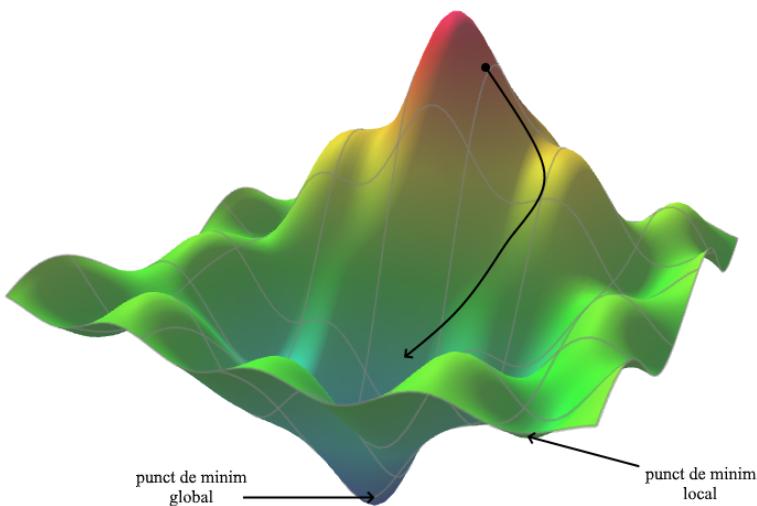


Figura 4.9: Direcția pasului de optimizare. Punct de minim global și puncte minime locale.

##### 4.2.7.1 Algoritmul de optimizare Adam

Atunci când vine vorba de a alege un algoritm de optimizare potrivit, algoritmul lui Kingma et al.[8] întâlnim sub numele de Adam, este des folosit în literatura de specialitate și în implementările acestora, reprezentând unul dintre cei mai performanți algoritmi de optimizare existenți.

Această performanță este datorată combinării unor concepe importante folosite în implementările altor algoritmi de optimizare:

- o rată de învățare adaptabilă individual pentru fiecare pondere în parte care aduce îmbunătățiri în performanță atunci când majoritatea valorilor gradientilor se apropiu (sau sunt chiar egale) de valoarea de 0;
- o rată de învățare adaptabilă ce ține cont de modificările recent aduse pentru fiecare parametru în parte.

Pentru o optimizare corespunzătoare prin intermediul algoritmului Adam, este necesară ajustarea următorilor hiperparametri:

- **Rata de învățare** - o valoare mare poate determina o antrenare mai rapidă, însă există posibilitatea de a rata punctul de minim, în timp ce o valoare prea mică poate duce la un proces de antrenare mult mai lent, uneori cu posibilitatea redusă de a ajunge la un punct de minim;
- **Parametrul Epsilon ( $\epsilon$ )** - reprezintă o marjă adaugată pentru a preveni o posibilă împărțire la valoarea 0 (valoarea standard este  $1e - 8$ );
- **Parametrul beta ( $\beta_1, \beta_2$ )** - reprezentat de un tuplu ce conține doi coeficienți folosiți pentru calcularea mediei exponentiale și pătratul a mișcării gradientilor;
- **Parametrul beta** - reprezentat de un tuplu ce conține doi coeficienți folosiți pentru calcularea mediei exponentiale și pătratul a mișcării gradientilor;
- **Parametrul weigh\_decay** - reprezintă o termen de penalizare L2 cu rolul de a preveni fenomenul de overfitting și a păstra valorile ponderilor mici pentru evitarea fenomenului de explozie a gradientilor.

#### 4.2.8 Overfitting și Underfitting

Provocarea domeniului de Machine Learning, respectiv Deep Learning, este reprezentată de capacitatea algoritmului de a oferi performanță pe seturi de date noi, diferite de datele de intrare pe care acesta le-a primit în momentul antrenării. Abilitatea de a obține performanțe bune pentru intrări noi poartă numele de **generalizare**.

În general, atunci când dezvoltatorul dorește antrenarea corectă a unui model de machine learning, acesta va avea acces la un set de date separat, folosit doar pentru sarcina de antrenare. Valoarea de eroare obținută din urma procesului

de antrenare (dintre valoarea estimată și cea de adevăr) nu reprezintă un criteriu corect pentru aprecierea calității modelului.

Separarea unei probleme de machine learning de o problemă de optimizare este realizată de valoarea erori în momentul generalizării, numită și eroare la testare. Această eroare este definită ca eroare la momentul inferenței unor date nemaivăzute de rețea.

Eroarea pentru generalizare va fi estimată primăscurarea performanței modelului pe un set nou de date, numit set de testare. În general acest set de testare reprezintă o proporție din setul de antrenare, date ce nu vor fi oferite pentru procesul de antrenare. În mod ideal, rezultatele obținute de pe urma metricilor de eroare pentru un astfel de set de date ar trebui să fie cât se poate de apropiat de cele obținute pentru setul de date de testare.

Există două concepte ce reprezintă o piedică pentru fenomenul de generalizare: overfitting și underfitting. Aceste două concepte sunt cunoscute și des întâlnite în domeniul de Deep Learning, reprezentând probleme reale în dezvoltarea de noi soluții folosind rețele neurale artificiale.

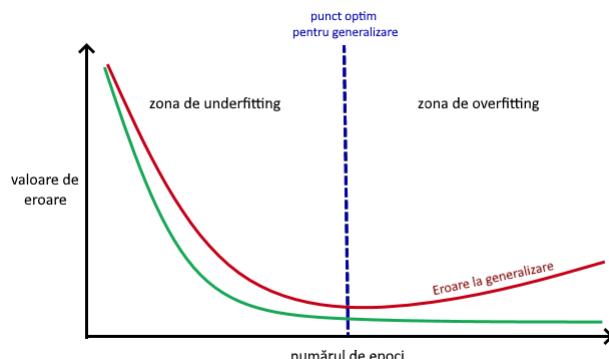


Figura 4.10: Exemplu de grafic al unui model ce face overfitting. Linia de culoare verde reprezintă funcția de cost pentru setul de antrenare, iar cea roșie funcția de cost pentru setul de validare.

#### 4.2.8.1 Overfitting

Noțiunea de overfitting se referă la capacitatea modelului de a oferi o generalizare eronată pe setul de validare sau cel de testare, deoarece în timpul antrenării modelul a învățat foarte bine setul de date de antrenare. Acest aspect se poate observa ușor prin verificarea valorilor metricilor de eroare obținute pe setul de antrenare, respectiv pe cel de testare. Pentru setul de antrenare vom obține un scor mult mai mare decât scorul obținut de pe urma setului de testare.

O metodă des întâlnită pentru identificarea overfitting-ului este studierea graficului funcției de cost în raport cu numărul de epoci, se va face o comparație între graficul obținut atât de pe urma setului de date din timpul antrenării, cât și pentru cel de testare. După cum se poate observa în figura(vezi mai sus), linia punctată reprezintă momentul în care s-a atins valoarea de eroare potrivită pentru un set de date oferit (antrenare sau testare). Creșterea funcției de cost după acestă linie marcată în imagine, demonstrează faptul că modelul începe să învețe particularitățile acestui set de date și generalizează slab pe setul de testare și validare.

Câteva metode ce ajută la evitarea fenomenului de overfitting sunt:

- antrenarea modelului pe mai multe exemple
- schimbarea complexității modelului prin modificarea cu un număr mai mic a numărului de straturi sau a numărului de parametri al straturilor
- prin diverse procedee de regularizare (prezentate în secțiunea următoare)
- adăugare de zgomot statistic pentru intrările din timpul antrenării

#### 4.2.8.2 Underfitting

Noțiunea de underfitting se referă la incapacitatea modelului de a învăța setul de date de antrenare rezultând într-o generalizare incorectă. Fenomenul de underfitting poate fi identificat folosindu-se aceeași metodă prezentată mai sus. Urmărind graficele erorilor pentru setul de date de antrenare, respectiv cel de validare, se observă valorile slabe ale metricilor de eroare (vezi figura 4.11).

Câteva metode ce ajută la evitarea fenomenului de underfitting sunt:

- schimbarea complexității modelului prin modificarea cu un număr mai mare a numărului de straturi
- creșterea numărului de parametri pe straturile ascunse
- reducerea termenului de regularizare, penalizarea fiind mai ușoară

#### 4.2.9 Regularizare

După cum am enunțat în secțiunea anterioară, există o discrepanță între soluționarea problemei de underfitting și overfitting, unde metodele de rezolvare sunt mai complexe iar în majoritatea cazurilor rezultatele fiind nesatisfăcătoare.

Introducerea unor date noi reprezintă un proces dificil, având în vedere faptul că pentru anumite tipuri de probleme, numărul de seturi de date și datele

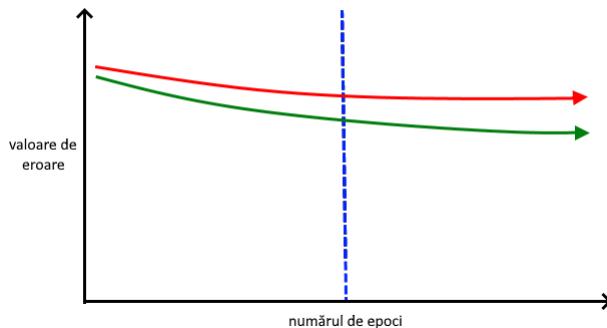


Figura 4.11: Exemplu de grafic al unui model ce face underfitting. Linia de culoare verde reprezintă funcția de cost pentru setul de antrenare, iar cea roșie funcția de cost pentru setul de validare.

oferte de seturi sunt într-un număr redus. Astfel, pe lângă timpul necesar construirii unui set de date prin adnotare manuală și preprocesare de date, cel mai mare impediment îl reprezintă procesul de achiziție de date.

Astfel, singura soluție fiabilă pentru obținerea unor rezultate bune pentru diferitele probleme îl reprezintă folosirea metodelor de regularizare. În cele ce urmează vor fi prezentate unele metode de regularizare existente, folosite în îmbunătățirea procesului de antrenare:

- Batch Normalization
- Weight Normalization
- Spectral Normalization

#### 4.2.9.1 Batch Normalization

Standardizarea datelor reprezintă un pas important înainte de a începe procesul de antrenare al unei rețele neurale artificiale, valorile datelor fiind transpusă într-o distribuție a cărei medie este egală cu valoarea 0, iar variația egală cu valoarea 1. Acest pas are ca scop prevenirea saturării activărilor non-liniare din straturile apropiate stratului de intrare a datelor.

Însă nu e de neglijat faptul că distribuția valorilor se schimbă pe parcursul antrenamentului, ca și consecință a diferitelor operații care se realizează pe straturile ascunse, ceea ce aduce probleme precum saturarea activărilor în interiorul acestor straturi. Această problemă poate fi rezolvată odată cu introducerea conceptului de Batch Normalization [5].

Utilizarea acestei tehnici în cadrul rețelelor neurale inteligente duc la o accelerare a antrenamentului, creșterea performanței dar și o stabilizare mult mai bună prin intermediul normalizării valorilor de intrare pentru fiecare batch al fiecarui strat din interiorul rețelei. Pașii matematici ce stau la baza logicii acestei funcții sunt următorii:

- Determinarea mediei și variației variabilelor de intrare pentru fiecare strat;
- Se standardizează valorile de intrare folosind rezultatele obținute în pasul precedent;
- Se standardizează valorile obținute din urma aplicării funcției de activare, folosindu-se parametrii  $\alpha$  și  $\beta$  (parametrii ce sunt învățați în timpul antrenării).

#### 4.2.9.2 Weight Normalization

Deși Batch Normalization reprezintă o soluție bună pentru îmbunătățirea antrenării unei rețele neurale artificiale, aceasta nu duce lipsa unor dezavantaje:

- o aproximare corectă este în corelație cu dimensiunea batch-ului, acesta purtând un rol important în determinarea unor valori potrivite pentru medie și variație (un batch mai mare rezultă într-o aproximare mai bună);
- nu oferă rezultate promițătoare pentru rețelele neurale recurente (vezi secțiunea 4.4);

Ca o combatere a acestor dezavantaje, Tim Salimans și P.Kingma [15] prezintă o variație a tehnicii batch norm, ce poartă denumirea de Weight Normalization. Această tehnică constă în reparametrizarea vectorului de ponderi dintr-o rețea neurală artificială prin decuplarea lungimii acestui vector de direcția lui. Prin intermediul noii metode de calcul aduce îmbunătățiri mărind viteza de convergență a algoritmului de optimizare.

Autorii sugerează introducerea a doi parametrii noi:  $g$  ce reprezintă lungimea vectorului de ponderi și  $v$  ce reprezintă direcția același vector. Aceștia vor înlocui vectorul de ponderi prin următoarea formulă:

$$\mathbf{w} = \frac{g}{\|\mathbf{v}\|} \mathbf{v} \quad (4.14)$$

Spre deosebire de tehnica batch norm, Weigh Normalization reușește să mărească viteza de antrenare, un plus fiind posibilitatea de aplicabilitate în rețelele neurale recurente. Însă metoda aduce și dezavantaje, fiind semnificativ mai instabilă pentru rețelele neurale adânci și evitată, în practică, pentru acest tip de problemă.

#### 4.2.9.3 Spectral Normalization

Pe parcursul antrenării unei rețele generativ adversariale<sup>5</sup> întâlnim diverse impiedimente, cea mai mare provocare fiind instabilitatea acestui tip de model. Însă Miyato et al. [12] reușesc să promoveze o nouă tehnică ce aduce o mai bună performanță, variație a tehnicii Weight Normalization, ce poartă numele de Spectral Normalization. Aceasta reușește să ofere o stabilizare a modelului de discriminator (referință) în procesul de antrenare, cu rezultate bune nu doar pentru acest tip de problemă.

Raționamentul matematic din spatele tehnicii weight normalization, implică un conflict de interes între normalizarea ponderilor și dorința de a folosi cât mai multe caracteristici (feature's) posibile, pentru a distinge distribuția generatorului de cea a discriminatorului.

Spectral Normalization este soluția pentru această problemă, prin reducerea dimensiunii vectorul de ponderi folosind normalizarea spectrală (normalizarea L2), având proprietatea de a avea un singur hiper-parametru ce trebuie ajustat de către dezvoltator.

$$\mathbf{W}_{SN} = \frac{\mathbf{W}}{\sigma(\mathbf{W})}, \sigma(\mathbf{W}) = \max_{\mathbf{h}: \mathbf{h} \neq 0} \frac{\|\mathbf{Wh}\|_2}{\|\mathbf{h}\|_2} \quad (4.15)$$

### 4.2.10 Antrenarea

#### 4.2.10.1 Batch Size

Pentru înțelegerea conceptului de batch size putem introduce, mai întâi de toate, noțiunea de exemplu al setului de date. Un set de date reprezintă o mulțime de intrări ce se pot afla în diverse forme: valori numerice, text, imagini. Un set de date este compus din mai multe astfel de exemple, purtând numele de instanță sau vector de caracteristici.

Astfel, noțiunea de batch size reprezintă un hiperparametru ce semnifică dimensiunea vectorului de intrare (numărul de exemple) ce vor urma a fi simultan procesate de către o rețea artificială.

Un set de date de antrenare poate fi împărțit în unul sau mai multe batch-uri, depinzând de capacitatea de procesare existentă sau de limitările modelului folosit. Mărimea unui batch este aleasă de către dezvoltator, existând următoarele 3 opțiuni:

- **Batch Gradient Descent** - executarea tuturor exemplelor simultan, întâi pașii de propagare înainte și calcul de gradienți pentru fiecare exemplu în parte, ulterior executând actualizarea tuturor ponderilor;

---

<sup>5</sup>Din engleză: Generative Adversarial Network (GAN)

- **Mini-batch Gradient Descent** - executarea exemplelor în subgrupuri, setul de date fiind împărțit în grupuri de dimensiunea valorii hiper-parametrului ales de către dezvoltator;
- **Stochastic Gradient Descent** - executarea particulară a exemplelor, actualizarea ponderilor se va face pentru fiecare instanță.

#### 4.2.10.2 Numărul de epoci

Numărul de epoci reprezintă un alt hiperparametru ce are ca scop definirea de către dezvoltator a numărului de parcurgeri a întregului set de date. Astfel, o epocă reprezintă o parcurgere totală a setului de date, fiind luată în calcul în momentul în care se vor actualiza ponderile.

În funcție de dimensiunea batch-ului, o epocă poate fi reprezentată de unul sau mai multe batch-uri. În practică, acest hiperparametru nu are o valoare standardizată, luând diferite valori în funcție de adâncimea și numărul de parametri ai rețelei sau mărimea setului de date.

#### 4.2.10.3 Rata de învățare

Pentru problemele de Machine Learning, rata de învățare reprezintă un hiperparametru cu rol în a seta intensitatea pasului de optimizare pentru fiecare iterare. Astfel, rata de învățare determină viteza cu care o rețea neurală inteligentă va învăța.

Determinarea unei valori potrivite pentru acest hiperparametru are o importanță majoră în procesul de antrenare al unui model, reprezentând un compromis între rata de convergență și fenomenul de depășirea a minimului global.

Pentru obținerea unei convergențe potrivite, prevenirea oscilațiilor sau posibilitatea de a rămâne blocat într-un punct de minim local, rata de învățare este variată pe parcursul antrenării, acest lucru fiind realizat cu ușurință prin utilizarea unui planificator al ratei de învățare. Dezvoltatorul are posibilitatea de a defini tipul de planificator utilizat, astfel încât să servească nevoilor fiecărei probleme în parte.

#### 4.2.11 Metrici de performanță

Pasul de evaluare al unei rețele neurale artificiale este esențial pentru confirmarea unor rezultate corecte. Totodată, pentru dezvoltarea unor noi soluții mai performante, este necesar un criteriu de departajare al rețelelor în competiție, lucru realizat prin intermediul unor metrici ce măsoară performanța acestora. În funcție de tipul problemei, dar și al tipurilor de date utilizate pentru antrenare,

există o diversitate de metriki cu ajutorul cărora pot fi comparate soluțiile diferitelor ramuri din domeniul Deep Learning.

În domeniul computer vision, pentru problema ce implică procesarea de imagini, interesul este reprezentat de măsurarea similarității dintre imaginea estimată și imaginea adevărată. Astfel de metriki sunt dezvoltate și folosite pentru determinarea performanței rețelelor neurale. Cele mai cunoscute astfel de metriki, folosite în măsurarea similarității sunt:

- Peak Signal-To-Noise Ratio
- Structural-to-Noise Index

#### 4.2.11.1 Peak Signal-to-Noise Ratio

Termenul de peak signal-to-noise ratio (prescurtat PSNR) reprezintă metrika ce determină raportul dintre o valoare maximă posibilă și valoarea de corupere (zgomot) ce afectează calitatea unei reprezentări, în cazul nostru o imagine. Datorită valorilor aflate într-un interval mai larg (interval definit de cea mai mică și cea mai mare valoare a semnalului), PSNR în general este exprimată prin intermediul scării logaritmice de decibeli ( $dB$ ).

Îmbunătățirea calității unei imagini digitale poate fi subiectivă, de aceea este necesară stabilirea unei metriki cantitative pentru compararea îmbunătățirilor rezultate cu ajutorul rețelelor neurale inteligente.

La baza raționamentului matematic pentru determinarea acestei similarități, se folosește media erorii pătratice (MSE), ce reprezintă o medie aritmetică a tuturor diferențelor dintre valorile de adevăr( $Y_i$ ) și valorile estimate ( $\hat{Y}_i$ ) de către rețea, ridicate la pătrat:

$$MSE = \frac{1}{n} \sum_{i=0}^n (Y_i - \hat{Y}_i)^2 \quad (4.16)$$

Astfel pentru metrika PSNR, obținem următoarea ecuație:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \|f(i, j) - g(i, j)\|^2 \quad (4.17)$$

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned} \quad (4.18)$$

- $f$  - matrice de valori a pozei referință;
- $g$  - matrice de valori a pozei degradate (estimată);
- $MAX_I$  - valoarea maximă posibilă a unui pixel din imaginea adevărată.

#### 4.2.11.2 Structural Similarity Index

Structural Similarity Index (prescurtat SSIM) reprezintă o metrică ce determină similaritatea a două imagini, pe baza a trei valori: luminanță, contrast și structură. Sistemul vizual uman este adaptat pentru a extrage informații structurale, însă algoritmul SSIM separă similaritatea pe baza acestor 3 valori.

$$SSIM(x, y) = [l(x, y) \cdot c(x, y) \cdot s(x, y)] \quad (4.19)$$

unde  $l(x, y)$  reprezintă valoarea luminanței,  $c(x, y)$  valoarea de contrast, iar  $s(x, y)$  valoarea structurală.

Valoarea luminanței dintre două semnale este determinată cu ajutorul mediei de intensitate a tuturor semnalelor, contrastul este determinat prin deviația standard, iar structura din corelația dintre cele două semnale. Pentru determinarea valorilor celor 3 variabile ne vom folosi de următoarele ecuații:

$$\begin{aligned} l(x, y) &= \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \\ c(x, y) &= \frac{\sigma_x\sigma_y + c_2}{2\sigma_x^2 + \sigma_y^2 + c_2} \\ s(x, y) &= \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \end{aligned} \quad (4.20)$$

unde  $\mu$  reprezintă media imaginii oferite,  $\sigma$  reprezintă deviația standard,  $\sigma_{xy}$  reprezintă covarianța dintre imagini, iar  $c_1, c_2, c_3$  sunt constante cu rol de stabilizare a împărțirii la o valoare mică.

## 4.3 Rețele neurale artificiale de convoluție

Apariția rețelelor de convoluție (LeCun, 1989) cunoscute și cu numele de rețele neurale convecționale(CNN), au reprezentat un pas esențial în evoluția arhitecturii și soluțiilor problemelor de computer vision, fiind inspirată din organizarea cortexului vizual al creierului.

### 4.3.1 Strat conoluțional

Stratul conoluțional valorifică trei proprietăți importante ce ajută la îmbunătățirea sistemelor automate inteligente:

- **sparse connectivity** - straturile neurale artificiale tradiționale folosesc operații de multiplicare a matricelor de parametrii, unde fiecare valoare reprezintă o interacțiune între unitatea de intrare și cea de ieșire; prin aplicarea repetitivă a unui filtru pe suprafața imaginii de intrare, obținem trăsături importante stocate cu ajutorul unui număr redus de parametrii, micșorând memoria necesară și îmbunătățind eficiența;

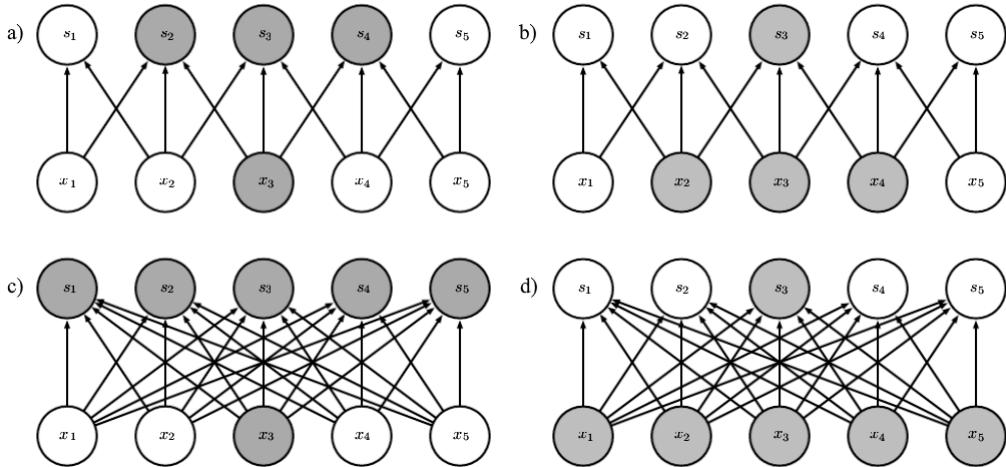


Figura 4.12: Exemplu de aplicabilitate a proprietății sparse connectivity. Se evidențiază efectul unității de intrare  $x$  pentru ieșire  $s$ . (a) Prin folosirea unei conoluții cu un filtru de dimensiunea 3, o intrare  $x$  va avea efect asupra unui număr de 3 unități de ieșire  $s$ , (b) iar o ieșire va fi afectată de 3 intrări. (c) Pentru straturile tradiționale, fiecare intrare  $x$  va avea importanță asupra tuturor unităților de ieșire  $s$ , (d) iar o ieșire  $s$  va fi afectată la rândul ei de toate intrările  $x$

- **partajarea parametrilor** - refolosirea parametrilor pentru mai multe funcții dintr-un model; într-o rețea neurală artificială tradițională fiecare element dintr-o matrice de ponderi este folosit o singură dată pentru determinarea ieșirii din următorul strat, astfel valorile ponderilor pentru o singură intrare sunt în strânsă legătură cu valorile ponderilor aplicate în altă parte a arhitecturii (se învăță un set de parametrii pentru fiecare locație); în schimb rețelele conoluționale aduc beneficii din punct de vedere al spațiului de stocare necesar și al eficienței, prin învățarea unui singur set de parametrii;

- **reprezentări echivariante** - o funcție este echivariantă atunci când prin modificarea unităților de intrare vom obține modificări pentru unitățile de ieșire; mai precis, o funcție  $f(x)$  este echivariantă pentru o funcție  $g$  dacă  $f(g(x)) = g(f(x))$ .

Stratul conoluțional are rolul de a aplica operația de conoluție asupra imaginilor de intrare, fiind format din 3 stagi. În primul stagiu se vor calcula mai multe conoluții în paralel pentru a produce un set de funcții liniare. Stagiul al doilea constă în aplicarea unei funcție de activare non-liniară (vezi secțiunea 4.2.4) pentru fiecare funcție liniară obținută precedent. Pentru stagiul final se va folosi un strat de pooling pentru reducerea ieșirii stratului de conoluție. Rezultatele obținute prin intermediul operației de conoluție poartă denumirea de hărți de caracteristici.

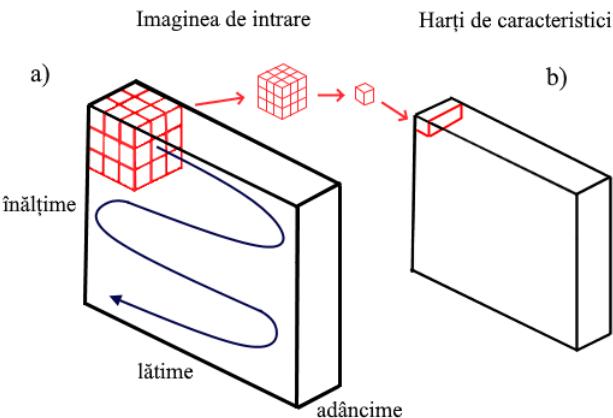


Figura 4.13: (a) Exemplu de traseul al unui filtru pe o imagine de intrare, adâncimea reprezentând numărul de canale. (b) Prin aplicarea unui filtru obținem hărți de caracteristici cu o dimensiune mai mică spre deosebire de imaginea de intrare.

Bordarea este de asemenea o operație specifică stratului de conoluție, reprezentând un parametru ce mărește dimensiunea imaginilor de intrare prin adăugarea de valori pe marginile acestora (în general cel mai des folosit este valoarea 0). Scopul acestei bordări este de a controla dimensionalitatea rezultatului obținut în urma operației de conoluție.

### 4.3.2 Strat de pooling

Stratul de pooling reprezintă o componentă unei rețele neurale artificiale de conoluție, cu rolul de reducere a dimensiuni hărților de caracteristici prin intermediul unui filtru prestabilit de dezvoltator. Această reducere a dimensiunilor se realizează prin extragerea de caracteristici dominante.

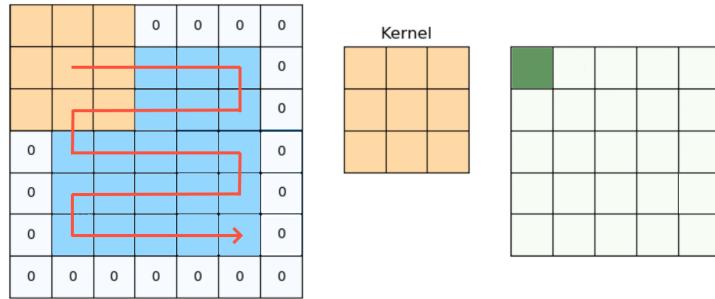


Figura 4.14: Prin trecerea unui filtru cu dimensiunea  $3 \times 3$  pe o matrice de  $5 \times 5$  înconjurate cu valori de 0, se obține o imagine de ieșire de aceeași dimensiune.

Cele mai des utilizate straturi de pooling sunt:

- **Max Pooling** - se realizează prin extragerea valorii maxime din întreaga regiune acoperită de filtru;
- **Average Pooling** - se realizează prin calcularea mediei aritmetice a tuturor valorilor aflate în regiunea acoperită de filtru.

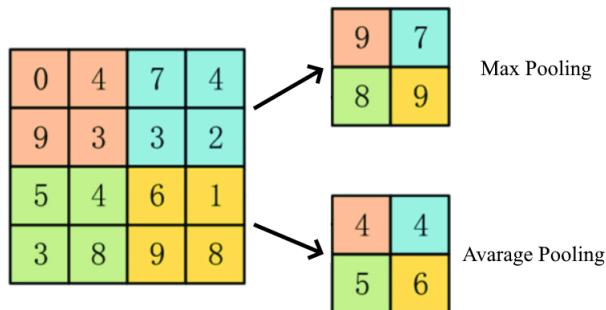


Figura 4.15: Exemple de aplicarea a celor două straturi de pooling.

## 4.4 Rețele reziduale recurente

Prin apariția arhitecturii AlexNet [9], compusă din 5 straturi conoluționale, intuiția a dus la crearea unor arhitecturi cu un număr mai mare de straturi precum VGG[16] ce folosesc 19 straturi conoluționale. Însă simpla creșterea a numărului de straturi nu reprezintă o soluție fiabilă datorită problemei de blocarea a gradienților la valoarea 0.

Rețele neurale artificiale reziduale sunt o subclasă a rețelelor neurale artificiale ce au la bază ocolirea unor straturi în timpul antrenării (skip connections). Aceasta rezolvă problema blocării gradientilor, prin folosirea scurtăturilor ce permit gradientului de a trece mai departe prin arhitectură. Un alt beneficiu îl reprezintă uniformizarea importanței straturile, astfel încât straturile adânci se beneficiază de o importantă asemănătoare cu cea a straturile de la începutul unei rețele adânci.

O rețea reziduală va fi compusă astfel din mai multe blocuri reziduale. Introduse ca parte a arhitecturii ResNet, reprezintă blocuri de tip skip connection ce au ca rol învățarea unor funcții de identitate. Pentru un block rezidual a cărui intrare este reprezentată de  $x$  vom dori învățarea unei distribuții  $H(x)$ , astfel prin ecuația  $R(x) = \text{iesire} - \text{intrare} = H(x) - x$  se obține o valoare pe care o numim **reziduu**. Prin rearanjarea acestei ecuații, vom obține  $H(x) = R(x) + x$  (observăm faptul că un bloc rezidual încearcă de fapt să învețe  $R(x)$ ).

În timp ce într-o rețea neurală tradițională se încearcă învățarea într-un mod direct a valorii de adevăr  $H(x)$ , straturile dintr-o arhitectură reziduală vor avea rolul de a învăța unui reziduu  $R(x)$  ce se adaugă la o intrare  $x$  pentru a obține valorii de adevăr  $H(x)$ .

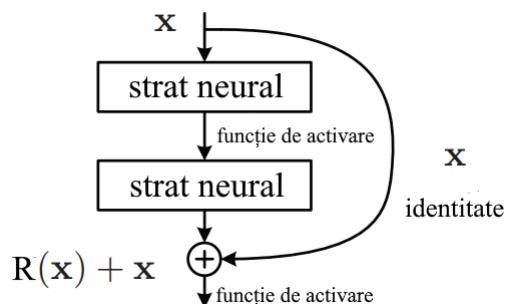


Figura 4.16: Exemplu de bloc rezidual folosit în arhitectura ResNet.

Unul dintre avantajele aduse de acest tip de arhitectură îl reprezintă realizarea legăturilor semantice între informația precedentă și cea curentă, cu aplicabilități în special pentru probleme ce au la bază folosirea unor videoclipuri. Utilizarea unui cadru anterior dintr-un videoclip poate oferi informații necesare în înțelegerea cadrului curent.

# Capitolul 5

## Modele investigate

Arhitectura folosită în această lucrare este inspirată din Sajjadi et al.[14], scopul fiind unul experimental, prin observarea de comportamente odată cu modificări aduse asupra rețelelor artificiale propuse.

Primul pas în dezvoltarea soluției pentru problema VSR a constat în replicarea și antrenarea modelului, pentru a obține în urma trecerii unui videoclip LR, un videoclip la o claritate mai înaltă prin adăugarea de detalii asupra cadrelor. De asemenea, un efort semnificativ a fost depus în reproducerea cât mai fidelă a pașilor de pregătire și construire a setului de date și a intrărilor necesare rețelei de super-rezoluție. Cu toate acestea, odată cu antrenarea acestui model s-a sesizat o diferență pentru valorile numerice rezultate.

Arhitectură	PSNR	SSIM
FRVSR	22.96	0.79
FRVSR_IP	17.67	0.48

Tabela 5.1: Comparația între rezultatele obținute de arhitectura FRVSR[14] și implementarea proprie (FRVSR\_IP).

Posibilele motive ce au stat la baza obținerii acestei diferențe ar putea fi date:

- Setul de date diferit: setul de date oferit de autorii articolelor este incomplet (videoclipurile au fost șterse de pe platformă), astfel singura soluție a constat în crearea unui set propriu de date pentru antrenarea, validare și testare;
- Antrenare: a fost realizată prin intermediul hiper-parametrilor precizați în articol, însă setul de date joacă un rol important în pasul de replicare a unui model;

- Diferențe de implementare: apariția unor diferențe în implementarea și optimizarea codului propriu spre deosebire de implementarea propusă de autori.

Deși există această diferență a rezultatelor, modelul implementat este suficient de capabil în a fi utilizat în scop experimental, pentru observarea diferitelor comportamente odată cu introducerea de noi elemente precum funcții de cost sau modificarea rețelelor de SR, respectiv estimare de flux optic.

Pornind de la rezultatele obținute, scopul este de a îmbunătăți valorile numerice prin înlocuirea cu diferite tipuri de modele derivate:

- SRGAN [10]
- SRDenseNet [19]
- RDN [21]

## 5.1 Arhitecturi abordate

### 5.1.1 Arhitectura SRGAN

Arhitectura SRGAN [10], provine din ramura arhitecturilor ce folosesc capacitatele și beneficiile aduse de arhitecturile GAN, oferind rezultate remarcabile printr-o generare cât mai naturală a detaliilor.

Problema ce se încearcă a fi rezolvată de acest tip de arhitectură este cea de SISR, prin estimarea unei imagini  $I^{SR}$ , imagine SR determinată prin tehnica de super-rezoluție, pornind de la o poză LR  $I^{LR}$  obținută din imaginea de adevăr  $I^{HR}$ . Pentru antrenament,  $I^{LR}$  este obținută prin micșorarea imaginii  $I^{HR}$  cu un factor  $r$ , urmând a fi aplicat un filtru gausian pentru obținerea unui blur pe imaginea de intrare. Pentru o imagine cu  $C$  canale de color, vom descrie  $I^{LR}$  drept un tensor de forma  $W \times H \times C$ , respectiv  $rW \times rH \times C$  pentru valoarea estimată  $I^{SR}$ .

Scopul este de a antrena o rețea de generator  $G$  ce va avea ca intrare o imagine LR, ieșirea SR fiind ulterior comparată cu imaginea HR corespunzătoare ei. Pentru a obține acest lucru, se vor antrena parametrii generatorului  $\theta_G$ ,  $\theta_G = W_{1:L}; b_{1:L}$  unde  $W$  și  $b$  reprezintă ponderile instruibile și termenii de bias, iar  $L$  numărul de straturi neurale adânci. Pentru antrenarea unui set de imagini  $I_n^{HR}, n = 1, \dots, N$  și imaginilor corespunzătoare  $I_n^{LR}, n = 1, \dots, N$ , vom rezolva:

$$\hat{\theta}_G = \arg \min_{\theta_G} \frac{1}{N} \sum_{n=1}^N l^{SR}(G_{\theta_G}(I_n^{LR}), I_n^{HR}) \quad (5.1)$$

- $\hat{\theta}_G$ : ponderile estimate de către generator;
- $N$ : numărul de perechi de imagini LR/HR din setul de antrenare;
- $G_{\theta_G}(I_n^{LR})$ : rezultatul obținut prin trecerea imaginilor LR prin generator;
- $l^{SR}$ : funcția de cost aplicată rețelei de generare a imaginilor SR;

astfel modelul propus încercă minimizarea funcției de cost  $l^{SR}$  pentru obținerea unei estimare  $\hat{\theta}_G$  cât mai corecte.

Pentru acest experiment au fost păstrați pașii de obținere a intrărilor arhitecturii FRVSR, însă rețea era proprietatea propriu-zisă pentru estimarea de imagini SR a fost înlocuită cu generatorul arhitecturii SRGAN.

Ca și concept de baza, generatorul folosește o rețea neurală reziduală cu modificări atât în structura blocurilor reziduale, cât și în etapa de upsampling, porțiune ce se ocupă de reconstrucția imaginii SR din informațiile imaginii LR și reziduu.

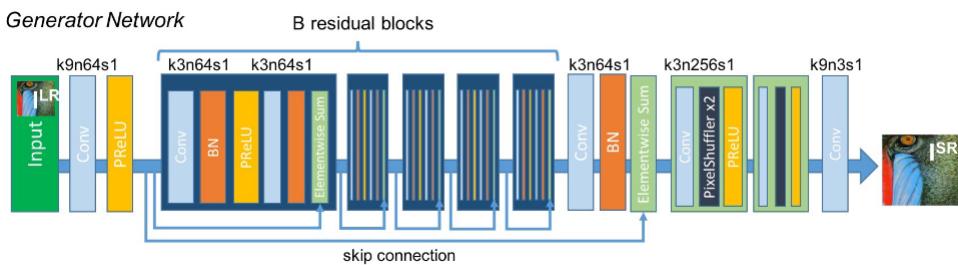


Figura 5.1: Arhitectura rețelei de generator din SRGAN. Imaginea  $I^{LR}$  este trecută printr-un strat convolutional de intrare pentru obținerea numărului de canale corespunzător zonei reziduale. Etapa de învățare a unui reziduu folosind un număr de  $B$  blocuri reziduale și conexiuni de tip "skip connection" pentru fiecare bloc în parte. Obținerea imaginii  $I^{SR}$  este realizată prin etapa de upsampling, prin folosirea a două straturi de convecție "sub-pixel" aplicată pe imaginea  $I^{LR}$  la care se adaugă informația reziduală învățată.

Rezultatele numerice determinate prin intermediul metricilor de performanță, prezentate în secțiunea 3.3, obținute în urma acestui antrenament pot fi observate în tabelul 5.2.

Antrenamentul folosind această arhitectură a fost realizat prin intermediul următorilor hiper-parametri:

- **Batch size:** 4
- **Rată de învățare:** 0.0001 - 0.00001

Arhitectură	PSNR	SSIM
FRVSR	22.96	0.79
FRVSR_IP	17.67	0.48
FRVSR_IP + SRGAN	17.12	0.47

Tabela 5.2: Comparația între rezultatele obținute de arhitectura FRVSR[14], implementare proprie și utilizarea arhitecturi generatorului din SRGAN[10].

- **Functii de cost:** L1, Mean Squared Error
- **Optimizare:** Algoritmul Adam
- **Numărul de epoci:** 100

### 5.1.2 Arhitectura ESRGAN

Propunerea de arhitectură oferită de [20], ce aduce schimbari asupra arhitecturii [10], introduce două modificări în structura rețelei de generator:

- **Eliminarea straturilor de batch norm.** Introducerea acestora în componentă straturilor reziduale, mărește performanța antrenării și reduce complexitatea computațională pentru probleme orientate în minimizarea valo- rii PSNR (ex. problemele de SR, VSR și cea de reducere ca cantității de blur din imagini). Folosirea acestor straturi în arhitecturile adânci au un efect negativ prin apariția ocazională a artefactelor, ce au efecte negative asupra performanței antrenamentului. Pentru problemele de SR, eliminarea straturilor batch norm oferă îmbunătățiri pentru pasul de generalizare, reduce complexitatea computațională și memoria folosită.
- **Introducerea blocurilor RRDB.** Spre deosebire de SRGAN, blocurile reziduale simple sunt înlocuite cu blocuri RRDB (Residual-in-Residual Dense Block) ce oferă adâncime și o structură mult mai complexă, benefice pentru o în- vătare mai bună și imagini rezultate calitative.

Adițional, se introduc mici tehnici de îmbunătățire a arhitecturilor adânci: scalarea reziduală ce reprezintă normalizarea valorilor reziduale prin înmulțirea acestora cu o constantă între (0, 1) și inițializarea arhitecturii cu ponderi foarte mici.

Pentru acest experiment pașii de obținere a datelor de intrare au fost păstrați, în schimb arhitectura de estimare a cadrelor SR a fost înlocuită cu generatorul arhitecturii ESRGAN.

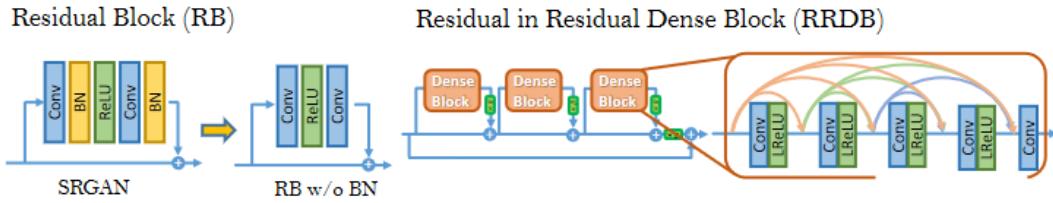


Figura 5.2: Arhitectura rețelei de generator din ESRGAN. (Stânga) Pasul de eliminare a straturilor batch norm din componența blocurilor reziduale propuse de [20]. (Dreapta) Introducerea blocurilor RRDB în arhitectura propusă, folosirea conexiunilor "skip connection" și parametrului  $\beta$  ce reprezintă factorul de scalare al blocurilor reziduale.

Rezultatele numerice calculate prin intermediul metricilor de performanță PSNR și SSIM (vezi secțiunea 4.2.11) în urma acestui antrenament fiind observate în tabelul 5.3, antrenarea acestei rețele fiind realizată pe o perioadă mai lungă fără a oferi rezultate.

Arhitectură	PSNR	SSIM
FRVSR	22.96	0.79
FRVSR_IP	17.67	0.48
FRVSR_IP + ESRGAN	17.34	0.47

Tabela 5.3: Comparația între rezultatele obținute de arhitectura FRVSR[14], implementare proprie și utilizarea arhitecturii generatorului din ESRGAN[20].

Antrenamentul folosind această arhitectură a fost realizat prin intermediul următorilor hiperparametri:

- **Batch size:** 4
- **Rată de învățare:** 0.0001 - 0.00001
- **Funcții de cost:** L1, Mean Squared Error
- **Optimizare:** Algoritmul Adam
- **Numărul de epoci:** 100

### 5.1.3 Arhitectura RDNSR

Marea majoritate a rețelelor neurale adânci pentru SR, ce folosesc la bază straturi neurale de conoluție, nu exploatează potențialul oferit de informațiile

provenite din imaginile de intrare LR, rezultând în performanțe slabe. Ca soluționare a acestui aspect, [21] propune o rețea reziduală adâncă și un model de bloc rezidual adânc (RDB) pentru extragerea de caracteristici locale abundente, pas realizat prin intermediul straturilor de convecție.

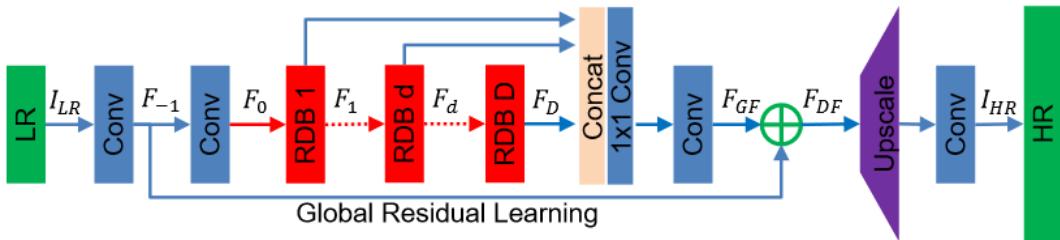


Figura 5.3: Arhitectura rețelei neurale reziduale propusă (RDNSR).

Blocurile RDB permit în plus conexiuni directe de la starea blocului RDB precedent către restul blocurilor, formând astfel un mecanism de memorie contiguă. Caracteristicile locale sunt fuzionate în blocurile RDB pentru o învățare mai eficientă oferind stabilizare în timpul etapei de antrenare. În urma obținerii caracteristicilor din conexiunile dense locale, se vor determina caracteristici globale prin fuzionarea informațiilor obținute ierarhic.

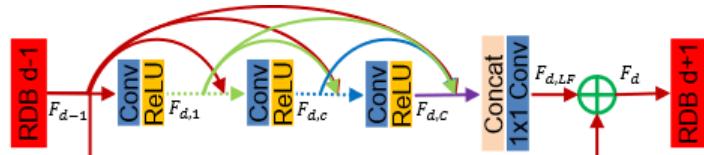


Figura 5.4: Blocurile reziduale dense (RDB) propuse.

Mecanismul de memorie contiguă este realizat prin trecerea informației obținute din blocul RDB precedent, către toate straturile blocului RDB curent. Considerăm  $F_{d-1}$  și  $F_d$  intrarea respectiv ieșirea unui bloc  $d$  din arhitectura propusă, cu hărțile de caracteristici  $G_0$ . Ieșirea corespunzătoarea unui strat convolutional  $c$  al unui bloc  $d$  va fi formulată astfel:

$$F_{d,c} = \alpha(W_{d,c}[F_{d-1}, F_{d,1}, \dots, F_{d,c-1}]) \quad (5.2)$$

unde  $\alpha$  reprezintă funcția de activate ReLU,  $W_{d,c}$  ponderile unui strat de convecție  $c$ , iar  $[F_{d-1}, F_{d,1}, \dots, F_{d,c-1}]$  operația de concatenare a tuturor hărților de caracteristici produse de blocul RDB anterior  $d - 1$ . Observăm astfel că informația ieșirilor precedente ale blocurilor RDB vor avea legături cu straturile curente, realizând astfel conexiuni dense locale.

Inspirat din arhitectura SRDenseNet, autorii acestei articole aduc 3 modificări asupra soluției oferite de [19]:

- Introducerea unui nou tip de bloc rezidual. Beneficiile aduse de acest de bloc constau în introducerea mecanismului de memorie continuă și obținerea de caracteristici locale, mărind astfel performanța și stabilitatea antrenamentului.
- Lipsa conexiunilor dense dintre blocurile RDB. În schimb se utilizează fuziunea de caracteristici globale și învățarea caracteristicilor globale, aducând astfel performante semnificative în pasul de generare de imagini SR.
- Înlocuirea funcție de cost  $L_2$  cu funcția de cost  $L_1$  oferă rezultate mai plăcute vizual.

Rezultatele numerice calculate prin intermediul metricilor de performanță PSNR și SSIM (vezi secțiunea 4.2.11) în urma acestui antrenament pot fi observate în tabelul 5.4, rezultate vizuale ale acestui videoclip beneficiază de o consistență temporară mai bună.

Arhitectură	PSNR	SSIM
FRVSR	22.96	0.79
FRVSR_IP	17.67	0.48
FRVSR_IP + RDB	17.79	0.48

Tabela 5.4: Comparația între rezultatele obținute de arhitectura FRVSR[14], implementare proprie și utilizarea arhitecturii RDB[21].

Antrenamentul folosind această arhitectură a fost realizat prin intermediul următorilor hiper-parametri:

- **Batch size:** 8
- **Rată de învățare:** 0.0001 - 0.00001
- **Funcții de cost:** L1, Mean Squared Error
- **Optimizare:** Algoritmul Adam
- **Numărul de epoci:** 100

## 5.2 Utilizarea unui model pre-antrenat pentru determinarea fluxului optic

Pentru îmbunătățirea pasului de determinare a fluxului optic, se introduce arhitectura pre-antrenate PWC-Net[17] ce va înlocui modelul arhitectură curentă (FRVSR). Rețeaua PWC-Net este una dintre cele mai capabile și eficiente modele pre-antrenate pentru determinarea fluxului optic, folosit des de dezvoltatorii ce abordează probleme de interpolare a cadrelor. Acest model a fost antrenat pe setul de date sintetic "FlyingChairs" dezvoltat de [4].

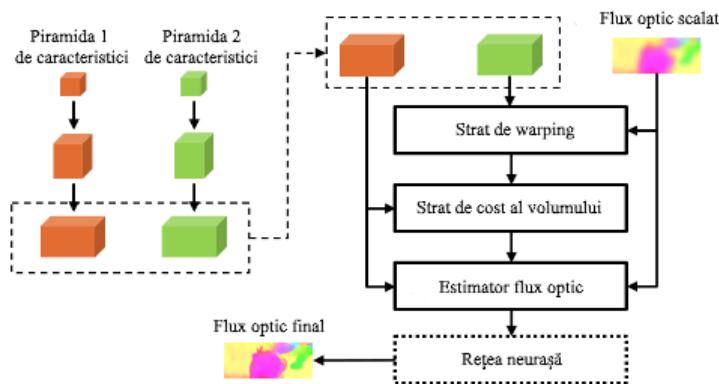


Figura 5.5: Arhitectura rețelei PWC-Net de estimare a fluxului optic.

Intuitiv, prin utilizarea acestui model, ne așteptăm ca performanța rețelei și calitatea cadrelor SR generate să fie îmbunătățite. Contradictoriu însă, înlocuirea rețelei de flux optic curente cu rețeaua pre-antrenată a dus la obținerea unor performanțe mai slabe, estimările cadrelor fiind de o calitate mai redusă spre deosebire de rezultatele experimentelor anterioare. În urma pasului de antrenare a acestei rețele, s-a sesizat de asemenea o diferență pentru valorile numerice rezultate. Posibilele motive ce au stat la baza obținerii acestei diferențe sunt:

- Odată cu progresul antrenamentului, estimarea fluxului optic se va îmbunătăți pentru a oferi rețelei de super-rezoluție date de intrare calitative, prin intermediul deplasării pixelilor din cadrul estimat precedent  $I_{t-1}^{est}$ .
- Rețea de super-rezoluție învață să ignore pasul de deplasare a imaginii precedente la imaginea curentă atunci când informația oferită de modelul de estimare a fluxului optic este irelevantă. Aceste cazuri vor fi detectate, iar rețeaua de super-rezoluție va învăța să redimensioneze imaginea LR curentă.

Concluzionăm faptul ca antrenarea modelului de determinare a fluxului optic trebuie făcută în paralel cu antrenarea rețelei de super-rezoluție, existând astfel o dependență între cele două. Prin folosirea unui model pre-antrenat pe un set de date conceput pentru problema de super-rezoluție, vor exista diferențe precum dimensiunea imaginilor utilizate în etapa de antrenarea acestor rețele (rețea folosită în FRVSR este antrenată pe imagini mici spre deosebire de imaginile oferite de setul de date "Flyingchairs").

### 5.3 Utilizarea funcției de cost Ping-Pong

Idem modelelor SRGAN și ESRGAN, TecoGAN este o arhitectură propusă de [2] provenită din familia rețelelor generativ adversariale. Autorii propun funcția de cost Ping-Pong (PP) ce oferă îmbunătățiri în consistența temporală pentru secvențele lungi de cadre consecutive. Unicitatea acestei funcții este datorată caracterului bidirectional, reprezentând astfel o inovație în rândul funcțiilor de cost folosite pentru problema de super-rezoluție aplicată videoclipuri.

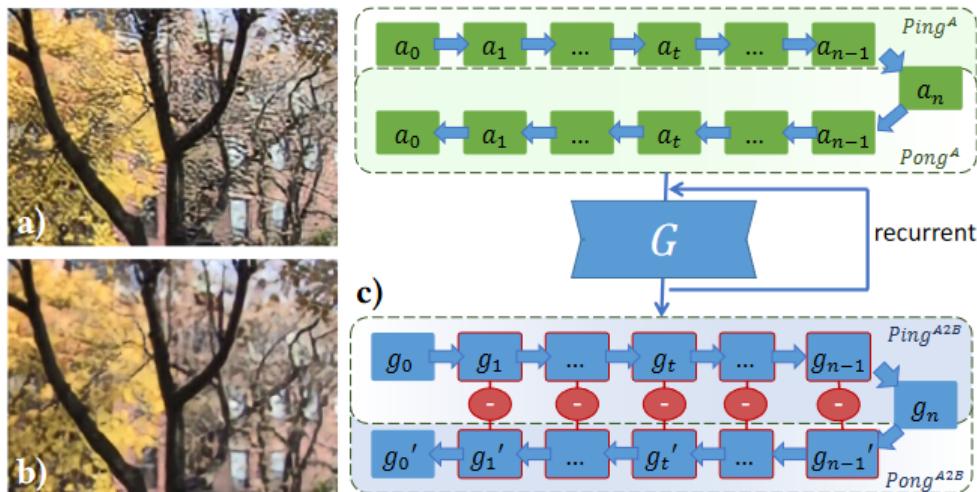


Figura 5.6: (a) Rezultatele obținute fără funcția de cost PP. Pentru secvențe mai lungi la pasul de antrenare, cadrele îndepărtate vor suferi prin apariția artefactelor de drifting. (b) Rezultatele obținute prin antrenarea cu funcția de cost PP. (c) Pentru o secvență de cadre construită simetric (secvență normală numită Ping, iar secvență inversă numită Pong), se va calcula distanța  $L_2$  dintre cadrele Ping  $g_t$  și cadrele Pong  $g'_t$ . PP reduce apariția artefactelor de drifting și îmbunătățește consistența temporară.

Pentru o secvență de intrare  $a$  cu  $n$  cadre consecutive, acestea se vor con-

catenate simetric obținând astfel secvența  $a_1, \dots, a_{n-1}, a_n, a_{n-1}, \dots, a_1$  (vezi figura 5.6). Prin antrenarea în această ordine, aproximarea imaginilor noi generate va fi realizată prin intermediul cadrele dispuse în ordinea succesiunii în timp,  $g_t = G(a_t, g_{t-1})$ , și totodată a cadrele dispuse într-o ordinea inversă a timpului,  $g^t = G(a_t, g'_{t-1})$ . Acest lucru constrângе algoritmul în a obține o similaritate între cadrele generate  $g_t$  și cadrele generat invers  $g'_t$ .

În contrast cu funcția de cost L1, utilizată des datorită performanțelor este-  
tice pentru problema de SR, PP folosește la baza norma  $L_2$  ce va evita favorizarea  
obținerii unui rezultat cât mai calitativ vizual, ci dorește constrângerea arhitectu-  
rii recurente de a favoriza informația temporară. Cadrele apropriate în timp, (ex.  
 $g_{n-1}, g'_{n-1}$  figura 5.6 (c)) vor avea rolul de a păstra consistența informației pe ter-  
men scurt, în timp ce cadrele depărtate, (ex.  $g_1$  împreună cu  $g'_1$ , vezi figura 5.6 (c))  
vor avea ca scop păstrarea consistenței pe termen lung, evitând astfel apariția ar-  
tefactelor de drifting din cadrele rezultat (fenomenul de drifting poate fi observat  
în figura 5.6 (a)).

Rezultatele numerice calculate prin intermediul metricilor de performanță PSNR  
și SSIM (vezi secțiunea 4.2.11) în urma acestui antrenament pot fi observate în  
tabelul 5.5, acest antrenament o oferit rezultate mai bune prin obținerea unor  
imaginii mai detaliate.

Arhitectură	PSNR	SSIM
FRVSR	22.96	0.79
FRVSR_IP	17.67	0.48
FRVSR_IP + RDB + PPLoss	18.03	0.49

Tabela 5.5: Comparația între rezultatele obținute de arhitectura FRVSR[14], im-  
plementare proprie și utilizarea arhitecturii RDB[21] cu adăugarea funcției de cost  
PP.

Antrenamentul folosind această arhitectură a fost realizat prin intermediul ur-  
mătorilor hiper-parametri:

- **Batch size:** 4
- **Rată de învățare:** 0.0001 - 0.00001
- **Funcții de cost:** L1, Mean Squared Error, PP Loss
- **Optimizare:** Algoritmul Adam
- **Numărul de epoci:** 100

## 5.4 Modificarea arhitecturii FRVSR

### 5.4.1 Descrierea arhitecturii

O mare parte e experimentelor au avut ca punct de plecare structura modelului propus de Sajjadi et al.[14], acesta reprezentând de asemenea o abordare recurrentă pentru soluționarea problemelor VSR.

Unele dintre cele mai importante contribuții aduse de către autori sunt:

- Folosirea cadrelor anterior generate încurajează algoritmul în păstrarea consistenței temporare, reducând costul computațional.
- Propagarea informațiilor pe o distanță mai îndelungată de timp fără a aduce o creștere computațională în timpul antrenării.

Componentele antrenabile (figura 5.7, cele de culoare roșie) includ rețeaua de estimare a fluxului optic FNet și rețeaua SRNet de generare a cadrelor prin super-rezoluție. Pentru obținerea unui cadru cu o claritate ridicată  $I_t^{est}$ , modelul se folosește de cadrul curentă  $I_t^{LR}$ , cadrul anterioară  $I_{t-1}^{LR}$ , respectiv cadrul generat anterior  $I_{t-1}^{est}$ .

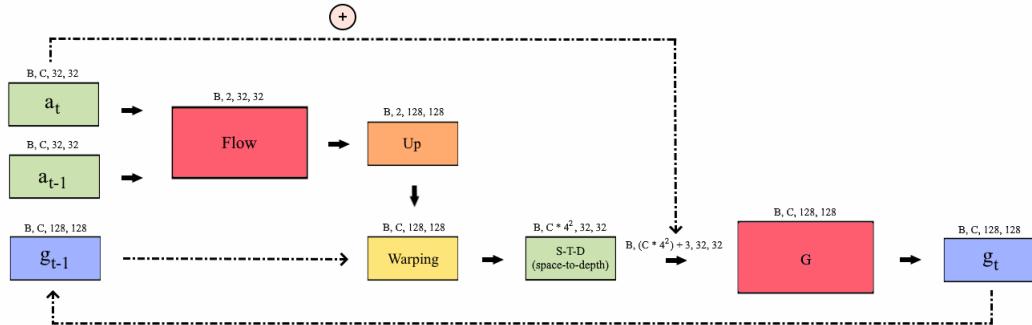


Figura 5.7: Pași de formare a intrărilor rețelei SRNet (G).

Primul pas în procurarea cadrului dorit este reprezentat de folosire arhitecturii FNet pentru estimarea fluxului dintre cadrele  $I_{t-1}^{LR}$  și  $I_t^{LR}$ , rezultatul fiind o harta de caracteristici a fluxului cu valori în intervalul  $[-1, 1]$ :

$$F^{LR} = FNet(I_{t-1}^{LR}, I_t^{LR}) \in [-1, 1]^{H \times W \times 2} \quad (5.3)$$

pentru asignarea pozițiilor pixelilor din imaginea  $I_{t-1}^{LR}$  către o locație din imaginea  $I_t^{LR}$ .

Prin tratarea drept o imagine a rezultatului din pasul anterior, se aplică o operație de redimensionare cu factor  $s$  pentru modificarea hărții de caracteristici de la dimensiunea cadrelor LR, la dimensiunea asemănătoare cadrelor HR:

$$F^{LR} = UP(F^{LR}) \in [-1, 1]^{sH \times sW \times 2} \quad (5.4)$$

iar prin folosirea hărții de flux HR obținute, pixelii din imaginea generată anterior vor fi deplasati către imaginea curentă:

$$\tilde{I}_{t-1}^{est} = WP(\tilde{I}_{t-1}^{est}, F^{HR}) \quad (5.5)$$

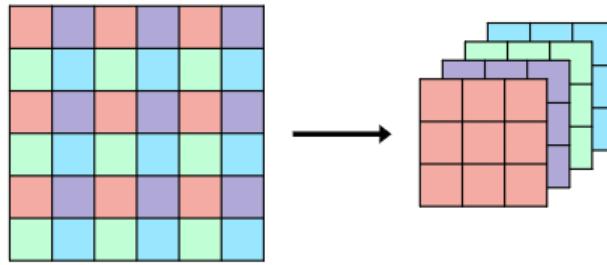


Figura 5.8: Operația de transformare "space-to-depth". Extragerea de informație din imaginea HR și mutarea acesteia pe dimensiunea de canal, obținând astfel imaginea LR.

Etapa finală pentru realizarea unui cadru estimat  $I_t^{est}$  este reprezentat de treccerea prin arhitectura cadrelor  $I_t^{LR} \oplus S_s(\tilde{I}_t^{est})$  unde,  $\oplus$  reprezintă operația de concatenare pe dimensiunea canalelor. Observăm astfel o problemă datorată diferențelor de dimensiuni între  $I_t^{LR}$  și  $\tilde{I}_t^{est}$ . Operației de extragere a informației din dimensiunea imaginii în adâncime<sup>1</sup> reprezintă o soluție pentru această problemă, prin deplasarea cu un factor  $s$  a informației din dimensiunile pozei pe dimensiunea canalelor:

$$S_s : [0, 1]^{sH \times sW \times C} \rightarrow [0, 1]^{H \times W \times s^2 C} \quad (5.6)$$

Operația poate fi formulată astfel:

$$S_s(I)_{i,j,k} = I_{si+k\%s, sj+(k/s)\%s, k/s} \quad (5.7)$$

unde  $i, j, k$  reprezintă indexi ce pornesc de la valoarea 0, % operația de modul și \ operația de împărțire la un număr întreg.

---

<sup>1</sup>Operația de "space-to-depth"

Pentru antrenare modelelor menționate, autorii propun folosirea a două funcții de cost,  $L_{SR}$  dedicată pentru rețeaua SRNet, respectiv  $L_{flow}$  pentru rețeaua de flux optic. Deoarece setul de date nu dispune de imagini de adevăr pentru calcularea diferenței dintre fluxul estimat și flux real, vom determina în schimb valoarea de eroare dintre rezultatul ecuației (5.5) și cadrul curent  $I_t^{LR}$ . Pentru determinarea valorii de eroare a rețelei SRNet, avem:

$$L_{SR} = \|I_t^{est} - I_t^{HR}\|_2^2 \quad (5.8)$$

respectiv, asemănător pentru calcularea valorii erorii rețelei FNet:

$$L_{SR} = \|WP(I_t^{est} - 1, F^{LR}) - I_t^{LR}\|_2^2 \quad (5.9)$$

Pasul de propagarea înapoi va consta în propagarea erorii totale pentru ambele arhitecturi, valoare obținută din  $L = L_{SR} + L_{flow}$ .

#### 5.4.2 Modificări aduse structurii modelului

Așa cum s-a demonstrat de-a lungul timpului prin nenumăratele experimente, o direcția bună o reprezintă modificarea dimensiunii unei rețele. Astfel primele experimente au constat în mărirea adâncimii modelului SRNet, respectiv ale dimensiunilor straturilor de conoluție.

În arhitectura referință, autorii folosesc un număr de 16 blocuri reziduale, structura unui bloc fiind format dintr-un strat de conoluție de intrare cu 64 de canale de intrare/ieșire, aplicarea funcției ReLU și trecerea rezultatului obținut printr-un strat final de conoluție asemănător cu cel de intrare. Ieșirea blocului rezidual este formată din adunarea reziduului obținut la cadrul inițial de intrare.

În cadrul acestui experiment au fost propuse modificări asupra numărului de blocuri reziduale, din 16 în 20, respectiv dimensiunea straturilor de conoluție, din 64 în 128 de canale. Atât aspectul vizual cât și valoarea metricilor obținute din urma acestor modificări au fost cu puțin mai bune spre deosebire de rezultate obținute de către modelul de bază.

Arhitectură	PSNR	SSIM
FRVSR	22.96	0.79
FRVSR_IP	17.67	0.48
FRVSR_IP_20-64	17.87	0.49
FRVSR_IP_20-128	17.90	0.49

Tabela 5.6: Comparația între rezultatele obținute de [14], implementare proprie 16-64 (numărul de blocuri și numărul de canale din stratul de conoluție), implementarea proprie 20-64 și 20-128.

Antrenamentul folosind această arhitecturi (descrise în ordinea 16-64, 20-64, 20-128) a fost realizat prin intermediul următorilor hiper-parametri:

- **Batch size:** 16
- **Rată de învățare:** 0.0001 - 0.000001
- **Funcții de cost:** L1, Mean Squared Error
- **Optimizare:** Algoritmul Adam
- **Numărul de epoci:** 100

Experimentul următor a constat în adăugarea unui nou strat de normalizare în structura blocului rezidual. Pornind de la structura blocului din figura 5.9 , sugestia făcută de [20] constă în eliminarea straturilor batch norm. Propunerea proprie este de a introduce în schimb un strat spectral norm (vezi secțiunea 4.2.4.9) după stratul de intrare , iar stratul batch norm final să fie eliminat. Motivul eliminării stratului din finalul blocului rezidual este de a păstra valori asemănătoare cu cele obținute din ultimul strat de convoluție al blocului rezidual din modelul FRVSR.

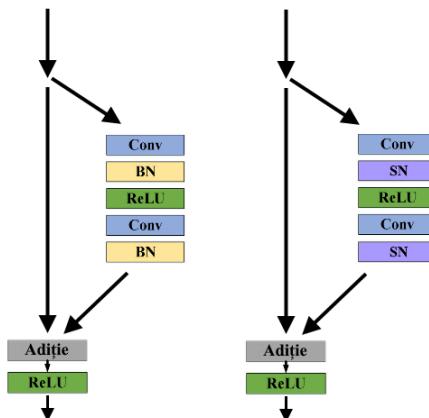


Figura 5.9: (Stânga) Structura standardă a unui bloc rezidual. (Dreapta) Structura folosită în cadrul acestui experiment.

Rezultatele obținute în urma acestui antrenament au fost semnificative, straturile adăugate jucând un rol important în obținerea unor cadre mai calitate și de reducere a timpului necesar pentru antrenare. Acest experiment a fost desfășurat pentru cele trei arhitecturi menționate anterior (arhitectura de bază și cele două variante mai adânci) obținând valorile afișate în tabelul 5.7.

Antrenamentul folosind această arhitecturi (descrise în ordinea 16-64, 20-64, 20-128) a fost realizat prin intermediul hiper-parametrilor folosiți la experimentul precedent.

Arhitectură	PSNR	SSIM
FRVSR	22.96	0.79
FRVSR_IP_16-64 + SN	17.84	0.47
FRVSR_IP_20-64 + SN	17.96	0.49
FRVSR_IP_20-128 + SN	18.02	0.49

Tabela 5.7: Comparația între rezultatele obținute de [14], implementare proprie 16-64 (numărul de blocuri și numărul de canale din stratul de convoluție), implementarea 16-128 și și 20-128.

Ultimul experiment a fost desfășurat pe baza implementării de bază (FRVSR\_IP\_16-64) prin adăugarea de noi funcții de cost la pasul de propagare înapoi a valorii de erori pentru cele două rețetele.

Primul experiment a constat în adăugarea funcției de cost PP sugerate anterior (secțiunea 5.3), obținând un videoclip cu o consistență temporară mult mai bună spre deosebire de arhitectura de bază.

Pentru al doilea experiment a fost introdus o nouă funcție de cost ce primește o atenție deosebită din partea cercetătorilor, propusă de Johnson et al. [6] și numită funcția de cost perceptuală (perceptual loss). Această funcție folosește formula de calcul a distanței L2, însă imaginea reală și cea estimată vor fi înlocuite cu hărți de caracteristici obținute în urma trecerii cadrului real, respectiv cadrului estimat de rețeaua de super-rezoluție prin rețeaua VGG16.

VGG16 este o rețea neurală conluțională clasică propusă de K. Simonyan și A. Zisserman [16] folosită în obținerea funcției de cost perceptuale. Prin trecerea pe rând a cadrelor prin modelul VGG16 pre-antrenat, obținem hărți de caracteristici pentru diferite nivele de adâncime, ce reprezintă de fapt diverse trăsături abstracte. Compararea acestor hărți pentru cele două imagini introduse sugerează de fapt, compararea trăsăturilor interpretate de rețeaua VGG16.

Funcția de cost perceptuală este calculată prin media valorilor obținute în urma măsurării distanței L2 dintre hărțile de caracteristici pentru diferite nivele din arhitectura, hărți obținute prin trecerea prin model a cadrului estimat  $I_t^{est}$  și a imaginii reale  $I_t^{HR}$ .

Acest experiment a constat în introducerea celor 2 funcții de eroare propuse (PP și perceptuală) la funcția de cost de bază (formată din funcții de cost ale rețelelor SRNet și FNet). Rezultatul obținute în urma antrenamentului (cu modelul FRVSR\_IP\_20-64 + SN și adăugarea blocurilor reziduale) s-a situat în topul clasamentului, videoclipul generat prin această metodă fiind cu o calitate vizuală și o consistență temporară mai bună.

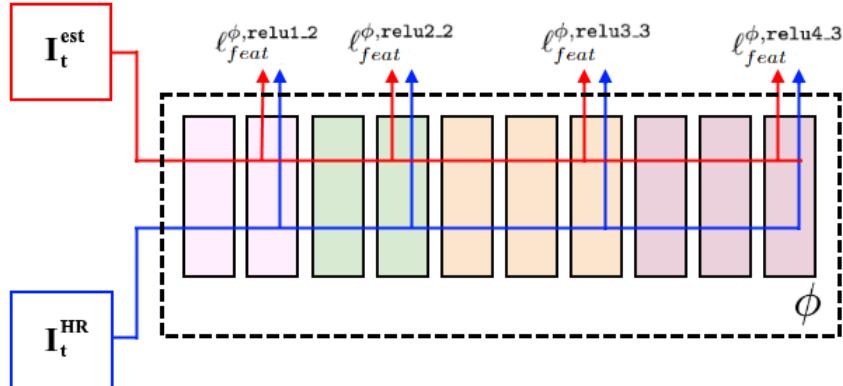


Figura 5.10: Treceea cadrelor prin VGG16. Obținerea valorilor de eroare pentru diferite nivele de adâncime. Valoarea finală va fi reprezentată de media tuturor valorilor calculate. Hărțile de caracteristici vor fi obținute din startul anterior pașului de aplicarea a funcției de cost ReLU, respectiv cel de modificare a adâncimi.

## 5.5 Soluția finală

În obținerea soluției finale au fost utilizate următoarele componente:

- Rețeaua de super rezoluție alesă este FRVSR\_IP\_20\_64. Alegerea este justificată de mărimea rețelei și rezultatele obținute vizual dar și ale valorilor pentru metricile PSNR și SSIM. Modele mai adânci au oferit rezultate la o diferență foarte mică, în schimb dificultatea antrenării a fost mult mai ridicată. Acest model însă oferă rezultate de top cu o antrenare mai ușoară.
- Folosirea funcțiilor de cost PP și perceptuală aduce beneficii precum calitate mai bună a videoclipurilor și păstrarea consistenței temporare.
- Folosirea blocului rezidual propus ce oferă detalii și o antrenare mult mai rapidă a rețelei.

Antrenamentul folosind această arhitectură a fost realizat prin intermediul următorilor hiper-parametri:

- **Batch size:** 8
- **Rată de învățare:** 0.0001 - 0.000001
- **Funcții de cost:** L1, Mean Squared Error, PP Loss, Perceptual Loss
- **Optimizare:** Algoritmul Adam
- **Numărul de epoci:** 100

Arhitectură	PSNR	SSIM
FRVSR + PWC-Net	-	-
FRVSR + SRGAN	17.12	0.47
FRVSR + ESRGAN	17.34	0.47
FRVSR_IP	17.67	0.48
FRVSR + RDB	17.79	0.48
FRVSR_IP_16-64 + SN	17.84	0.49
FRVSR_IP_20-64	17.87	0.49
FRVSR_IP_20-128	17.90	0.49
FRVSR_IP_20-64 + SN	17.96	0.49
FRVSR_IP_20-128 + SN	18.02	0.49
FRVSR + RDB + PP Loss	18.03	0.49
<b>FRVSR_IP_20-128 + Bloc + PP + PLoss</b>	18.17	0.50
<b>FRVSR</b>	22.96	0.79

Tabela 5.8: Compararea tuturor rezultatelor experimentale pentru metricile PSNR și SSIM.

# Capitolul 6

## Tehnologii utilizate

Pentru realizarea acestui proiect au fost îmbinate diverse tipuri de tehnologii, având ca scop obținerea unui cod coerent și corect, comparativ cu alte abordări de rezolvare a diferitelor probleme de machine learning. În continuare vom prezenta tehnologiile folosite cât și beneficiile pe care le aduc acestea.

### 6.1 Limbajul de programare Python

Python este un limbaj de programare atractiv ce aduce programatorului beneficii precum rapiditate în dezvoltarea unei aplicații, folosirea pentru programe de tip scripting, fiind considerat un limbaj “legătură” pentru conectarea diferitelor componente hardware. Câteva dintre aspectele importante ce ies în evidență atunci când se dorește implementarea unui soft în acest limbaj sunt:

- **Calitatea softului** - pentru mulți programatori, Python se focusează pe aspecte precum ușurința de citire, coerența și calitatea pe care o oferă în special pentru softurile de scripting. Uniformitatea acestui limbaj oferă o înțelegere mult mai ușoară pentru un programator care a lucrat într-un limbaj diferit. Aceasta dispune de o sintaxă simplificată, considerată uneori apropiată cu limbajul natural;
- **Productivitate** - Python menține un cod de dimensiuni reduse spre deosebire de alte limbi de programare (ex: C++, Java). Un cod mai scurt uneori determină un efort mai mic când vine vorba de probleme des întâlnite precum timpul de scriere, debug sau mentenanță;
- **Suport de biblioteci** - Python este însotit de o colecție largă de funcționalități direct integrate odată cu instalarea acestuia, cunoscute și sub denumirea de librării standard. De asemenea numărul de librării se extinde într-o

colecție vastă, ce oferă soluții pentru aplicații de tip web, numerice, dezvoltarea de jocuri și multe altele;

- **Integrarea componentelor** - script-urile scrise în Python comunică ușor și eficient cu alte parți ale unei aplicații, utilizându-se o varietate de mecanisme integrate. Codul Python poate invoca librării scrise în limbaje precum C sau C++ sau poate fi apelate de programe dezvoltate în aceste două limbaje. Totodată Python poate fi integrat în componente Java sau .Net, asigură o interacțiune cu dispozitive serial-port sau o “legătura” cu internetul prin intermediul interfețelor precum SOAP sau XML-RPS.

## 6.2 PyTorch

PyTorch este o bibliotecă ce oferă programelor scrise în Python, facilități pentru crearea și dezvoltarea proiectelor de tip machine learning. Fiind una dintre platformele utilizate în cercetare din acest domeniu, oferă flexibilitate și viteză maximă. Trăsătura de bază caracterizată de acestă bibliotecă este calculul optimizat de tensori ce suportă accelerarea prin intermediul unității de procesare grafică. PyTorch oferă un suport grafic de calcul dinamic (dynamic computation graphs) ceea ce permite schimbarea comportamentului arhitecturii în timpul execuției. Comparativ cu alte platforme precum Tensorflow, aceasta folosește grafice de calcul static (static computation graphs) ceea ce înseamnă întâi o definire, ulterior realizând pasul de executare a modelului dorit.

### 6.2.1 TorchVision

TorchVision este o componentă a platformei PyTorch, cu funcționalități utile pentru probleme de viziune printr-un calculator (computer vision). Printre funcționalitățile importante întâlnim seturi de date, modele de arhitecturi implementate și antrenate, operații de transformare.

### 6.2.2 Module și clase abstracte

PyTorch este format din trei module principale:

- **torch.autograd** - un sistem de diferențiere automată care stă la baza antrenării rețelelor neurale;
- **torch.optim** - un pachet ce conține implementarea diferitilor algoritmi de optimizare, totodată oferind utilizatorului posibilitatea unei integrări ușoare a unor noi algoritmi prin extindere a algoritmilor deja existenți;

- **torch.nn** - pachet de bază ce ajută la crearea și antrenarea unei rețele neurale, conținând module importante precum Parameters, Containers, funcții de activare, diferite tipuri de straturi neurale, funcții de cost cât și alte funcții utile.

Un efort substanțial în rezolvarea unei probleme de machine learning este direcționat în pregătirea setului de date. Calitățile acestei biblioteci sunt expuse atunci când lucrăm cu seturile de date, oferind soluții intuitive pentru procesul de încărcare și transformare a acestora. Acest lucru se realizează prin intermediul claselor abstrakte:

- **Dataset** - clasă abstractă ce se regăsește în pachetul `torch.utils.data`, stochează probele (samples) cât și etichetele (labels) corespunzătoare;
- **Dataloader** - clasă abstractă ce ajută la iterarea clasei Dataset pentru un acces simplu al datelor.

## 6.3 Biblioteci

### 6.3.1 NumPy

NumPy (Numerical Python) este unul dintre pachetele fundamentale pentru calcul științific în Python. Acesta conține funcționalități precum calculul multidimensional al vectorilor, operații din algebra liniară, operații de transformare și generare aleatorie de numere. Funcțiile oferite beneficiază de implementări foarte bine optimizate din punct de vedere al timpului și costului de memorie. Acest pachet stă la baza implementării aplicațiilor în domenii precum știința datelor (data science) și machine learning.

Câteva exemple de funcționalități folosite în implementarea lucrării sunt: liste de tip numpy ce beneficiază de o gamă largă de funcții optimizate precum sortări de liste, modificarea dimensionalității unei liste multidimensionale, conversie tipurilor de date, produs de matrici, rotunjirea valorilor, ridicare la putere, media aritmetică, logaritm zecimal.

### 6.3.2 Matplotlib

Matplotlib este o una dintre cele mai des folosite biblioteci pentru creare de statistică prin intermediul unor grafice animate și interactive, implementată în limbajul Python și considerată o extensie pentru NumPy. Scopul acestei biblioteci este de a replica funcționalitățile oferite de limbajul MATLAB ce oferă capabilități asemănătoare de creare de statistică grafice.

În decursul implementărilor au fost creare graficelor și a diverselor statistică pentru a valida datele de antrenare, funcționalitatea codului implementat, dar și a rezultatelor obținute în decursul antrenării, validări și etapei de testare.

### 6.3.3 OpenCV

OpenCV (Open Source Computer Vision Library) este o bibliotecă open-source folosită în implementarea aplicațiilor de computer vision și machine learning. Aceasta a fost concepută pentru a oferi o infrastructură stabilă, prin implementarea a peste 2500 de algoritmi optimizați, ce includ atât abordări clasice cât și algoritmi contemporani "la rang de artă" (state-of-the-art) pentru ramura de computer vision și machine learning. Câteva exemple de domenii ce beneficiază de pe urma acestei biblioteci sunt: detectarea și recunoașterea facială, identificarea obiectelor, monitorizarea obiectelor în mișcare, extragerea modelelor 3D, identificarea imaginilor similare dintr-un set de date, monitorizarea oculară etc.

OpenCV este extinsă pentru o gamă largă de limbaje de programare precum Python, C++, MATLAB și Java, oferind totodată suport pentru sistemele de operare precum Linux, Windows, Android, Mac OS și iOS.

### 6.3.4 Flow-Vis

Flow-Vis (Flow Visualisation) este o bibliotecă open-source ce permite vizualizarea matricei de transport (flow transport matrix) prin intermediul unei imagini RGB. Acest lucru se face prin convertirea unei matrici bidimensionale, ce conține direcțiile și viteza de deplasare a pixelilor dintr-o imagine  $a$  la o imagine  $b$ , într-o imagine RGB unde culoarea pixelilor reprezintă diferite direcții de deplasare, iar intensitatea culorii viteza acestei deplasări.



Figura 6.1: Mișcarea rezultată prin inferență a două cadre într-o arhitectură de flow. Variați de direcție în funcție de culoarea obținută.

### **6.3.5 youtube-dl și FFmpeg**

Prin intermediul bibliotecii open-source youtube-dl se obțin videoclipuri prin descărcarea acestora de pe platforma Youtube, cât și câteva alte platforme precum. Această bibliotecă este implementată în limbajul Python, cu versiuni stabile pentru Python 2.6, 2.7 și 3.2. Prin intermediul acestei biblioteci au fost obținute videoclipurile necesare formării setului de date.

FFmpeg este o bibliotecă ce se ocupă cu prelucrarea fișierelor multimedia, precum fișiere video sau audio. Obținerea de cadre din videoclipuri a fost realizată prin intermediul acestei biblioteci. Aceste imagini au fost ulterior prelucrate și folosite în etapa de antrenare, validare și testarea a modelelor implementate.

### **6.3.6 CUDA**

Nvidia este una dintre cele mai mari companii de tehnologii, ce se ocupă cu proiectarea unităților de procesare grafică (GPU). CUDA (Compute Unified Device Architecture), creată ulterior de Nvidia, reprezintă o platformă software care împreună cu aceste unități grafice, ușurează dezvoltarea aplicațiilor prin folosirea accelerării computaționale. Beneficiul adus de utilizarea unităților grafice (GPU) în locul unităților centrate (CPU) este puterea de computație oferită de acestea. În timp ce CPU se ocupă de computații generale, GPU oferă o rapiditate în execuțarea calculelor, prin intermediul calcului paralel.

Calculul paralel reprezintă împărțirea unui calcul complex în sarcini mici independente ce pot fi executate simultan. Rezultatele oferite de aceste computații sunt ulterior recombinante, sincronizate, împreună formând un rezultat pentru calculul complex oferit.

Numărul de core-uri se ocupă de executarea unui calcul, astfel numărul de sarcini este în strânsă legătură cu numărul de core-uri oferit de unitatea grafică folosită. Dacă în timp ce o unitate centrală de procesare poate ajunge la un număr de 4, 8 sau chiar 16 core-uri, unitatea grafică de procesare poate avea un potențial de câteva mii de core-uri (ex: cel mai performant GPU actual este Nvidia GeForce RTX 3090 ce conține cca. 10500 CUDA core-uri).

## **Capitolul 7**

### **Ghid aplicației utilizator**

Pentru a oferi utilizatorului o interacțiune cu forma finală a modelului antrenat propus, a fost necesară implementarea unei aplicații ce dispune de o interfață grafică intuitivă și ușor de accesat, astfel încât utilizatorul să poată încărca un videoclip dorit și să obțină rezultatul din urma procesării acestuia de către modelul de super-rezoluție.

Pașii ce trebuie urmăriți pentru obținerea rezultatului sunt:

- Încărcarea videoclipului de către utilizator, formatul suportat de către aplicație fiind "mp4" sau "avi". Acest pas are ca rol transformarea videoclipului în cadrele necesare procesului de obținere a rezultatului.
- Încărcarea de către utilizator a fișierului ce conține modelul pre-antrenat, formatul suportat de către aplicație fiind fișierele cu extensia "ckpt".
- Selectarea de către utilizator a directorului de ieșire, în acest director urmează a fi salvat rezultatul obținut reprezentat de un videoclip cu formatul "avi".
- Pornirea procesului de aplicare a tehnicii de super-rezoluție pe videoclipul selectat este realizat prin intermediul butonului "Start".

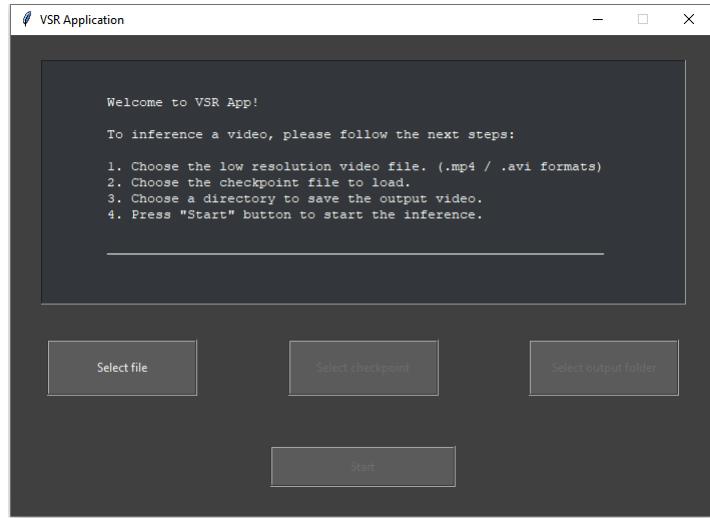


Figura 7.1: Ecranul principal al aplicației. Butoanele puse la dispoziția utilizatorului pentru încărcarea de videoclip, a fișierului ce conține modelul pre-antrenat, directorul de ieșire și butonul de pornire. Pașii menționati trebuie făcuți în ordinea descrisă.

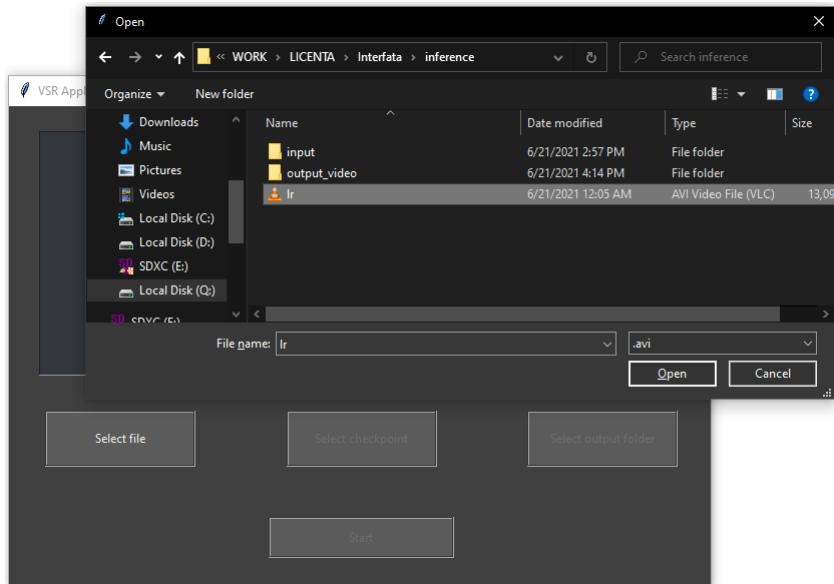


Figura 7.2: Pentru deschiderea ferestrei de selectare a unui videoclip se va utiliza butonul "Select file". Odată ce fereastra este deschisă, utilizatorul va selecta formatul fișierului și videoclip dorit, iar prin apăsarea butonului "Open" se va executa încărcarea acestuia.

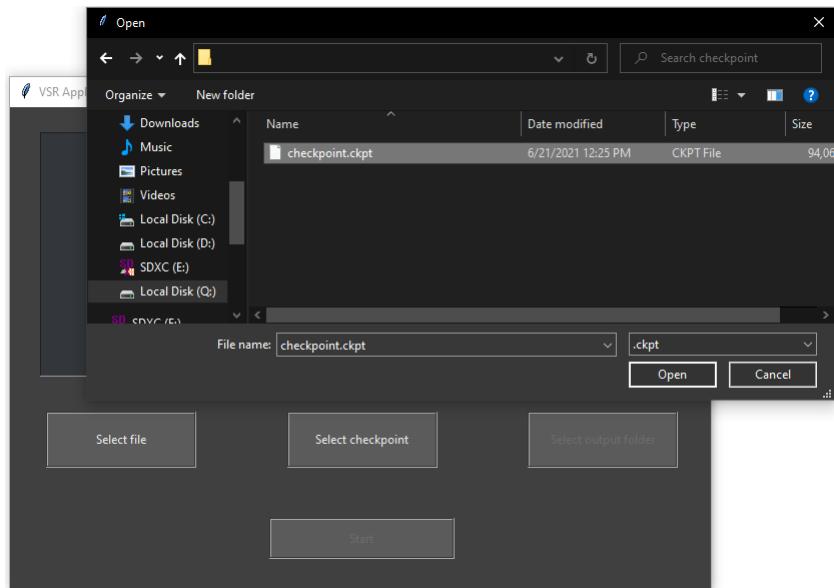


Figura 7.3: Pentru deschiderea ferestrei de selectare fișierului ce conține modelele pre-antrenate se va folosi butonul "Select checkpoint". Odată ce fereastra este deschisă, utilizatorul va selecta fișierul dorit și va apăsa butonul "Open" pentru încărcarea acestuia.

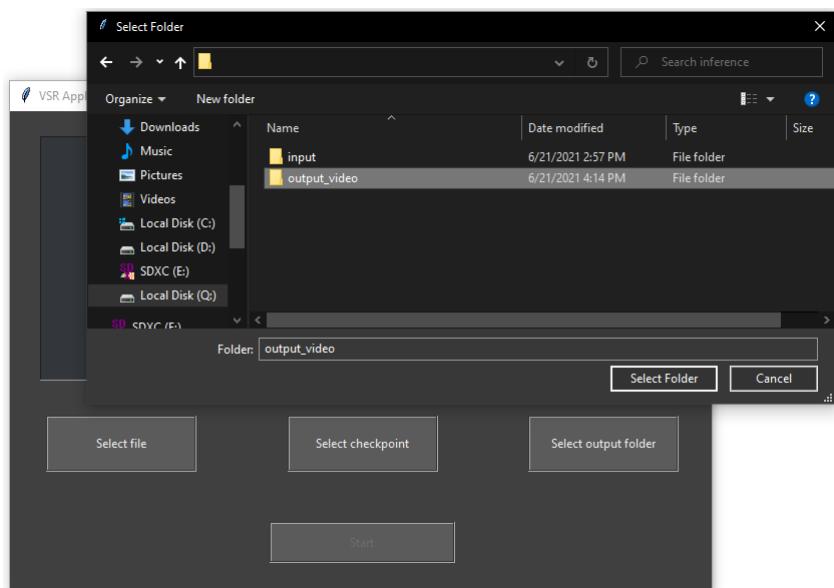


Figura 7.4: Pentru deschiderea ferestrei de selectare directorului de ieșire se va folosi butonul "Select output folder". Odată ce fereastra este deschisă, utilizatorul va selecta directorul țintă și va apăsa butonul "Select folder".

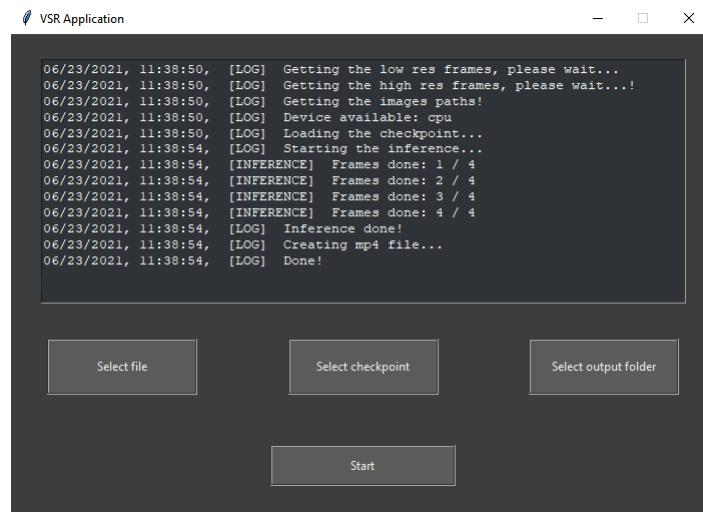


Figura 7.5: Obținerea videoclipului va fi finalizată în momentul afișării mesajului "Done!" pe ecran. Statusul procesului poate fi urmărit de către utilizator, numerele oferite reprezentând numărul de cadre obținute și numărul total de cadre al videoclipului selectat.

## **Capitolul 8**

# **Concluzii și direcții ulterioare de dezvoltare**

Obținerea videoclipurilor prin tehnici de super rezoluție reprezintă un subiect de actualitate, cu o popularitate în continuă creștere. În fiecare zi în întreaga lume, milioane de ore de conținut video sunt vizualizate prin intermediul diferitelor surse precum cablu TV, Blu-Ray, platforme de streaming online sau DVD. Odată cu apariția dispozitivelor ce oferă cadre la rezoluții HR, UHD 4K sau chiar 8k, a apărut și necesitatea de conținut cu o claritate înaltă. Tehnicile de super-rezoluție astfel, reprezintă o soluție fiabilă în locul aparaturilor costisitoare de capturare a conținutului cu o rezoluție ridicată. Necesitatea de conținut multimedia nu este singura problema rezolvată prin super-rezoluție, unele dintre exemplele demne de menționat fiind: astronomia, supravegherea video, jocuri video, procesarea de imagini, microbiologia sau medicina.

Abordările acestei lucrări a luat diverse forme, precum: obținerea setului de date pentru antrenarea rețelelor de super-rezoluție aplicate pe videoclipuri, înlocuirea rețelei de super-rezoluție cu modele asemănătoare, introducerea unor noi funcții de cost pentru o instruire mai eficientă și modificarea arhitecturii de bază prin adăugarea de noi concepte pentru blocurile reziduale folosite în componența rețelele neurale reziduale.

Punct forte reprezentat de derularea diferitelor experimente pentru a studia comportamentelor rețelelor de super-rezoluție în timpul antrenamentului și efectele asupra rezultatelor obținute. De asemenea, modelul propus beneficiază de o interfață grafică prin intermediul căruia utilizatorul poate studia rezultatele în urma trecerii unui videoclip cu o rezoluție scăzută, prin arhitectură pre-antrenată dedicată problemei de super-rezoluție aplicată pe videoclipuri.

Câteva dintre sugestiile de dezvoltare a proiectului ce pot fi luate în considerare pentru îmbunătățirea rezultatelor sunt:

- **Îmbunătățirea setului de date.** Pentru problema de super-rezoluție aplicată pe videoclipuri, setul de date folosit în antrenare rețelelor joacă un rol esențial în obținerea de rezultate calitative. Achiziția de date noi, claritate videoclipurilor de antrenare, dar și diversitatea acestora sunt câteva aspecte de luat în considerare.
- **Optimizarea codului și o implementare cât mai fidelă.** Optimizarea și replicarea cât mai fidelă a pașilor necesari pentru antrenarea arhitecturii de super-rezoluție poate duce la obținerea unor rezultate cu un aspect vizual mai plăcut.
- **Obținerea valorilor potrivite pentru hiper-parametrii.** Valorile hiperparametrilor reprezintă un pas important în antrenarea unei rețele neurale. Prin determinarea valorilor potrivite obținem astfel un proces de instruire mai stabil și rezultate mai calitative. Acest lucru se poate face prin experimentarea cu diferite seturi de hiper-parametrii pentru antrenare modelelor sugerate în decursul acestei lucrări.

## 8.1 Experiența dobândită

Lucrarea prezentată reprezintă o lucrare de reproducere și extindere de modele existente, prin urmare a fost necesară documentarea amănunțită înaintea pasului de proiectare, pe tot parcursul acestuia fiind necesară înțelegerea articolelor științifice. Au fost dobândite astfel cunoștințe din domeniul Deep Learning, susținute de punerea în practică a diferitelor concepte întâlnite. Dezvoltarea acestei lucrări a reprezentat astfel un prim pas către domeniul Machine Learning, caracterul explorativ oferit de problema alesă a permis dobândirea de fundamente din arii ale domeniului viziunii computaționale.

# Bibliografie

- [1] Herminio Chavez-Roman, Volodymyr Ponomaryov **and** Ricardo Peralta-Fabi. “Image super resolution using interpolation and edge extraction in wavelet transform space”. **in***2012 9th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*: 2012, **pages** 1–6. **DOI:** 10.1109/ICEEE.2012.6421202.
- [2] Mengyu Chu, You Xie, Jonas Mayer, Laura Leal-Taixé **and** Nils Thuerey. “Learning temporal coherence via self-supervision for GAN-based video generation”. **in***ACM Transactions on Graphics*: 39.4 (july 2020). **ISSN:** 1557-7368. **DOI:** 10.1145/3386569.3392457. **URL:** <http://dx.doi.org/10.1145/3386569.3392457>.
- [3] Chao Dong, Chen Change Loy, Kaiming He **and** Xiaoou Tang. *Image Super-Resolution Using Deep Convolutional Networks*. 2015. **arXiv:** 1501.00092 [cs.CV].
- [4] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers **and** T. Brox”. *FlowNet: Learning Optical Flow with Convolutional Networks*. 2015. **URL:** <http://lmb.informatik.uni-freiburg.de/Publications/2015/DFIB15>.
- [5] Sergey Ioffe **and** Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. **arXiv:** 1502.03167 [cs.LG].
- [6] Justin Johnson, Alexandre Alahi **and** Li Fei-Fei. *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. 2016. **arXiv:** 1603.08155 [cs.CV].
- [7] Armin Kappeler, Seunghwan Yoo, Qiqin Dai **and** Aggelos K. Katsaggelos. “Video Super-Resolution With Convolutional Neural Networks”. **in***IEEE Transactions on Computational Imaging*: 2.2 (2016), **pages** 109–122. **DOI:** 10.1109/TCI.2016.2532323.

- [8] Diederik P. Kingma **and** Jimmy Ba.  
*Adam: A Method for Stochastic Optimization.* 2017.  
arXiv: 1412.6980 [cs.LG].
- [9] Alex Krizhevsky, Ilya Sutskever **and** Geoffrey E Hinton.  
“ImageNet Classification with Deep Convolutional Neural Networks”.  
in*Advances in Neural Information Processing Systems:*  
**byeditor**F. Pereira, C. J. C. Burges, L. Bottou **and** K. Q. Weinberger.  
**volume** 25. Curran Associates, Inc., 2012.  
URL: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.
- [10] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero,  
Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani,  
Johannes Totz, Zehan Wang **and** Wenzhe Shi. *Photo-Realistic Single Image  
Super-Resolution Using a Generative Adversarial Network.* 2017.  
arXiv: 1609.04802 [cs.CV].
- [11] Alice Lucas, Santiago Lopez-Tapia, Rafael Molina **and**  
Aggelos K. Katsaggelos. “Generative Adversarial Networks and Perceptual  
Losses for Video Super-Resolution”. in*IEEE Transactions on Image  
Processing: 28.7 (july 2019)*, **pages** 3312–3327. ISSN: 1941-0042.  
DOI: 10.1109/tip.2019.2895768.  
URL: <http://dx.doi.org/10.1109/TIP.2019.2895768>.
- [12] Takeru Miyato, Toshiki Kataoka, Masanori Koyama **and** Yuichi Yoshida.  
*Spectral Normalization for Generative Adversarial Networks.* 2018.  
arXiv: 1802.05957 [cs.LG].
- [13] David E. Rumelhart, Geoffrey E. Hinton **and** Ronald J. Williams.  
“Learning Representations by Back-propagating Errors”.  
in*Nature: 323.6088 (1986)*, **pages** 533–536. DOI: 10.1038/323533a0.  
URL: <http://www.nature.com/articles/323533a0>.
- [14] Mehdi S. M. Sajjadi, Raviteja Vemulapalli **and** Matthew Brown.  
*Frame-Recurrent Video Super-Resolution.* 2018.  
arXiv: 1801.04590 [cs.CV].
- [15] Tim Salimans **and** Diederik P. Kingma. *Weight Normalization: A Simple  
Reparameterization to Accelerate Training of Deep Neural Networks.*  
2016. arXiv: 1602.07868 [cs.LG].
- [16] Karen Simonyan **and** Andrew Zisserman.  
*Very Deep Convolutional Networks for Large-Scale Image Recognition.*  
2015. arXiv: 1409.1556 [cs.CV].

- [17] Deqing Sun, Xiaodong Yang, Ming-Yu Liu **and** Jan Kautz. *PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume*. 2018. arXiv: 1709.02371 [cs.CV].
- [18] Xin Tao, Hongyun Gao, Renjie Liao, Jue Wang **and** Jiaya Jia. *Detail-revealing Deep Video Super-resolution*. 2017. arXiv: 1704.02738 [cs.CV].
- [19] Tong Tong, Gen Li, Xiejie Liu **and** Qinquan Gao. “Image Super-Resolution Using Dense Skip Connections”. in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*: october 2017.
- [20] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao **and** Xiaoou Tang. *ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks*. 2018. arXiv: 1809.00219 [cs.CV].
- [21] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong **and** Yun Fu. *Residual Dense Network for Image Super-Resolution*. 2018. arXiv: 1802.08797 [cs.CV].