

**Sistem expert pentru optimizarea fișierelor media**

**Coordonator ştiinţific:                                                             Absolvent:**

**Doris Pogăcean Gabriel Comănescu**

**București, 2021**

Contents

[INTRODUCERE 3](#_Toc75014029)

[**Motivarea alegerii temei** 3](#_Toc75014030)

[**Obiectivele propuse in cadrul lucrării** 3](#_Toc75014031)

[**Structura lucrării de licență** 3](#_Toc75014032)

[1. ASPECTE TEORETICE REFERITOARE LA APLICAȚII DE INTELIGENȚĂ ARTIFICIALĂ 4](#_Toc75014033)

[**1.1. Aplicații ale inteligenței artificiale în diferite domenii** 4](#_Toc75014034)

[**1.1.1.** **Aplicări ale inteligenței artificiale în domeniul medicinii:** 4](#_Toc75014035)

[**1.1.2.** **Mașini care se conduc singure:** 5](#_Toc75014036)

[**1.1.3.** **Aplicații în industria jocurilor și algoritmi care învață din experiență:** 6](#_Toc75014037)

[**1.2.** **Inteligența artificială și machine learning** 8](#_Toc75014038)

[**1.3.** **Despre machine learning** 10](#_Toc75014039)

[**1.3.1.** **Rețele neuronale artificiale** 10](#_Toc75014040)

[**1.3.2.** **Rețele neuronale convoluționale** 13](#_Toc75014041)

[**1.3.3.** **Tipuri de machine learning** 16](#_Toc75014042)

[**1.3.4.** **Rețele Generativ Adversariale (GAN)** 17](#_Toc75014043)

[2. TEHNOLOGII ȘI INSTRUMENTE SOFTWARE FOLOSITE ÎN IMPLEMENTAREA PROIECTULUI 18](#_Toc75014044)

[**2.1. Visual Studio Code** 18](#_Toc75014045)

[**2.2. Python** 19](#_Toc75014046)

[**2.3. PyTorch** 19](#_Toc75014047)

[**2.4. NumPy** 20](#_Toc75014048)

[**2.4. Pandas** 20](#_Toc75014049)

[**2.4. Matplotlib** 21](#_Toc75014050)

[**2.4. Anaconda** 21](#_Toc75014051)

[**2.4. HTML** 21](#_Toc75014052)

[**2.4. CSS** 21](#_Toc75014053)

[**2.4. JavaScript** 22](#_Toc75014054)

[**2.4. Flask** 22](#_Toc75014055)

[3. STUDIU DE CAZ „SISTEM EXPERT PENTRU OPTIMIZAREA FIȘIERELOR MEDIA” 24](#_Toc75014056)

[**3.1.** **Prima rețea neuronală: Optimizarea rezoluției imaginilor** 25](#_Toc75014057)

[**3.1.1.** **Pregătirea setului de date** 25](#_Toc75014058)

[**3.1.2.** **Definirea funcțiilor de pierdere** 25](#_Toc75014059)

[**3.1.3.** **Modelul generatorului** 26](#_Toc75014060)

[**3.1.4.** **Modelul discriminatorului** 28](#_Toc75014061)

[**3.1.5.** **Antrenamentul** 29](#_Toc75014062)

# **INTRODUCERE**

## **Motivarea alegerii temei**

Tehnologiile utilizate sunt căutate si folosite într-o sfera larga de domenii. Întrucât inteligența artificială este o știință aflata la început de drum, denota un potențial extrem de mare sa se dezvolte pe o scara si mai larga in viitor. Majoritatea companiilor care vor sa se dezvolte au adoptat, sau încep sa adopte, implementarea de programe bazate pe inteligenta artificiala in produsele lor. Aceasta ramura a informaticii este in stare sa automatizeze toate lucrurile care nu pot fi preprogramate si aduce si mai multa comoditate in viața noastră.

Am ales sa realizez o astfel de aplicație deoarece am fost întotdeauna pasionat de automatizarea treburilor monotone si repetitive, iar aceasta tehnologie este in stare sa spargă bariera impusa de complexitatea unor probleme ce nu pot fi rezolvate de un program simplu. Sper ca aceasta lucrare sa fie primul pas pe care îl voi face către o cariera plina de proiecte care sa aducă comoditate in viețile oamenilor.

Întrucât calitatea unui produs este extrem de importanta, atât pentru consumator, cat si pentru producător, am ales o tema care reprezintă îmbunătățirea calității produselor media fără a se depune mult efort in plus.

## **Obiectivele propuse in cadrul lucrării**

In cadrul dezvoltării aplicației cu titlul „Sistem expert pentru optimizarea fișierelor media”, mi-am propus sa realizez doua rețele neuronale antrenate sa îmbunătățească calitatea imaginilor si videoclipurilor:

* Prima rețea se ocupa cu mărirea calității unei imagini. Este in stare sa dubleze rezoluția de baza a unei imagini, astfel mărindu-i calitatea si claritatea.
* A doua rețea se ocupa cu inserarea unor cadre noi in videoclipuri cu scopul de a le face mult mai fluide.

Ambele rețele vor rula pe un server si vor putea fi accesate prin intermediul unui site web

## **Structura lucrării de licență**

**De completat mai incolo**

# **ASPECTE TEORETICE REFERITOARE LA APLICAȚII DE INTELIGENȚĂ ARTIFICIALĂ**

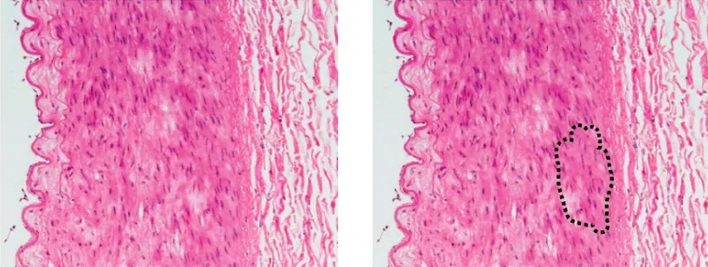
Inteligenta Artificiala (AI) se refera la simularea inteligentei umane in mașinării care sunt programate sa gândească si sa copieze acțiunile oamenilor. Termenul se poate aplica oricărei mașinării care prezinta trăsături asociate cu mintea umana, cum ar fi învățarea si rezolvarea de probleme.

Caracteristica ideală a inteligenței artificiale este abilitatea acesteia să raționalizeze și să aleagă acțiunile care au cea mai mare șansă de reușită în urmărirea unui scop. Aceasta se bazează pe principiul că inteligența umană poate fi definită într-un mod în care poate fi ușor replicată de un program și folosită sa execute o anumită sarcină, de la cele mai simple, până la unele foarte complexe. În domeniul de cercetare se fac progrese imense în replicarea activităților ca și învățarea, percepția, raționarea, în măsura în care acestea pot fi concret definite.

## **1.1. Aplicații ale inteligenței artificiale în diferite domenii**

### **Aplicări ale inteligenței artificiale în domeniul medicinii:**

- În 2018, cercetătorii din Colegiul Național de medicină din Seoul au dezvoltat un algoritm AI numit DLAD (Deep Learning based Automatic Detection) pentru a analiza radiografii ale pieptului si a detecta creșterile anormale ale celulelor, cum ar fi potențialul cancer. Algoritmul a avut o performanță mai bună decât 17 din 18 doctori.



Figură 1.1: Panoul din stânga reprezintă imaginea introdusă în algoritm. Panoul din dreapta reprezintă o regiune a unor celule potențial periculoase, identificate de algoritm, care ar trebui examinate mai atent.

- O alta aplicație creată tot în anul 2018 de către cercetătorii de la Google se numește LYNA (Lymph Node Assistant), care a analizat exemple de țesuturi afectate pentru a identifica tumorile de cancer la sâni din biopsii ale ganglionilor limfatici. Acesta nu este primul algoritm care încearcă să facă asta, însă a reușit să identifice regiuni suspicioase imperceptibile ochiului uman in biopsiile oferite. Lyna a fost testat pe două seturi de date diferite să determine dacă exemplul este canceros sau nu și a răspuns corect 99% din cazuri. Pe deasupra, programul a reușit sa înjumătățească timpul de analiză al fiecărei biopsii comparativ cu doctorii.

### **Mașini care se conduc singure:**

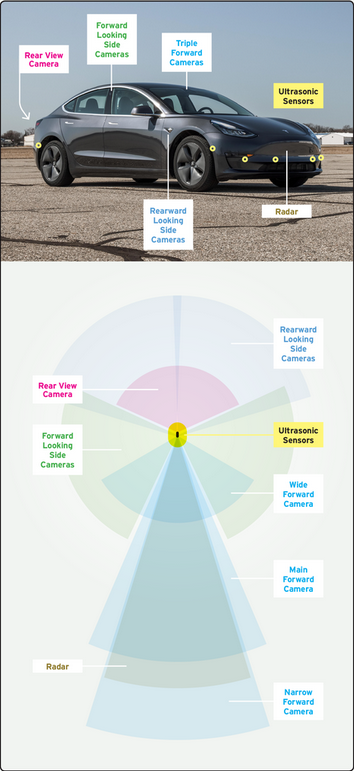
Începând cu anul 2010, mulți producători mari de mașini au început sa introducă ideea de mașini care se conduc singure în linia de producție, cum ar fi: Ford, Mercedes, Volkswagen, Audi...

În 2015, Tesla Motors a introdus tehnologia proprie de autopilot, fiind îmbunătățită in 2016 în urma unui accident de mașină. Acum mașinile Tesla sunt dotate cu 8 camere și 12 senzori cu ultrasunete, pe lângă radarul frontal cu procesare îmbunătățită.

Exista două pachete disponibile pentru orice model de Tesla: Autopilot și Full Self-Driving Capability. Al doilea pachet este cel superior și considerabil mai scump. Lucruri de care este capabil reprezintă parcarea automată, schimbarea benzilor automat, summon (mașina poate să iasă singură din parcare și să vină către tine). Abilități care încă se află în stagiul de beta reprezintă identificarea semnelor de circulație și luarea de acțiuni corespunzătoare atunci când este nevoie, intrarea și ieșirea automată de pe autostradă. Alte întrebuințări mai pasive includ frâna de urgență când se detectează o posibilă coliziune, decelerarea când mașina este prea aproape de un alt vehicul în față, atenționarea de obstacole aflate prea aproape de mașină la schimbarea benzilor.

Mașina atenționează șoferii să își țină întotdeauna mâinile pe volan în timpul condusului.

În 2013 compania NuTonomy a fost fondată. Aceasta a început ca o subcompanie de la MIT și se ocupă cu dezvoltarea mașinilor care se conduc singure și roboților autonomi.



Figură 1.: Tesla Model 3 sistem de navigare

În anul 2016, compania a lansat, în Singapore, serviciul de robo-taxi. Aceasta implică un taxi care se conduce singur și este folosit încă astăzi.

### **Aplicații în industria jocurilor și algoritmi care învață din experiență:**

Figură 1.:Robo-taxi lansat în Singapore de compania NuTonomy

Modele de învățare AI cele mai apropiate de modul în care învață oamenii sunt cele care învață prin încercare și eroare din experiență. Voi dezvolta puțin mai încolo în lucrare metoda folosită. Aceste programe sunt testate, în momentul de față, în învățarea jocurilor, unde sunt aruncate într-un anumit joc fără nicio cunoștință despre acesta în afara regulilor de bază și sunt lăsate să joace singure sute, în unele cazuri mii de jocuri, învățând astfel strategii complexe.

- Primul exemplu creat de compania Deepmind de la Google, este programul AlphaGo. Programul a fost antrenat să joace jocul go, un joc foarte popular în China de strategie în care fiecare jucător mută, pe rând, piese albe, respectiv negre pe tabla. Jocul este mult mai complex decâ șahul, existând 10^170 de posibile mutări. Astfel era considerat imposibil ca un program să poată învăța să joace go. În China, copiii care dau dovadă de talent sunt puși în școli speciale să învețe să joace go.

Totuși programul AlphaGo, lansat în 2015 s-a bazat pe un algoritm de căutare și a folosit experiența căpătată în urma a mii de jocuri jucate contra el însăși să perfecționeze jocul de go. În anul 2015 a devenit primul prgram care a bătut un jucător profesional de go, iar în 2016 l-a învins pe Lee Sedol, un multiplu campion internațional. A fost prima dată când un program a fost in stare să bată un jucător 9-dan. În al 4-lea meci AlphaGo a făcut o mutare care a fost considerată o greșeală de începător, oamenii find convinși că a pierdut meciul. Însă la final, strategia creată de program a ieșit la iveală, reușind să câștige meciul datorită acelei mutări. Acum mutarea respectivă este predată în școlile de go.

Pe parcursul anilor programul a avut mai multe versiuni până la finalul anului 2017 unde Deepmind au creat un nou program intitulat AlphaZero. Acesta nu mai folosea niciun algoritm de căutare predefinit, ci doar să învețe din experiențe. Programul s-a dovedit a fi exponențial mai bun decât predecesorul său, reușind să îl învingă pe AlphaGo 200 la 0.

AlphaZero a fost apoi antrenat să joace și șah, iar după numai 2 ore de jucat jocuri cu el însăși, a reușit să învingă cel mai performant program de șah, Stockfish, câștigând 28 din 100 de meciuri jucate, restul terminându-se în remize.

În momentul de față, AlphaZero deține locul 1 mondial în șah, go și shogi.

- Al doilea exemplu, OpenAI au început sa lucreze la proiectul OpenAI Five în anul 2016. Programul a fost antrenat să învețe jocul Dota2, un joc extrem de complex de înțeles și cu mult mai multe mecanici, opțiuni, decizii de luat și alegeri de făcut în fiecare secundă.

În 2017 a câștigat un meci 1 la 1 împotriva unui jucător profesional de Dota 2. Acesta a spus că s-a simțit de parcă joacă împotriva unui om care nu face alegeri greșite. Însă dota este un joc de echipa 5 contra 5, iar în anul 2019 a reușit să învingă echipa OG în finală devenind primul AI care a reușit să învingă campionii mondiali la un joc competitiv. Tot în 2019, OpenAI Five a început să joace pe internet contra oameni și echipe având o rată de câștig de 99,4%.

Un joc de dota rulează la 30 de cadre pe secundă având o medie de 45 minute pe joc, rezultând în 80.000 de acțiuni pe fiecare joc. OpenAI Five observă și decide o acțiune odată la 4 cadre, adică 20.000 de alegeri într-un joc. Unitățile și clădirile pot vedea în jurul lor, însă restul hărții este acoperită în ceață, deci modelul trebuie să aleagă acțiuni bazate pe date incomplete. De asemenea, în Dota, fiecare erou poate executa mii de acțiuni. Spațiul a fost discretizat în 170.000 de acțiuni posibile pe fiecare erou, nu toate fiind valabile în fiecare moment. După un calcul s-a ajuns la rezultatul de 1.000 de acțiuni valabile pentru un erou în fiecare moment. Modelul a observat jocul folosind un API de la Valve, primind 20.000 de puncte de informație diferite în fiecare cadru.

Acestea sunt problemele cu care au avut de a face persoanele de la Open AI. Pentru a putea produce destulă putere de computare, având în vedere că trebuia să facă alegerile în timp real.

Pentru CPU au folosit 128.000 de procesoare rulând în Google Cloud.

Pentru GPU au folosit 256 de plăci video P100 în Google Cloud.

Fiecare observație avea 36.8kB de date cu un număr de 7.5 observații pe secundă.

La finalul unei zile colecționa echivalentul a ~180 de ani de antrenament (~900 de ani pe zi dacă numărăm fiecare erou separat)

## **Inteligența artificială și machine learning**

Inteligența artificială și machine learning fac parte din aria de computer science și sunt corelate una cu cealaltă. Aceste două tehnologii sunt cele mai folosite pentru crearea sistemelor inteligente. Deși sunt foarte apropiate una de cealaltă, iar oamenii le folosesc câteodată ca și sinonime, sunt termeni diferiți folosiți in cazuri diferite.

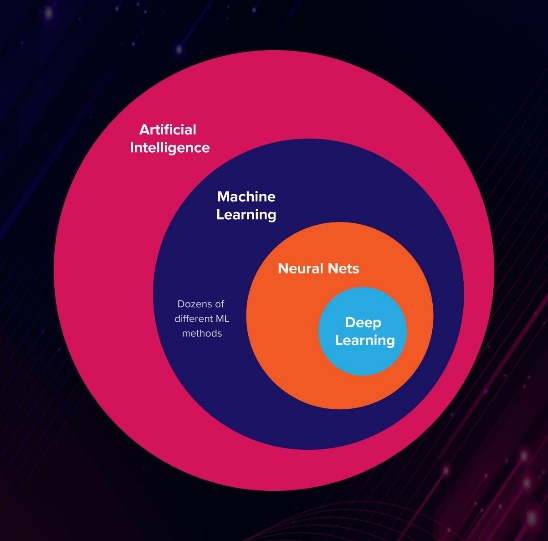
Diferențe cheie între inteligența artificială și machine learning:

**Inteligența artificială**

* Reprezintă o tehnologie care permite mașinăriilor să simuleze comportament uman
* Scopul este de a crea sisteme complexe asemănătoare oamenilor pentru a rezolva probleme complexe
* În AI se folosesc sisteme inteligente pentru a îndeplini orice sarcină ca un om
* Machine learning este un subdomeniu al inteligenței artificiale
* AI are o gamă foarte largă de domenii
* AI lucrează să creeze sisteme inteligente care pot îndeplinii sarcini complexe
* Sistemele AI caută să crească șansele de succes
* Aplicațiile principale sunt: Siri, relații cu clienții folosind roboți, sisteme expert, jocuri online, roboți humanoizi
* Include învățare, raționament și autocorectare
* Are de a face cu date structurate, semi-structurate si nestructurate

**Machine learning**

* ML este un subset al inteligenței artificiale care permite mașinăriilor să învețe singure din date anterioare fără a fi specific programate
* Scopul este de a permite mașinăriilor să învețe din date anterioare pentru a da un răspuns exact
* În ML mașinăriile sunt antrenate să îndeplinească sarcini particulare pe baza unor date și să ofere un răspuns exact
* ML are o gamă de domenii restrânsă
* ML lucrează să creeze mașinării care execută numai sarcinile specifice pentru care sunt antrenate
* Machine learning caută tipare și să aibă o acuratețe mare
* Aplicațiile principale sunt: sistemele de recomandări online, algoritmele de căutare Google, sugestiile de prieteni pe Facebook
* Include învățare și autocorectare când date noi sunt introduse
* Are de a face cu data structurată și semi-structurată



Figură 1.: Diferența între AI și ML

La un nivel larg, putem definii AI și ML in felul următor:

„AI este un concept mai mare folosit pentru a crea mașinării inteligente care pot simula comportamentul și abilitatea umană de a gândi, pe când ML este o arie sau un subdomeniu al inteligenței artificiale care permite mașinăriilor să învețe din date fără a fi programate explicit”.

## **Despre machine learning**

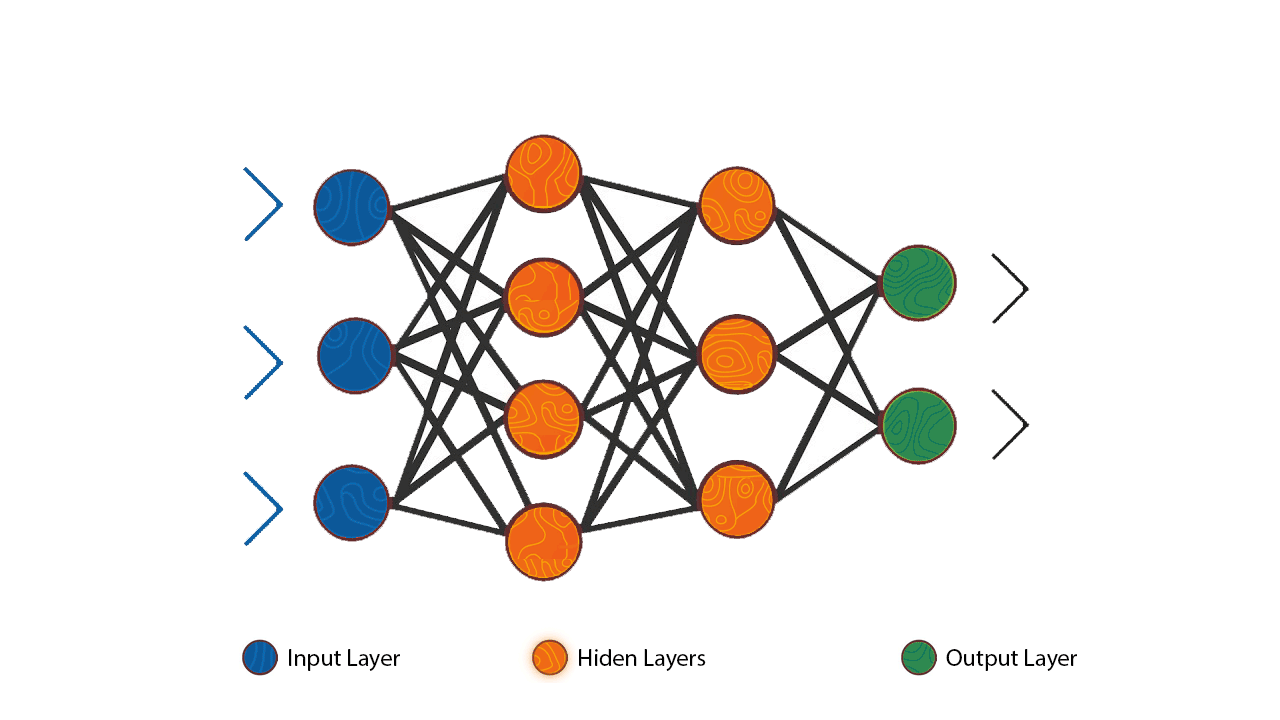
Machine learning funcționează prin intermediul unor algoritmi care învață singuri folosind date. Funcționează numai în domenii specifice cum ar fi dacă creăm un model care detectează imagini cu câini, va oferi rezultate numai pentru imagini cu câini. Dacă ar fi să îi arătăm poze cu pisici ar deveni inutil.

### **Rețele neuronale artificiale**

O rețea neuronala este o serie de algoritmi care caută să descopere relații și legături într-un set de date printr-un proces care copiază modul în care creierul uman operează. În acest sens, nodurile dintr-o rețea sunt numiți neuroni artificiali.

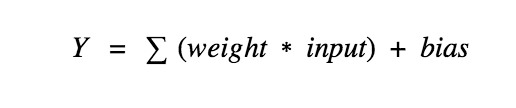
După cum se poate observa, o rețea neuronală simplă constă în 3 straturi: intrare, strat ascuns și ieșire. Stratul de intrare primește datele de intrare, stratul ascuns procesează datele, iar stratul de ieșire produce rezultatul.

Liniile care leagă neuronii între ei sunt numite greutăți, fiecare reprezentând o valoare care afectează neuronii la care sunt legate. Valorile fiecărei greutăți sunt ajustate pe parcursul antrenamentului până când rețeaua ajunge în punctul dorit. Pe lângă greutăți, există și bias-uri, care sunt atribuite fiecărui neuron. Acestea, de asemenea, sunt ajustate pe parcursul antrenamentului.



Figură 1.: Exemplu rețea neuronală simplă

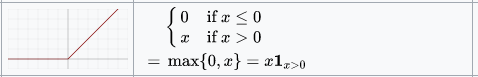
Modul în care rețeaua se antrenează este, mai mult sau mai puțin, un proces de operații matematice, în care, pentru fiecare neuron, fiecare greutate se înmulțește cu valoarea neuronului, se adună toate împreună cu bias-ul și sunt trecute printr-o funcție de activare.



Figură rezultatul unui neuron

Y reprezintă rezultatul dinainte de funcția de activare. Funcția de activare diferă în funcție de problema cu care avem de a face. Aceasta are rolul de a modul în care rezultatul y este transformat în date de intrare pentru neuronii de pe stratul următor. Alegerea funcției de activare are un mare impact asupra capacității și performanței ale rețelei neuronale. Diferite funcții de activare pot fi folosite în diferite părți ale modelului.

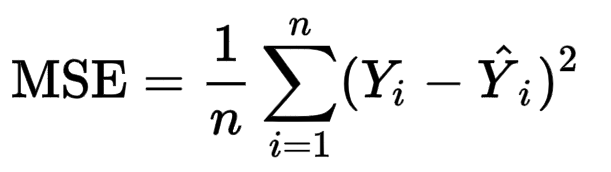
Cea mai des întâlnită și folosită funcție este ReLU (Rectified linear unit). Aceasta scapă de orice valoare negativă, transformând în zero tot ce este mai mic decât zero, iar restul datelor rămân neschimbate.



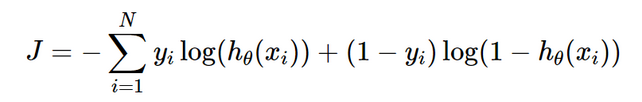
Figură 1.7: ReLU: Grafic și funcție

După ce rețeaua neuronală trece prin toți neuronii și calculează datele de ieșire, acestea sunt evaluate folosind funcții de eroare. O funcție de eroare definește un obiectiv cu ajutorul căruia modelul se autoevaluează. Cu alte cuvinte, funcția de eroare ia datele de ieșire ale modelului, împreună cu datele reale la care dorim să ajungem și le trece printr-un algoritm care are ca rezultat o eroare. Scopul general al modelului este să minimizeze eroare cât mai mult posibil.

Un exemplu de funcție de eroare foarte des folosită în probleme de regresie liniară este Mean Squared Error (MSE). Aceasta ridică la puterea a doua diferența dintre rezultatul modelului și rezultatul real pentru a scoate în evidență și a accentua eroare existentă.

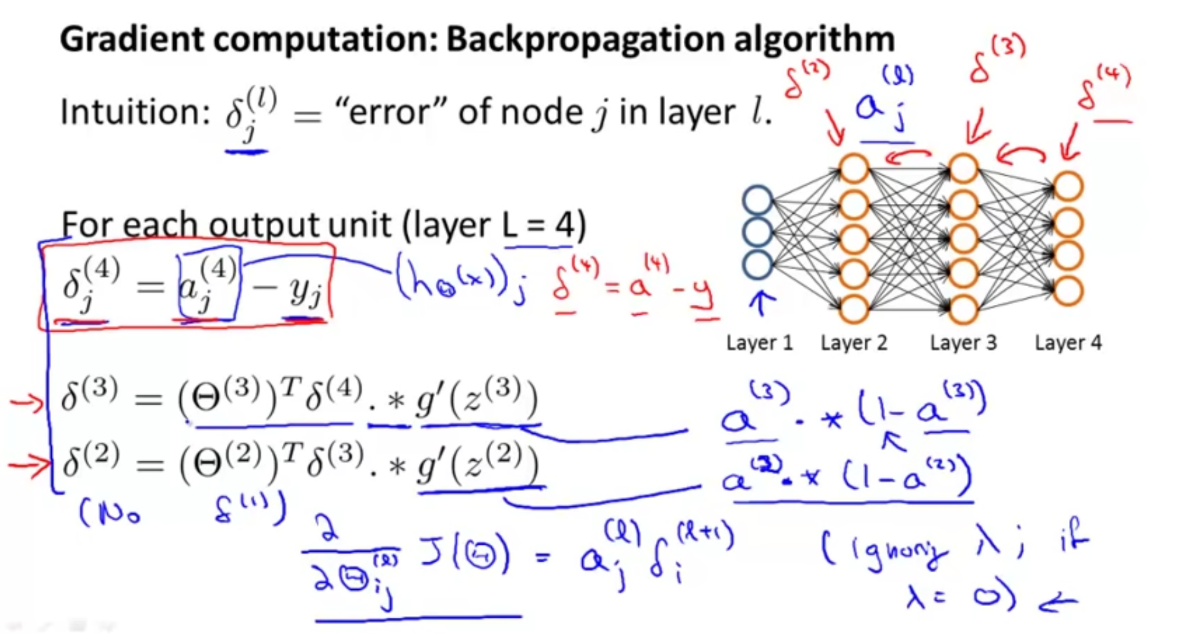


Figură 1.8: Mean Squared Error

Un alt exemplu ar fi Binary Cross Entropy Loss. Aceasta este utilizată în probleme de clasificare unde avem numai două clase și poate fi generalizată la mai multe. Funcția are ca rezultat probabilitatea pentru fiecare clasă posibilă să fie răspunsul. De exemplu, dacă vrem să clasificăm poze cu pisici și câini, și avem o poză cu o pisică pe care o introducem în rețea, aceasta va avea o probabilitate foarte mare să fie în clasa pisică și una aproape de zero să fie în clasa câine.

Figură 1.9: Binary Cross Entropy Loss Function

Odată ce am calculat eroarea ajungem la partea care face algoritmul să „învețe”. Folosind eroarea calculată, o vom lua de la ultimul strat către primul calculând gradientul fiecărui strat. Gradientul reprezintă valoarea de ajustare a fiecărei greutăți de pe un strat. Gradientul este calculat folosind eroarea, în cazul ultimului strat, după care sunt folosiți gradienții anteriori. Aceștia sunt înmulțiți cu neuronii de pe stratul respectiv și derivata funcției de activare.

La final, folosind gradienții calculați pentru fiecare strat, sunt ajustate toate greutățile pe baza gradienților. Acesta reprezintă un proces de antrenament pentru un singur exemplu de intrare. În mod normal, în machine learning, sunt folosite pachete de sute sau mii de date pentru care se calculează greutățile, și la finalul fiecărui pachet se execută backpropagation. De exemplu, putem avea 60.000 de date de intrare diferite și le luăm în pachete de câte 600. Asta înseamnă că după ce execută antrenamentul de 1.000 de ori, modelul va fi trecut prin toate datele de antrenament valabile. În momentul acesta, modelul a executat o epocă de antrenament. În funcție de datele pe care le avem și scopul modelului, acesta se poate antrena de la 20-100 de epoci pentru modelele simple, până la mii sau zeci de mii în cadrul modelelor complexe, care pot avea chiar mai multe date.

Figură 1.10: Backpropagation

Antrenamentul unui astfel de model pare foarte costisitor, însă cu ajutorul tehnologiilor recente, antrenamentul poate fi desfășurat pe placa video, în loc de procesor lucru care mărește exponențial performanța și timpul de execuție. Ca și exemplu, un model menționat mai sus, compus dintr-o rețea neuronală simplă, având 60.000 de date de antrenament care nu reprezintă poze sau fișiere mari, va fi antrenat pentru aproximativ 15 epoci. Dacă antrenamentul este rulat pe procesor, tot procesul poate dura până la 15-20 minute pe un procesor mai bun și se va duce către o jumătate de ora pe un procesor slab. Acum, dacă trecem același antrenament pe o placă video modestă, antrenamentul o să dureze aproximativ 1-2 minute, ceea ce este o diferență enorma. Dacă folosim și o placa video mai performantă putem ajunge la sub 1 minut.

Am menționat timpii de mai sus pentru un model simplu antrenat numai pe date numerice și simplu de interpretat. Timpul de execuție va crește considerabil pentru date mai complexe și pentru modele mai complexe. Mai departe voi vorbi despre antrenamentul modelelor folosind imagini ca și date de intrare. Pentru acestea se folosește un tip diferit de rețele care se numesc rețele neuronale convoluționale.

### **Rețele neuronale convoluționale**

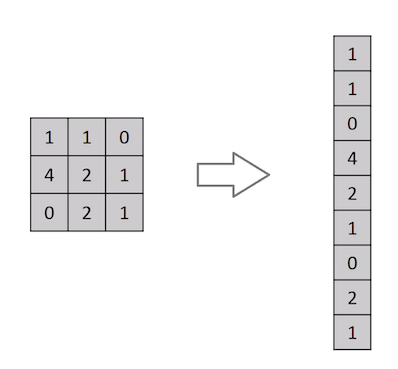
Rețelele convoluționale neuronale se folosesc de filtre, sau „nuclee”(kernels), folosite pentru a extrage trăsături relevante din datele de intrare folosind operații de convoluție. Sunt folosite în mod special în lucrul cu imagini și procesare video.

Arhitectura unei rețele convoluționale este similară cu conectivitatea neuronilor din creierul uman și este inspirată de organizarea cortexului vizual. Neuronii individuali răspund la stimuli numai în zone restrânse ale câmpului vizual, cunoscut ca și câmpul receptiv. O parte a acestor câmpuri se suprapun pentru a acoperi toată zona vizuală.

**De ce rețele convoluționale?**

O imagine nu este nimic mai mult decât o matrice de pixeli, deci în teorie ar trebui să poată fi transpusă într-un vector. Lucru care este posibil, iar în cazul unor imagini extrem de simple, alb negru, metoda ar putea avea o precizie destul de bună. Însă în momentul acesta se fiecare pixel pierde toate datele legate de pixelii adiacenți. Fiecare pixel depinde de pixelii din jurul său pentru a forma o imagine.

O rețea convoluțională este capabilă să captureze dependențele spațiale dintr-o imagine prin aplicarea filtrelor. Arhitectura aceasta se potrivește mult mai bine în lucrul cu imaginile datorită numărului de parametrii redus.



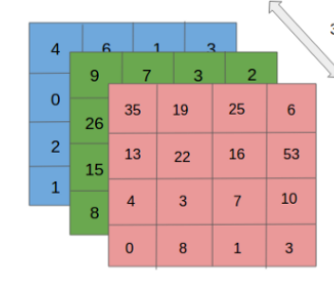
Figură 1.11: filtrarea unei imagini 3x3 într-un vector 9x1

**Imaginea de intrare**

Orice imagine care nu este alb-negru va fi împărțită în 3 canale de culoare pe parcursul procesării: roșu, verde și albastru. Pozele alb-negru au un singur canal de culoare.

Lucrurile pot deveni foarte intense din punct de vedere computațional la imagini de rezoluție ridicată. De aceea unul din rolurile unei rețele convoluționale este de a reduce imaginea într-o forma care este mult mai ușor de procesat, fără a pierde caracteristicile care sunt importante pentru prezicere.

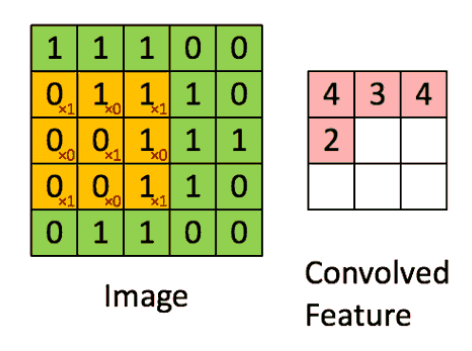
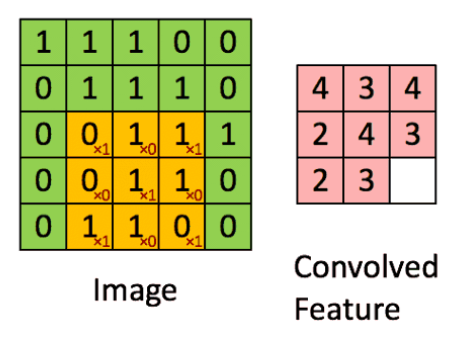
Acest lucru este foarte important în crearea modelului, întrucât trebuie să reușească să se ajusteze și la seturi de date mult mai mari.



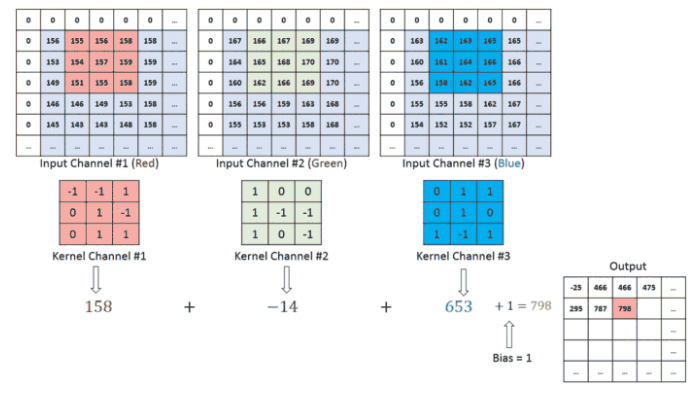
Figură 1.12: Impartirea unei imagini in 3 canale de culoare

**Nucleul (Kernel)**

Nucleul reprezintă o matrice pătratică de dimensiune reglabilă care aplică anumite filtre pe pixelii imaginii. În exemplul de mai jos avem un nucleu de dimensiune 3x3 pixeli care aplica un filtru pe pixelii unei imagini. Un filtru nu este nimic mai mult decât o serie de înmulțiri și adunări între valorile pixelilor imaginii și valorile de înmulțire ale filtrului respectiv. Pe parcursul antrenamentului, valorile acestui filtru sunt cele ajustate, puțin câte puțin, pentru a ajunge la rezultatul dorit.

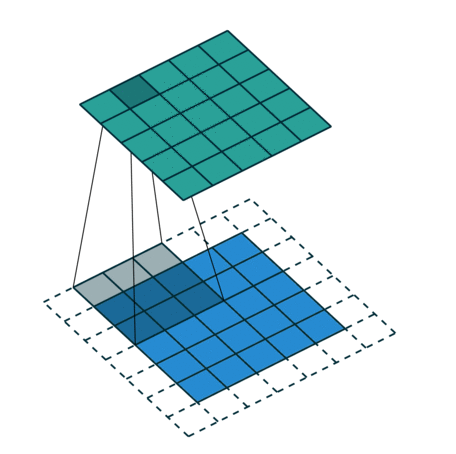
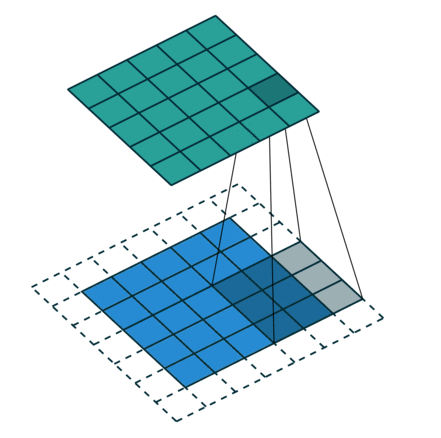


Figură 1.13: Exemplu al aplicării unui filtru

În cazul în care avem de a face cu o imagine color, ceea ce înseamnă 3 canale de culoare diferite, atunci filtrul se va aplica pe fiecare canal, însumând după aceea rezultatele obținute pentru fiecare pixel.

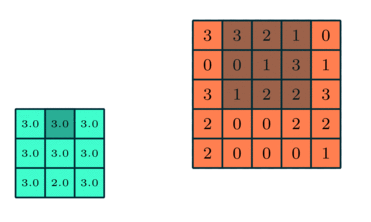
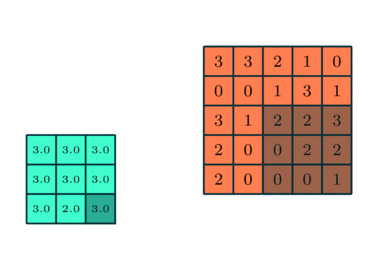
Figură 1.14: Aplicarea unui filtru pe mai multe canale de culoare

În mod normal atunci când aplicăm un filtru, dorim să ne folosim de toate caracteristicile imaginii, inclusiv de cele de pe margine. Atunci vom folosi ceea ce se numește padding: un strat de pixeli care înconjoară imaginea, care au, de obicei, valoarea zero. Folosim asta pentru a putea aplica filtre și pe pixelii aflați în marginea imaginii după cum este ilustrat în exemplul de mai jos.



Figură 1.15: Exemplu padding

În antrenamentul rețelelor convoluționale, valorile ajustate sunt filtrele aplicate în model. Un model poate avea mai multe filtre pe straturi diferite. În general, după fiecare filtru aplicat, se folosește o altă operație, numită pooling. Constă în simplificarea imaginii procesate în urma filtrului pentru a salva din resurse și timp de execuție. De exemplu, o matrice de 5x5 pixeli să fie transformată într-una de 2x2 pixeli, păstrând trăsăturile generale prezent în matricea de bază.



Figură 1.16: exemplu strat convoluțional de pooling

### **Tipuri de machine learning**

* **Supervised learning**:

Supervised learning folosește un set de date de antrenament pentru a antrena modele care să ofere un rezultat dorit. Setul de date include date de intrare și date de ieșire corecte pentru ca modelul să se poată corecta pe parcurs. Algoritmul își măsoară acuratețea folosind funcții de pierdere, ajustând parametrii până când eroarea este minimizată. Modelul menționat mai sus care caută creșteri anormale ale celulelor în radiografii ale pieptului este un exemplu de algoritm care folosește supervised learning. Se împarte în două tipuri de probleme:

* Clasificarea folosește un algoritm pentru a aranja un set de date în mai multe categorii. Este antrenat să caute și să recunoască entități specifice în setul de date, trăgând anumite concluzii despre cum aceste entități ar trebui clasificate sau definite.
* Regresia este folosită pentru a înțelege relația între variabile dependente și independente. Este folosită mult în proiecții de date, cum ar fi venituri din vânzări pentru diferite firme.
* **Reinforcement learning**:

În reinforcement learning este vorba despre luarea anumitor decizii pentru a maximiza recompensa într-o situație particulară. Față de supervised learning, unde setul de date conțin raspunsul cu ajutorul căruia modelul se antrenează, nu există un răspuns corect în reinforcement learning. Agentul trebuie să decidă ce să facă, învățând din experiență, pentru a îndeplinii sarcina care îi este atribuită. Antrenamentul constă în încercarea repetată a agentului de a rezolva o problemă, fiind răsplătit dacă face o alegere buna și pedepsit dacă face o alegere proastă. Modelul va continua întotdeauna să învețe, iar soluția pe care o alege este bazată pe maximizarea răsplății.

* **Unsupervised learning**

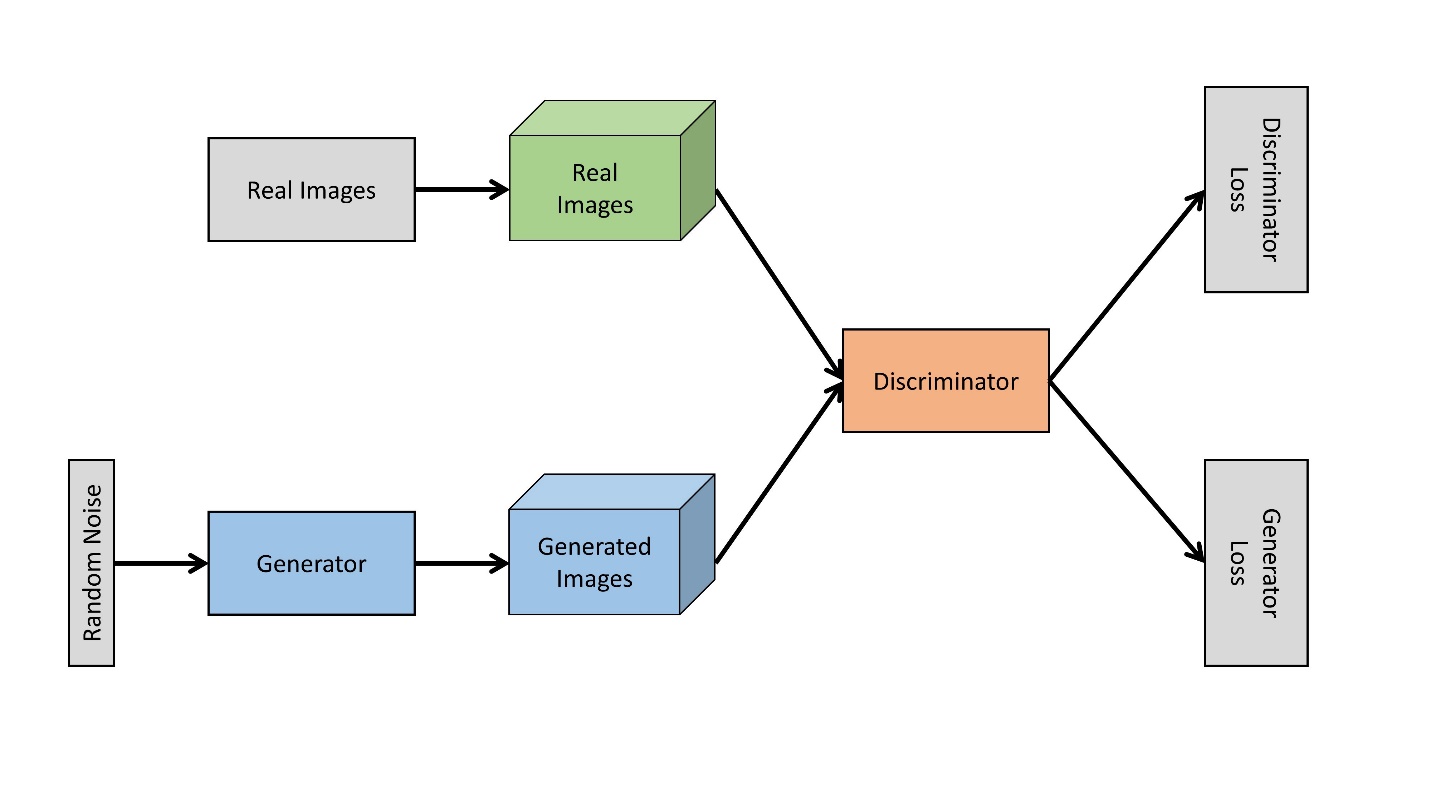
Unsupervised learning permite modelului să lucreze singur cu scopul de a descoperi tipare și informații încă nedetectate. Datele pe care se antrenează sunt neetichetate, de foarte multe ori sarcina fiind de a le grupa, căutând asemănări și tipare.

Proiectul meu de licență este bazat pe acest tip de machine learning. Mai exact pe o subcategorie a acestui tip numită Rețele Generativ Adversariale.

### **Rețele Generativ Adversariale (GAN)**

O rețea generativ adversarială este o rețea complexă formată din 2 modele care se antrenează reciproc: un generator și un discriminator. Ideea de bază a acestei rețele este că generatorul încearcă să creeze imaginile dorite neștiind cum trebuie să arate de fapt imaginea, iar discriminatorul compară imaginea creată de generator cu imaginea dorită și determină care este reală și care este creată de generator. În momentul în care discriminatorul nu mai poate face diferența între cele două imagini, putem spune că rețeaua a încheiat antrenamentul cu succes.

Generatorul pornește fiecare sesiune de antrenament de la un vector de zgomot. Acest vector este inițializat cu date aleatorii pe baza cărora generatorul scoate de fiecare dată imagini diferite. Modelul generatorului este modelul de care avem nevoie după pentru a ne genera imaginile dorite. La început, nefiind antrenat în niciun fel, imaginile generate nu vor avea nicio legătură cu rezultatul dorit, iar eroarea va fii imensă. Odată ce eroare este, puțin câte puțin diminuată, iar filtrele încep să fie ajustate corespunzător, imaginile rezultate vor începe să se aproprie din ce în ce mai mult de rezultat.

Discriminatorul pornește, de asemenea fără nicio informație, antrenându-se odată cu generatorul. Diferența este că eroarea și funcția de backpropagation sunt calculate după un model preexistent și antrenat. Astfel, discriminatorul pornește, de fapt, cu o idee despre ce trebuie să facă și se dezvoltă exact pe sarcina întrebuințată.

Figură 1.17: Arhitectură rețea generativ adversarială

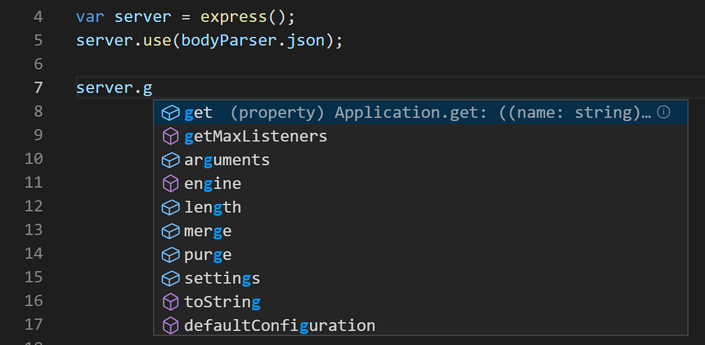
# **2. TEHNOLOGII ȘI INSTRUMENTE SOFTWARE FOLOSITE ÎN IMPLEMENTAREA PROIECTULUI**

În cadrul acestui capitol voi prezenta tehnologiile pe care le-am folosit în realizarea aplicației mele de machine learning. Printre acestea voi scrie despre: Visual Studio Code, Python, PyTorch, HTML, CSS, JavaScript, Flask, Anaconda si altele.

## **2.1. Visual Studio Code**

Visual Studio Code este un editor de cod, „ușor” dar puternic creat de Microsoft, care rulează în desktop și este valabil pentru Windows, macOS și Linux. Vine cu suport încorporat pentru JavaScript, TypeScript și Node.js și are un ecosistem foarte bogat de extensii pentru alte limbaje de programare (de exemplu C++, C#, Python, Java, PHP).

Într-un sondaj din 2019, pe platforma StackOverflow, Peste 50% din 87.317 dezvoltatori au votat că folosesc Visual Studio Code, fiind cel mai popular mediu de dezvoltare.



Figură 2.1 Visual Studio Code

În loc de o structură a proiectului, permite utilizatorilor să deschidă unul sau mai multe directoare, care pot fi apoi salvate într-un spațiu de lucru. Astfel permite scrierea în mai multe limbaje de programare având mai multe ferestre de cod deschise. Fișiere nedorite pot fi excluse din proiect prin intermediul setărilor. De asemenea, utilizatorul își poate personaliza experiența în Visual Studio Code prin instalarea a diferite teme sau extensii care schimbă aparența programului și a codului scris.

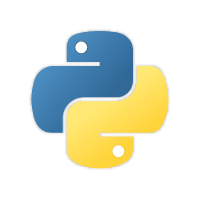
Visual studio code a fost lansat sub o licență MIT în anul 2015 și are codul sursă publicat pe GitHub.

## **2.2. Python**

Python este un limbaj de programare interpretat, orientat pe obiecte și de nivel înalt folosind semantică dinamică. Structurile de date predefinite și dezvoltarea dinamică de cod îl fac să fie o alegere foarte bună pentru crearea rapidă de aplicații. Sintaxa ușor de învățat oferă o ușurință în citire și reduce costul și nevoia de mentenanță a unui program. Python suportă module și pachete, lucru care încurajează modularitatea programului și reutilizarea codului.

Filozofia care stă la baza limbajului este rezumată în documentul „The Zen of Python”, care include aforisme cum ar fi:

* Frumos este mai bine decât urât
* Explicit este mai bine decât implicit
* Simplu este mai bine decât complex
* Complex este mai bine decât complicat
* Lizibilitatea contează



Figură 2.2 Python logo

În locul de a avea o funcționalitate la bază, Python a fost proiectat să fie extrem de extensibil folosind module. Modularitatea compactă l-a făcut foarte popular ca și unealtă pentru a adăuga interfețe programabile la aplicații deja existente.

Python se străduiește să ofere o sintaxă mai simplă, mai puțin dezordonată în timp ce programatorii au posibilitatea de ași alege singuri metodologia de codare. Limbajul de programare îmbrățișează filozofia „ar trebui să existe o – și preferabil o singură – cale evidentă de rezolvare”. Alex Martelli, angajat al fundației software de Python și autor al cărții Python scrie că: „Să descrii ceva ca fiind ‚inteligent’ nu este un compliment în cultura Python”.

Am folosit Python ca și limbaj principal de dezvoltare, întrucât tot algoritmul de machine learning este scris în Python. De asemenea partea de back-end a interfeței web este creată folosind Python.

## **2.3. PyTorch**

PyTorch este o biblioteca de machine learning, open source, bazată pe biblioteca Torch, folosită pentru diferite aplicații, cum ar fi viziune computerizată și procesarea limbajelor naturale. A fost inițial dezvoltată de laboratorul AI de cercetare Facebook și este gratis și open source. Deși interfața Python este mult mai îngrijită și obiectivul principal al dezvoltării, PyTorch are și o interfață pentru C++.

Multe proiecte de deep learning au fost dezvoltate folosind PyTorch, cum ar fi Tesla Autopilot și Pyro de la Uber.

PyTorch folosește o clasă numită Tensor pentru a stoca și a efectua operații pe matrici multidimensionale și omogene. Tensorii sunt similari cu matricile din NumPy, însă pot fi procesate și pe plăcile video Nvidia care suportă tehnologia CUDA.

## **2.4. NumPy**

NumPy este o bibliotecă pentru Python care adaugă suport pentru matrici și vectori imenși și multidimensionali, împreună cu o mare colecție de funcții matematice de nivel înalt pentru a opera pe aceste matrici.

Limbajul Python nu a fost inițial proiectat pentru computare numerică, însă a atras atenția oamenilor de știință și a inginerilor. În 1995 un grup a fost fondat cu scopul de a definii pachetul de computare al limbajului de programare și a extinde sintaxa pentru a-l face mult mai ușor de folosit. Mai târziu, un nou pachet, numit Numarray a fost dezvoltat ca o înlocuire mai flexibilă pentru Numeric. Acesta avea operațiuni mai rapide pentru vectori mari, însă operațiuni mai lente pentru vectori mici, astfel cele două erau folosite în paralel. În 2005, s-a dorit unificarea comunității în jurul unui singur pachet. Astfel prin combinarea celor două s-a ajuns la NumPy.

NumPy țintește implementarea referinței de CPython care este un interpretor. Folosind NumPy, se obține o funcționalitate comparabilă cu Matlab din moment ce ambele sunt interpretate. Ambele permit utilizatorilor să dezvolte programe rapide atât timp cât majoritatea operațiilor lucrează cu matrici, nu cu scalari.

## **2.4. Pandas**

Pandas este o bibliotecă software scrisă pentru Python pentru manipularea de date și analiza acestora. Oferă structuri de date și operațiuni pentru manipularea tabelelor numerice și seriilor de timp. Este un program gratuit. Numele derivă din termenul „panel data”, un termen folosit pentru seturi de date care include observații ale unor date individuale în mai multe ipostaze de timp.

Din moment ce, în machine learning se lucrează cu seturi mari de date, pandas este cea mai utilizată bibliotecă pentru curățarea și analizarea datelor, cât și modelarea acestora pentru a le aduce în forma dorită.

Oferă multe funcții esențiale pentru manipularea seturilor mari de date cum ar fi modurile în care are de-a face cu datele lipsă, aplicând transformări pe setul de date sau creând mulțimi separate pentru citire. Lucrează împreună cu biblioteca matplotlib pentru afișarea și vizualizarea datelor.

## **2.4. Matplotlib**

Matplotlib este o bibliotecă folosită în vizualizarea datelor, dezvoltată pentru Python. Oferă un API orientat pe obiecte pentru integrarea figurilor și grafurilor în aplicații. Există, de asemenea, și o interfață procedurală, dezvoltată să se asemene cu Matlab, deși folosirea acesteia este destul de descurajată.

## **2.4. Anaconda**

Aanconda este o distribuție de Python folosită pentru computare științifică în aplicații de data science și machine learning, care caută să simplifice administrarea și implementarea programelor.

Distribuția de Anaconda vine cu peste 250 de pachete deja preinstalate și peste 7500 pachete adiționale, open-source, care pot fi instalate. Se pot crea medii de lucru virtuale separate pentru fiecare proiect folosind funcțiile preinstalate. Marea diferență între conda și pip pentru instalarea pachetelor în Python este modul în care dependențele sunt administrate. Lucru care prezintă o provocare mare pentru Data science în Python și este motivul pentru care conda există.

## **2.4. HTML**

HTML (HyperText Markup Language) este standardul folosit pentru documentele create cu rolul de a fi afișate într-un browser web. Un browser web primește documente HTML de la un server sau din spații de stocare locală și interpretează documentele în pagini multimedia. HTML descrie structura, sau scheletul, unui site web într-un mod semantic.

Folosind elementele de HTML, imaginile și alte obiecte cum ar fi forme interactive pot fi integrate în pagini. Oferă posibilitatea de a crea documente structurate prin folosirea unor semantici structurale pentru text cum ar fi paragrafe, titluri, liste, linkuri, și altele. Elementele HTML sunt delimitate de taguri.

HTML poate fi asistat de alte tehnologii cum ar fi CSS și JavaScript pentru crearea de pagini.

## **2.4. CSS**

Cascading Style Sheets (CSS) este un limbaj folosit pentru descrierea prezentării unui document scris într-un limbaj de marcare cum ar fi HTML. Este proiectat pentru a permite separarea dintre prezentare și conținut folosind aspecte, culori, fonduri. Această separare poate îmbunătății accesibilitatea conținutului, oferi mai multa flexibilitate și control în specificarea caracteristicilor prezentației. Permite mai multor pagini web să împartă formatare și informații făcând referință la același fișier css. Lucru care reduce, evident, complexitatea și repetiția în structura conținutului.

Separarea și formatarea conținutului fac, de asemenea, posibil să se prezinte stiluri diferite pentru aceeași pagină de marcaj. Numele „cascading” provine din schema de prioritate care determină care regulă de stil de aplică în cazul în care mai multe reguli se potrivesc la un singur element.

## **2.4. JavaScript**

JavaScript este un limbaj de programare de nivel înalt, compilat dinamic, în timp ce rulează. Folosește sintaxă cu acolade, scriere dinamică, orientare pe obiecte bazată pe prototipuri și funcții de clasa întâi.

Împreună cu HTML și CSS, JavaScript este una din tehnologiile de baza ale internetului. Peste 97% din site-uri îl folosesc în partea clientului pentru comportamentul paginii, de obicei încorporate de pachete adiționale. Toate motoarele de căutare web majore au un motor dedicat de JavaScript inclus pentru a executa codul pe dispozitivul utilizatorului.

Motoarele de JavaScript erau original folosite numai în browserele web, însă acum prezintă o componentă de bază pentru alte sisteme software, în mod special servere și o varietate de aplicații.

Peste 80% din site-uri folosesc o bibliotecă terțiară sau un framework JavaScript pentru codarea de pe partea clientului. JQuery este de departe cel mai popular, folosit în peste 75% din site-uri. Facebook a creat biblioteca React pentru website-urile sale, lansând-o apoi ca o resursă open source. Este acum folosită de multe site-uri incluzând Twitter. De asemenea, framework-ul Angular, creat de Google pentru site-urile sale, incluzând Youtube și Gmail, este acum un proiect open source folosit de alții.

## **2.4. Flask**

Flask este un micro web framework scris in Python. Este clasificat ca și un microframework deoarece nu are nevoie de unelte sau biblioteci particulare. Nu are niciun strat de bază de date, validare sau orice altă componentă unde vreo biblioteca terțiară oferă funcții comune. Totuși Flask suportă extensii care pot adăuga caracteristici ca și cum ar fi implementate direct în Flask.

Flask este interconectat în două părți majore:

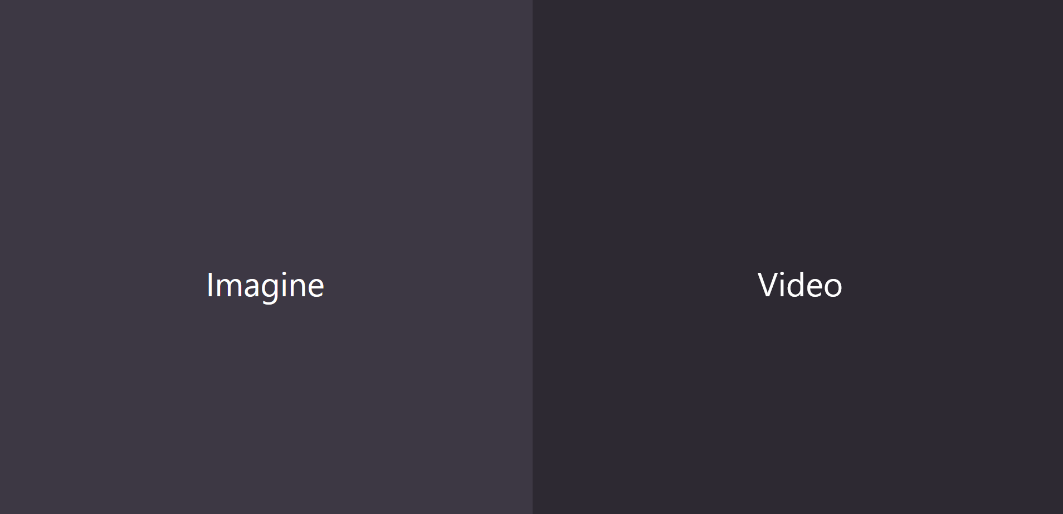
* Werkzeug este o bibliotecă utilitară menită pentru folosința cu Python. În mare parte este un gateway către un server care poate crea articole software pentru cereri, răspunsuri și funcții utile.
* Jinja este un motor șablon pentru programarea în Python și se aseamănă cu șblonul din Django.

Aplicații care folosesc Flask includ Pinterest și Linkedin.

# **STUDIU DE CAZ „SISTEM EXPERT PENTRU OPTIMIZAREA FIȘIERELOR MEDIA”**

Proiectul meu este împărțit în două rețele neuronale diferite: una dublează rezoluția imaginilor, făcându-le mai clare, cealaltă aplică interpolarea cadrelor pe videoclipuri și le face mai fluide.

Utilizatorul este întâmpinat cu o aplicație web în care poate alege dacă dorește să încarce o imagine sau un fișier video.



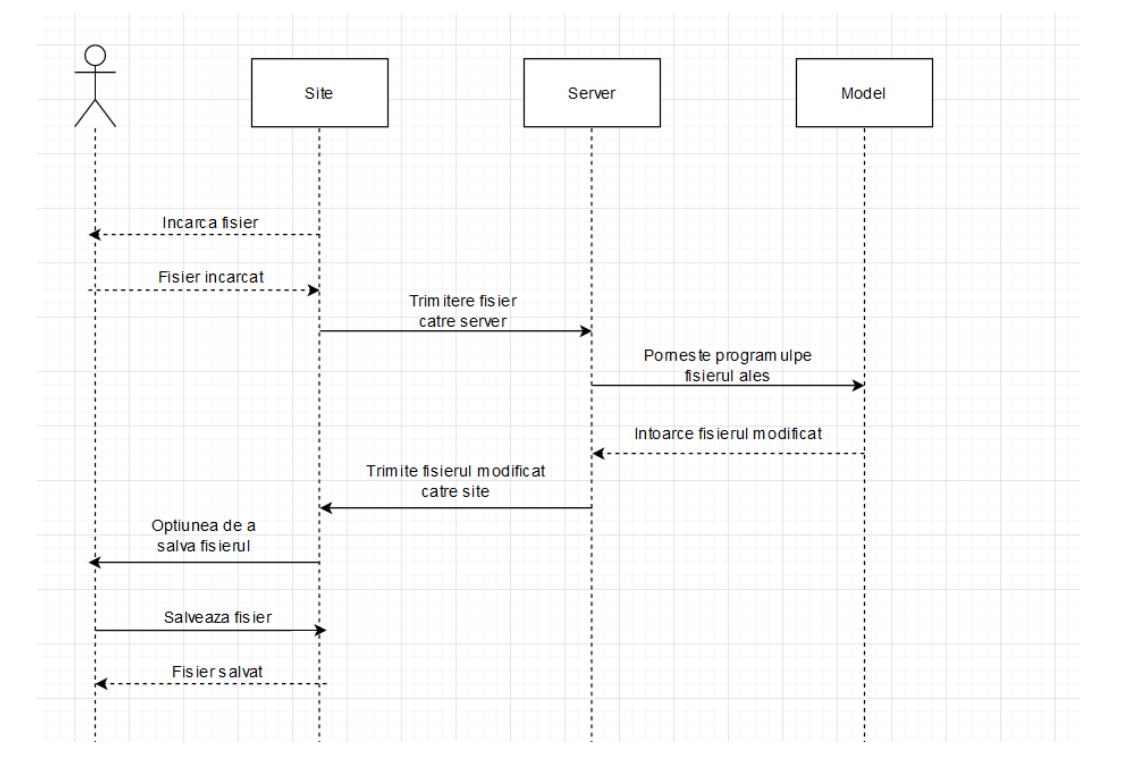
Figură 3.1 Interfața web

Selectând una dintre cele două va oferi posibilitatea de a încărca fișierul dorit pentru optimizare. În cazul în care încărcăm o imagine, va acționa prima rețea neuronală, altfel va acționa cea de-a doua rețea neuronală.



Figură 3.2 Interfața web

În urma încărcării unui fișier, API-ul va prelua imaginea sau videoclipul către server și îl va trece prin rețeaua neuronală aleasă, întorcând înapoi fișierul optimizat pentru descărcare.

Aplicația web este dezvoltată folosind HTML, CSS și JavaScript pentru front-end, iar pentru back-end am folosit Flask.

Figură Diagrama secventiala

Diagrama secvențială descrie pașii prin care poate trece utilizatorul în interacțiunea cu aplicația.

Modelul rulează pe server, astfel că, rularea algoritmului, nu depinde de loc de performanța dispozitivului de pe care accesează utilizatorul site-ul. Acest lucru este un mare plus pentru consumatorul obișnuit, care accesează site-ul, probabil, de pe un laptop sau pentru persoanele care nu dispun de o placă video dedicată care să suporte tehnologia CUDA.

De asemenea o mare problemă rezultată din rularea pe server poate fi întâmpinată atunci când un număr mare de utilizatori vor să acceseze aplicația web în același timp. Este posibil ca serverul să nu fie capabil să ducă așa de multe instanțe ale aplicației în același timp. O posibilă rezolvare a acestei probleme poate fi implementarea unei cozi de așteptare, însă serviciile atunci nu ar mai fi de calitate dacă utilizatorul trebuie să aștepte. Astfel, pentru rularea cât mai optimă a aplicației pentru un număr mare de utilizatori, este nevoie de niște resurse generoase din partea serverului, sau de rularea, cel puțin parțială pe partea utilizatorului.

## **Prima rețea neuronală: Optimizarea rezoluției imaginilor**

Pentru această rețea cât și pentru cealaltă am implementat un tip de rețea generativ adversarială, unde avem un generator și un discriminator care se antrenează împreună pentru a atinge scopul dorit. În cazul de față, scopul a fost optimizarea dimensiunii unei imagini.

### **Pregătirea setului de date**

Primul pas în machine learning este citirea datelor și modelarea lor astfel încât să poată fi introduse toate în program în timpul antrenamentului. Pentru antrenarea acestui model a fost folosit setul de date CelebA, având 60.000 de imagini diferite cu fețe ale celebrităților. Pentru test, am folosit la finalul fiecărei epoci de antrenament câteva imagini pe care le-am trecut prin rețeaua neuronală pentru a observa evoluția în timp a modelului.



Figură 3.3 Set de date CelebA

La momentul de față modelul acceptă numai imagini care au terminația .jpg, .jpeg, sau .png. În evoluția proiectului pe viitor se va avea în evidență adăugarea a mai multor permisiuni pentru tipul de imagine.

Fiecare imagine trebuie transformată pentru a avea toate același format. Imaginile sunt redimensionate și transformate în tensori pentru a permite folosirea lor mai departe în PyTorch. Pentru fiecare imagine vom avea o variantă redimensionată și blurată care va fi folosită ca și dată de intrare. Acesta va juca rolul termenului de comparație între el și rezultatul generatorului. Comparația este treaba discriminatorului care va întoarce un răspuns apoi către generator.

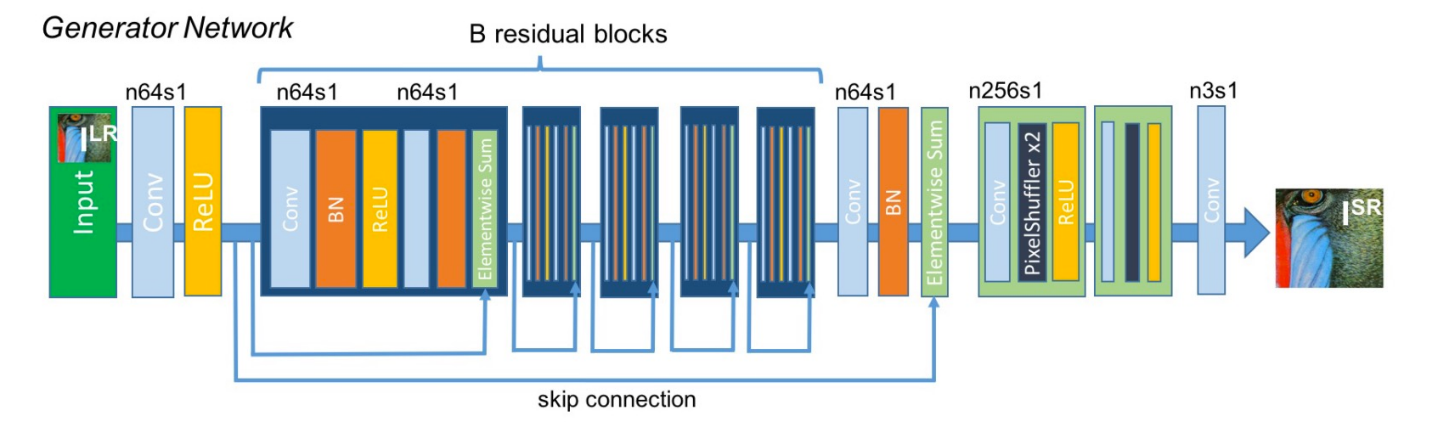
### **Definirea funcțiilor de pierdere**

Funcție de pierdere sau criterion în PyTorch reprezintă o secvență de formule matematice după care este calculată eroarea antrenamentului respectiv. În general, pentru majoritatea problemelor, există multe funcții predefinite despre care am vorbit mai sus. În cazul acestei rețele, avem o funcție diferită pentru generator și una pentru discriminator.

Eroarea discriminatorului este calculată în funcție de diferența dintre imaginea rezultată de generator și imaginea reală cu care o comparăm.

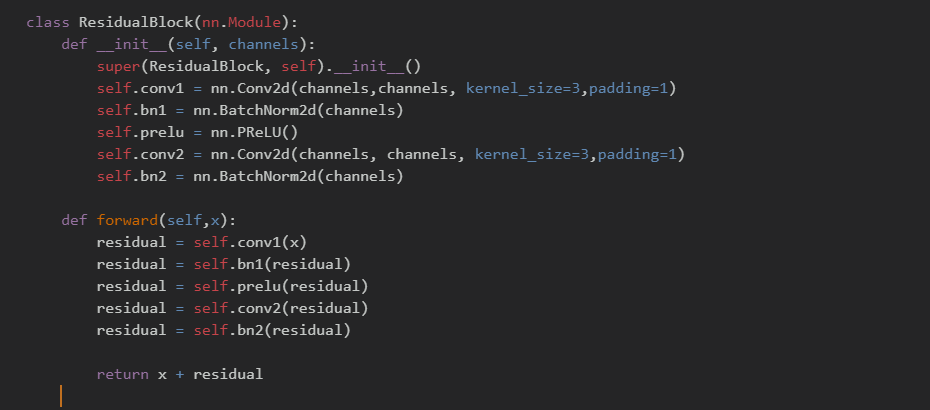
Eroarea generatorului este o combinație între mai multe erori calculate. La una din ele se folosește un model predefinit numai pentru calculul erorii. Rezultatul este apoi întors pentru executarea algoritmului de backpropagation.

### **Modelul generatorului**



Figură 3.4 Ilustrare Model generator

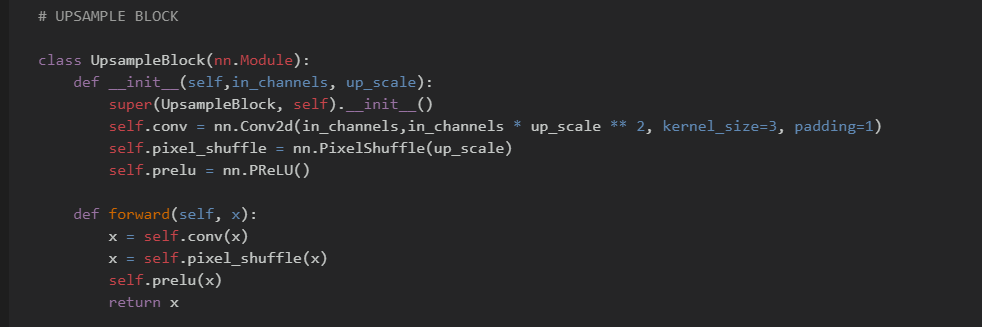
Modelul generatorului este compus din mai multe grupuri reziduale de straturi convoluționale, urmate de 2 straturi de upsample. Prima dată intrată în model, imaginea este trecută printr-un strat convoluțional, apoi printr-o funcție de activare. Mai departe, trece prin 5 grupuri reziduale de straturi convoluționale. Un singur grup rezidual este format din două straturi convoluționale, două normalizări de date și două funcții de activare, urmate de o însumare a elementelor calculate în grupul respectiv.



Figură 3.5 Cod, Grupul rezidual

Cele cinci grupuri reziduale sunt apoi urmate de jumătate de grup rezidual, unde avem un singur strat convoluțional, o singură normalizare de date și o singură funcție de activare.

Este apoi urmat de cele două grupuri de upsample compuse dintr-un strat convoluțional, un strat numit PixelShuffle care caută să mărească imaginea adăugând pixeli adiționali și finalizat cu încă o funcție de activare.



Figură 3.6 Cod, grupul de upsample

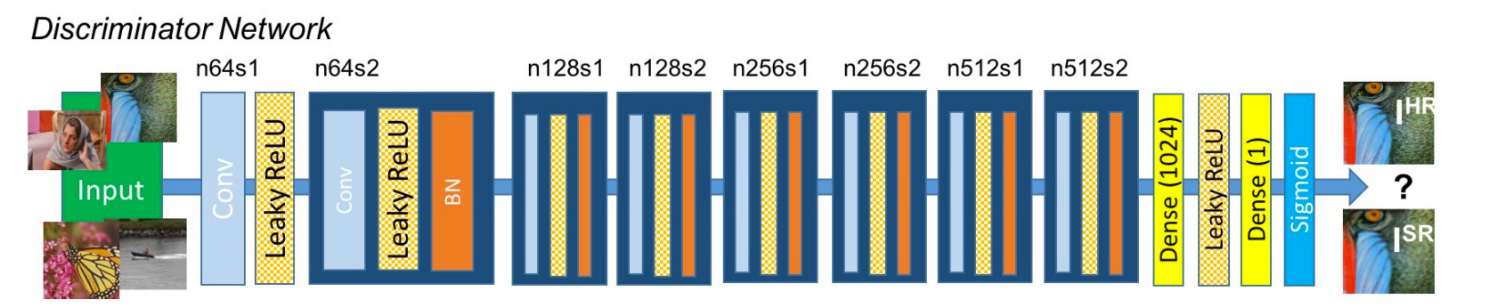


Figură 3.7 Cod, Clasa Generatorului

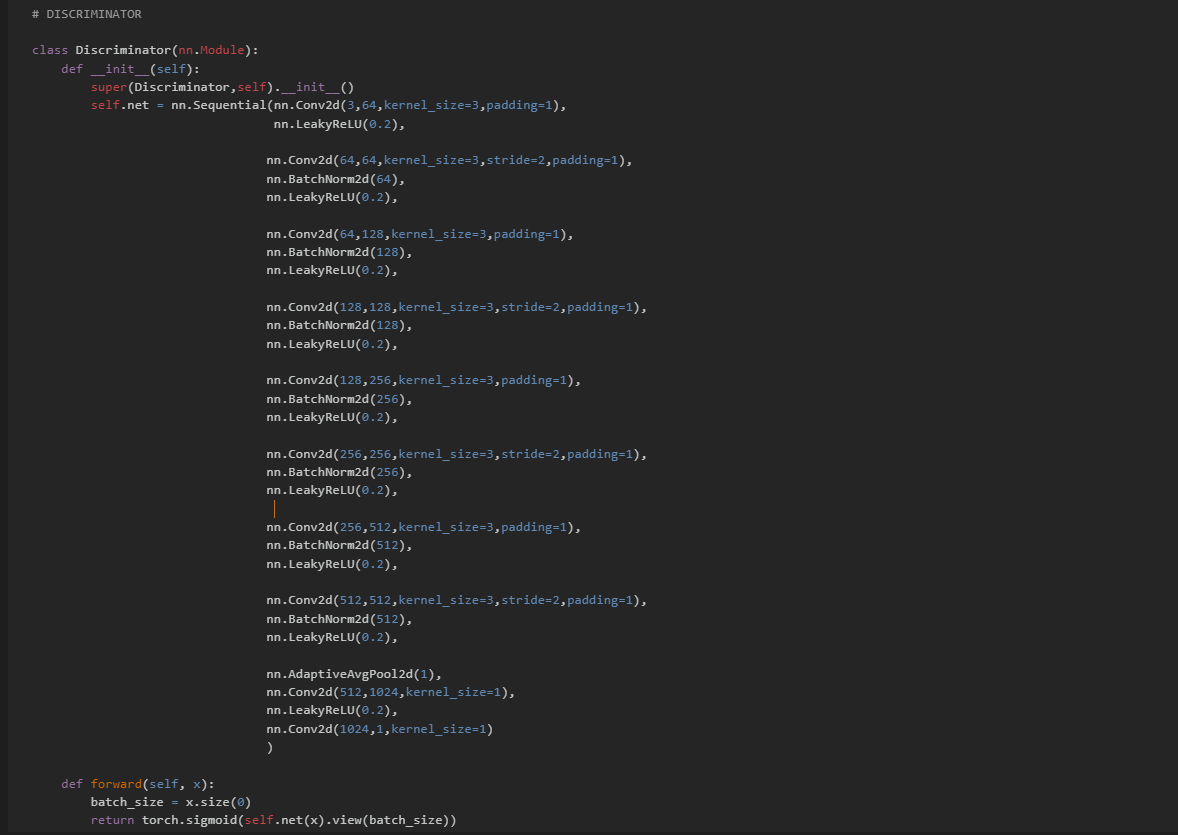
Puse toate cap la cap, Funcția care definește clasa generatorului arată astfel

Toate acestea însumează un total de 586.506 parametri pentru modelul generatorului.

### **Modelul discriminatorului**



Figură 3.8 Ilustrare model discriminator

Rețeaua discriminatorului începe cu un strat convoluțional, urmat de o funcție LeakyReLU de activare. Apoi este o secvență de 7 grupuri, fiecare conținând un strat convoluțional, o funcție de activare și un strat de normalizare a datelor. Cu fiecare 2 grupuri de dublează numărul de parametrii de intrare, începând de la 64 și ajungându-se la 512. Este apoi trecută într-un strat conectat și trecută prin funcția de sigmoid.

Figură 3.9 Cod, Clasa Generatorului

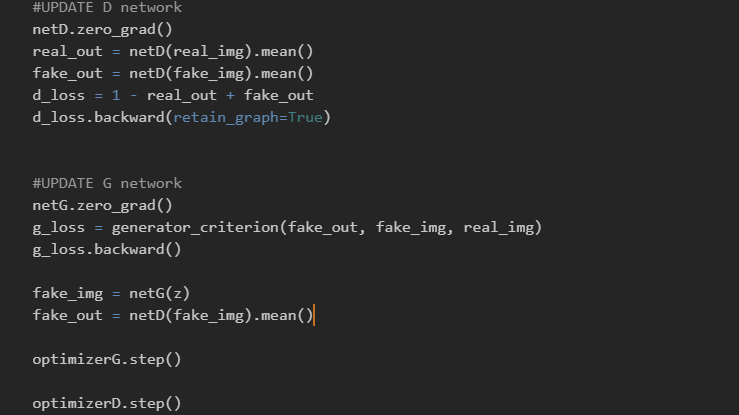
Toate acestea oferă un total de 5.215.425 de parametrii pentru modelul discriminatorului.

### **Antrenamentul**

Pentru fiecare epocă de antrenament sunt folosite toate cele 60.000 de imagini din setul de date CelebA. Limitările plăcii mele video mi-au permis să le iau în grupuri de câte 20 de imagini deodată cu o singură imagine de test. La fiecare epocă diferită, datele sunt amestecate, astfel că nu sunt luate niciodată imaginile în aceeași ordine.

O epocă de antrenament constă în împărțirea imaginilor în grupuri de câte 20, apoi trecând fiecare grup, imagine cu imagine, prin modelul de generator. Apoi fiecare imagine rezultată din generator este trecută împreună cu imaginea corespondentă în rețeaua discriminatorului pentru ca acesta să își dea verdictul.

Pierderea este calculată pentru fiecare imagine în parte, însă algoritmul de backpropagation se folosește o singură dată pe fiecare grup de câte 20 de poze în felul următor: după ce se calculează funcția de pierdere la fiecare imagine, se face media aritmetică între ele și greutățile cu bias-urile sunt ajustate în funcție de acea medie aritmetică.

După ce facem algoritmul de backpropagation pentru rețeaua discriminatorului, urmează și rețeaua generatorului.

Figură 3.10 Cod, backpropagation pentru cele doua modele

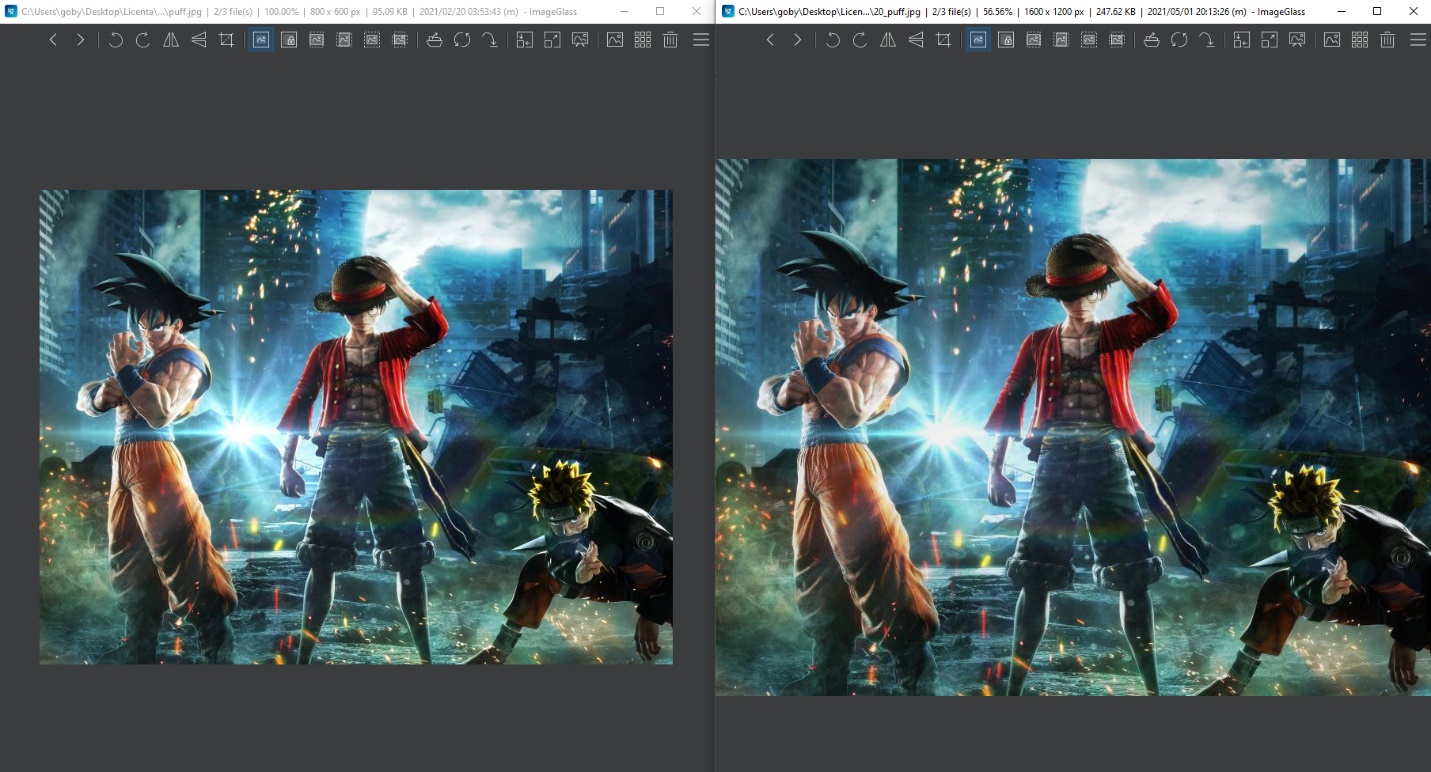
Acest procedeu se repetă pentru fiecare grup de 20 de imagini din cel 60.000 de poze. Cu alte cuvinte, se va repeta de 3.000 de ori pentru fiecare epocă. În medie, pe placa mea video GTX1060 3GB o epocă dura undeva la 20-30 de minute. Pentru un rezultat eficient, cartea din care m-am documentat pentru modelul acesta recomanda un antrenament de 10.000 – 20.000 de epoci. Lucru care ar fi durat probabil o lună în care nu aș fi închis calculatorul. Așa că am ținut antrenamentul să meargă câteva săptămâni, cu întreruperi, ajungând la aproximativ 4.000 de epoci de antrenament complete.

La finalul fiecărei epoci am afișat câteva date ajutătoare, incluzând pierderea fiecărui model și o imagine de test pentru a observa evoluția modelului în timp.

### **Rezultate**

După 4.000 de epoci de antrenament completate, rețeaua este în stare să dubleze rezoluția unei imagini destul de bine cât să îi mărească calitatea.

Scriptul de execuție constă numai în inserarea imaginii dorite în rețeaua de generator și luând rezultatul, fără a ne mai face griji de eroare sau pierdere sau de backpropagation.

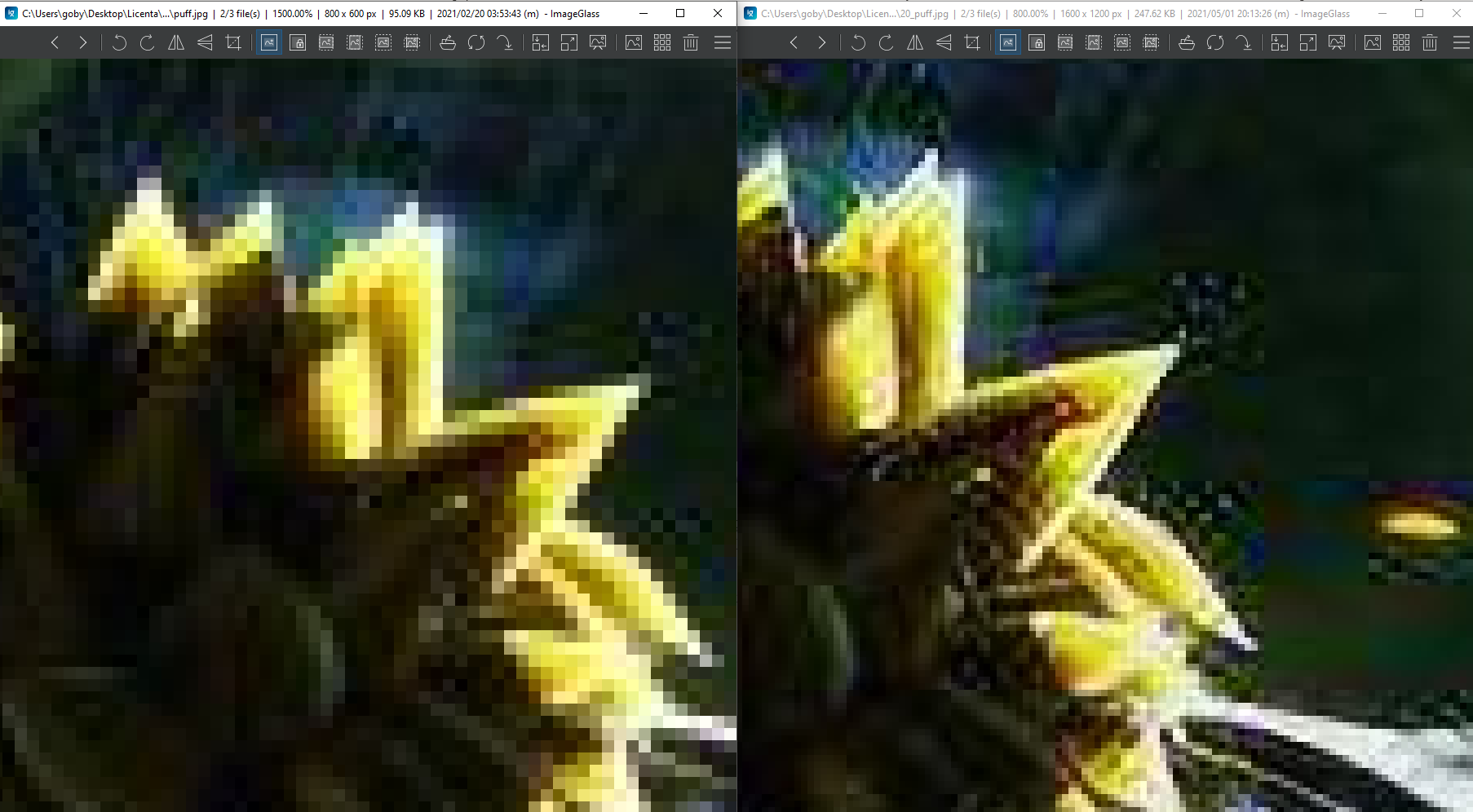
Exemplu poză înainte și după ce a trecut prin rețeaua neuronală:

Figură Exemplu imagine, inainte și după să fie trecută prin rețeaua neuronală a)

Rezoluția imaginii inițiale este de 800x600 pixeli pe când cea rezultată după ce a trecut prin generator este de 1600x1200 pixeli. Momentan aceasta este limita plăcii mele video, însă conversia a avut loc cu succes.

Rezultatele se pot observa foarte bine dacă încercăm să mărim mai mult imaginea și să observăm pixelii.

Se poate observa că în imaginea din dreapta calitatea este mai bună, părul se vede mult mai bine definit, iar detaliile sunt mult mai „ascuțite” și mai ușor de distins.



Figură Exemplu imagine înainte și după să fie trecută prin rețeaua neuronală b)

## **A doua rețea neuronală: Interpolarea fișierelor video**

A doua rețea este tot un model de rețea generativ adversarială cu un generator și un discriminator. Scopul este de interpolare a videoclipurilor pentru a le face mai fluide. Este capabil să genereze cadre lipsă pe parcursul unui videoclip cât și să genereze cadre intermediare între două imagini, creând un mini videoclip.

### **Introducere**

Interpolarea cadrelor video (VFI) caută să sintetizeze cadre intermediare între două cadre consecutive ale unui videoclip și este foarte mult folosită pentru creșterea fluidității și a calității unui videoclip. Este capabilă, de asemenea, de o mulțime de aplicații cum ar fi slow-motion, compresie video. Pe deasupra, algoritmii VFI care rulează în timp real pe videoclipuri cu rezoluții mari au un potențial foarte mare pentru aplicații, cum ar fi creșterea cadrelor într-un joc video sau videoclipuri live, furnizarea serviciilor de editare a videoclipurilor mai avansate pentru persoanele cu resurse computaționale limitate.

VFI este foarte provocator din cauza mișcărilor complexe și non liniare cât și a schimbărilor de iluminare în lumea reală. În ultimul timp, algoritmi VFI bazați pe flux au oferit un cadru de dezvoltare pentru a adresa aceste provocări și a obține rezultate impresionante. Abordările comune pentru aceste metode implică doi pași:

* Deformarea cadrelor de intrare în concordanță cu fluxul optic aproximativ
* Fuziunea și rafinarea cadrelor deformate folosind rețele neuronale de convoluție

După modul în care sunt deformate cadrele, algoritmii VFI pot fi clasificați în deformare înainte și deformare înapoi. Deformarea înapoi este mult mai folosite din cauza conflictelor de care suferă deformarea înainte atunci când multe surse de pixeli sunt atașați unei singure locații. Primind cadrele de intrare I0, I1, metodele de deformare înapoi trebuie să aproximeze fluxul intermediar din perspectiva cadrului intermediar pe care dorim să îl sintetizăm. Practici comune calculează, prima data, fluxul bidirecțional folosind modele deja antrenate, apoi le inversează și rafinează pentru a genera fluxuri intermediare. Totuși, aceste metode au defecte când vine vorba de obiecte care își schimbă poziția din cadru în cadru. Algoritmii VFI precedenți prezintă două probleme majore:

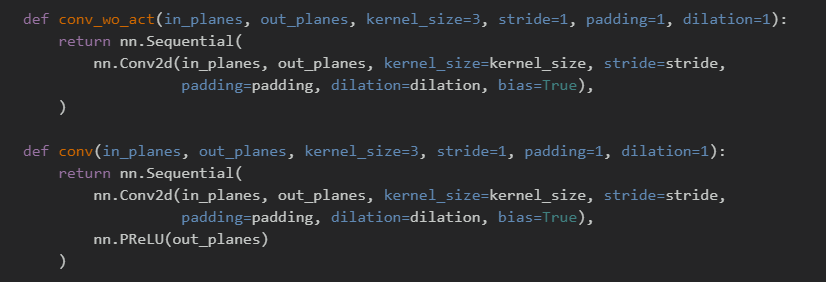
* Nevoia de componente adiționale: adâncimea imaginii, rafinarea fluxului și inversarea fluxului sunt straturi de convoluție introduse pentru a compensa pentru defectele inversării fluxului. Aceste operații necesită multe resurse de computare și nu sunt potrivite pentru scenarii în timp real.
* Lipsa supravegherii directe pentru fluxurile intermediare. Tot sistemul de interpolare este, de obicei, antrenat folosind numai pierderea finală. Nu există nicio altă supraveghere creată pentru estimarea procesului de flux sau degradarea performanței de interpolare.

### **Modelul IFNet**

Modelul de flux intermediar IFNet a fost creat special pentru a estima fluxul intermediar. IFNet adoptă o strategie cu rezoluții progresiv crescătoare: actualizează iterativ un câmp de flux prin grupuri succesive. Teoretic, în conformitate cu câmpurile de flux actualizate, putem muta pixelii corespondenți din două cadre de intrare în aceeași locație într-un cadru intermediar.

Folosirea unei supravegheri puternice în fluxul intermediar este, de asemenea foarte important. În antrenamentul rețelei IFNet, folosirea ulterioară a unor procese de fuziune după pierderea finală, modelul a produs rezultate mult mai slabe decât metodele precedente. Totul s-a schimbat dramatic după ce a fost adăugată supravegherea intensă pe modelul IFNet. O funcție de pierdere a fost concepută care are acces complet asupra cadrelor intermediare în timpul antrenamentului.

Combinând aceste design-uri, a luat naștere RIFE (Real-time Intermediate Flow Estimation). RIFE poate obține rezultate foarte satisfăcătoare atunci când este antrenat de la zero.



Figură Cod rețea IFNet b)

Figură Cod rețea IFNet a)

Schimbările majore ale acestui model față de cele precedente sunt:

Figură Cod rețea IFNet c)

* Proiectarea modelului eficient IFNet pentru a simplifica metodele VFI. IFNet poate fi antrenat de la zero și poate aproxima direct fluxul intermediar între două cadre de intrare.
* Adăugarea supravegherii efective asupra estimării fluxului intermediar, în special funcția de pierdere, care duce către o folosire mult mai stabilă și o creștere semnificantă în performanță.

### **Fluxul Optic**

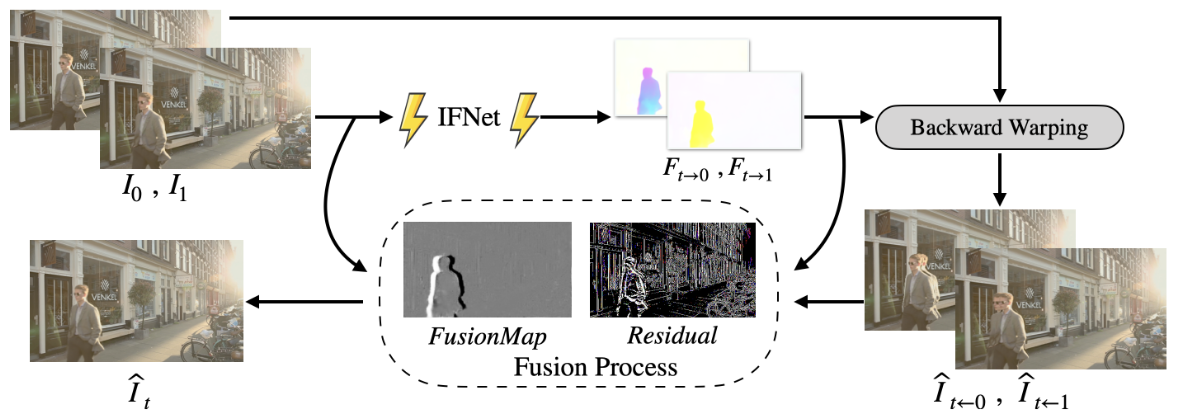
Estimarea fluxului optic este o problemă de vizualizare foarte veche care caută să estimeze mișcarea pe fiecare pixel, lucru extrem de folositor în foarte multe sarcini. De când a fost setat ca și referință munca adusă de FlowNet, bazată pe un codificator automat, arhitecturile pentru modelele de flux optic au evoluat pe parcursul ultimilor ani, oferind rezultate din ce în ce mai exacte, fiind, în același timp, mult mai eficiente. Aceste metode adoptă, de obicei, o abordare de rafinărie iterativă și care implică mulți operatori cum ar fi, costul volumului, caracteristici piramidale și deformare înapoi. Altă direcție de cercetare foarte importantă o reprezintă estimarea fluxului optic folosind metode nesupravegheate din cauza dificultății de etichetare a fluxului optic în contextul videoclipului.

### **Evoluția interpolării cadrelor video**

Liu a propus o rețea complet convoluțională pentru estimarea fluxului și generarea cadrelor intermediare prin eșantionare. Jiang a propus o combinație liniară ale celor două fluxuri bidirecționale ca o aproximare inițială a fluxului intermediar. Apoi rafinarea lor și prezicerea hărții de vizibilitate care codifica informațiile întâlnite. Bazat pe ultima lucrare, Reda a propus sintetizarea cadrelor intermediare folosind cicluri nesupravegheate consistente. Bao a propus modelul DAIN care presupune folosirea unui strat de proiecție a fluxului conștient de adâncimea imaginilor pentru a estima fluxul intermediar ca fiind o combinație între fluxurile bidirecționale. Nikolaus a propus un algoritm de deformare înainte a cadrelor și a hărții de adâncime. Xu a propus exploatarea a patru cadre consecutive și un filtru de inversare a fluxului pentru a calcula cadrele intermediare. Liu a extins această idee folosind o predicție rectificat pătratică a fluxului. Dintre toate aceste metode, DAIN a fost folosită în foarte multe aplicații software.

Împreună cu aceste metode bazate pe flux, au fost create cu succes și metode care nu se leagă de fluxul videoclipului pentru a atinge progrese remarcabile în ultimii ani.

### **Modelul RIFE**

Funcționalitatea modelului RIFE este ilustrată în imaginea următoare. Primind o pereche de cadre consecutive I0, I1, scopul este de a sintetiza cadrul intermediar It la timpul t = 0.5. Se estimează direct fluxul între cele două cadre de intrare și cadrul intermediar prin inserarea celor două cadre de intrare în modelul IFNet. Apoi putem primii două rezultate, exprimate prin cele două fluxuri bidirecționale prin deformarea înapoi a cadrelor. Pentru a înlătura artefactele nedorite create în urma deformării, inserăm cadrele de început, fluxurile aproximate și cadrele intermediare în procesul de fuziune prin intermediul unui codificator – decodificator pentru a genera rezultatul final.

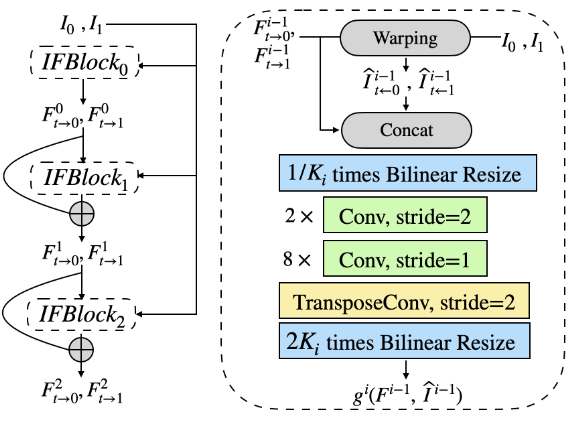
Figură Ilustrarea funcționalității modelului RIFE

RIFE are două componente majore:

* Estimarea fluxului intermediar foarte eficientă prin rețeaua IFNet
* Procesul de fuziune al cadrelor deformate

#### **Estimarea fluxului intermediar**

Unele metode precedente de estimare a fluxului inversează și rafinează fluxul bidirecțional. Rolul modelului IFNet este de a prezice direct și eficient fluxul între cadrele de intrare și cadrul intermediar pe care dorim să îl obținem.

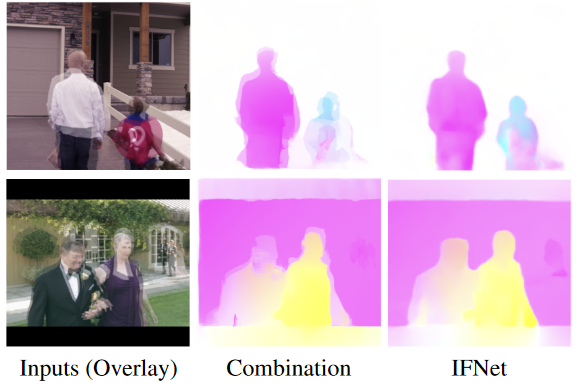
Pentru a putea lucra cu mișcări mari întâmpinate în estimările de flux intermediar, a fost creată o strategie care se folosește de rezoluții crescătoare cum este prezentat în figura de mai jos.

Figură Structura modelului IFNet

În mod special, prima dată este calculată o prezicere frustă asupra fluxului pe o rezoluție mai mică, lucru care pare să captureze mișcările mari mult mai ușor si mai eficient. Apoi câmpurile de flux sunt rafinate gradual folosind rezoluții din ce în ce mai mari. Urmând design-ul acesta, IFNet are o structură de clepsidră unde un câmp de flux este rafinat interativ prin grupuri succesive ale rețelei care operează pe rezoluții crescătoare.



Figură Formula estimării fluxului intermediar

Unde F^i-1 denotă estimarea curentă a fluxurilor intermediare de la grupul i-1, iar ultimul termen denotă cadrele de intrare deformate folosind fluxul aproximat precedent. Sunt folosite un total de 3 grupuri, unde fiecare are un parametru de rezoluție. Pentru a păstra design-ul simplist fiecare grup are o structură care este formată din mai multe straturi de convoluție și un operator de supraeșantionare. Exceptând stratul de ieșire, rezidurile lăsate de fluxul optic, harta de fuziune și rezidurile de după reconstruire, este folosită funcția de activare PReLU.

Figură Comparație vizuală între fluxul bidirecțional generat de o rețea pre-antrenată și fluxul generat de rețeaua IFNet

În figura de mai sus este reprezentat rezultatul vizual al modelului IFNet și comparația cu fluxul optic bidirecțional combinat, generat de o rețea precedentă pre-antrenată. IFNet reușește să producă granițe de mișcare bine definite și clare, unde modelul precedent întâmpină o problemă mare în pixelii care se suprapun.

Comparând timpul de rulare al modelelor existente în estimarea fluxului intermediar cu modelul IFNet s-a observat o îmbunătățire semnificativă. În momentul de față, celelalte modele au nevoie să ruleze modelul de două ori pentru a face rost de fluxul bidirecțional. Astfel estiamrea fluxului intermediar folosind modelul RIFE rulează la o viteză extrem de rapidă atingând o accelerare de până la 20 de ori mai mare față de alte modele.

#### **Procesul de fuziune**

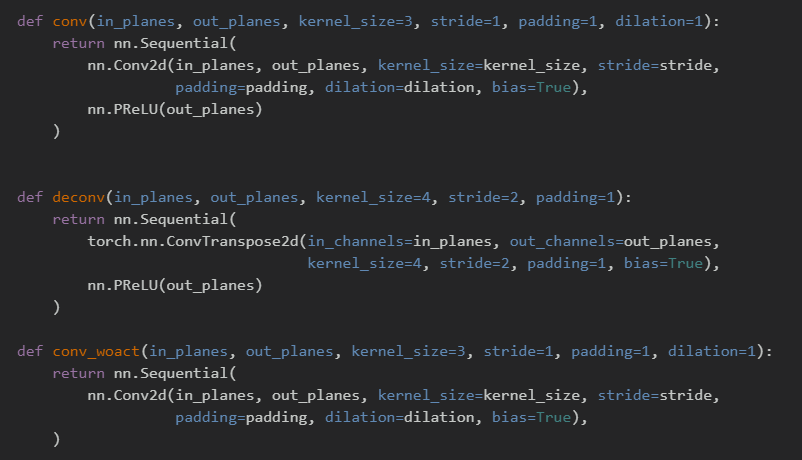
Cu ajutorul fluxului intermediar estimat, putem face rost de cadrele reconstruite prin procesul de deformare înapoi a cadrelor de intrare. Pentru a reduce artefactele severe în cadrele deformate, folosim procesul de fuziune și de rafinare după formula următoare.



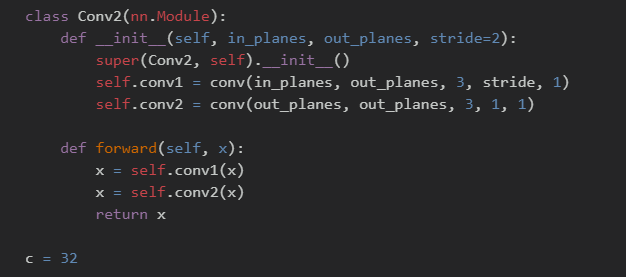
Figură Formulă proces de fuziune și rafinare

Unde M este harta de fuziune folosită pentru a fuziona cele două cadre deformate, delta este reconstrucția termenilor reziduali folosiți în rafinarea detaliilor din imagini.

Urmând lucrările precedente, procesul de fuziune include un extractor și o rețea FusionNet cu o arhitectură codificator – decodificator. Contextul exctractorului și codificatorului au arhitecturi similare, fiind formate din patru grupuri convoluționale și fiecare dintre ele este compuns din două 3x3 straturi convoluționale respectiv. Partea cu decodificator are patru straturi convoluționale transpuse.

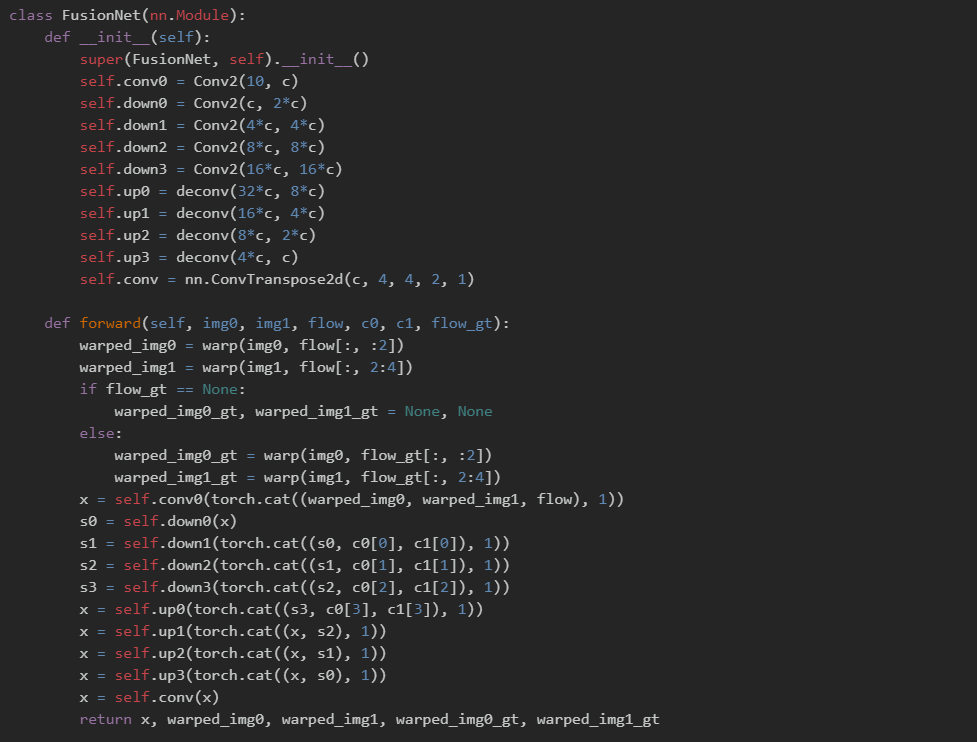


Figură Cod procesul de fuziune a)



Figură Cod procesul de fuziune c)

Figură Cod procesul de fuziune b)

În mod specific, extractorul extrage, prima dată, trăsăturile contextuale piramidale din cadrele de intrare în mod separat. Apoi realizează procedeul de deformare înapoi pe aceste trăsături folosind estimările fluxului intermediar pentru a produce trăsăturile piramidale aliniate. Cadrele deformate și fluxurile intermediare sunt apoi introduse în rețeaua de fuziune, împreună cu un codificator și un decodificator. Rezultattul unui grup de codificare este concatenat cu fluxul înainte de a fi introdus în următorul grup. Decodificatorul produce, într-un final, harta de fuziune și reconstituirea rezidurilor.

Figură Cod procesul de fuziune d)

### **Funcția de pierdere**

Aproximarea directă a fluxurilor intermediare este o dificultate din cauza lipsei de acces la un cadru intermediar și lipsa supravegherii. Pentru a adresa această problemă, am adăugat funcția de pierdere „distilată” la modelul IFNet, a cărei scop este de predicție a unei rețea profesor care are acces la cadrul intermediar. În mode specific am introdus cadrul intermediar estimat către o rețea de estimare a fluxului optic pre-antrenată pentru a face rost de prezicerea fluxului intermediar.

Funcția de pierdere distilată este definită astfel:



Figură Funcția de pierdere distilată creată pentru modelul IFNet

Urmând lucrările anterioare, functia aceasta a fost aplicată peste o secvență întreagă e decizii generată din actualizarea iterativă a procesului în rețeaua IFNet.

Schema de distilare este diferită de cele folosite în algoritmi de învățare semi-supravegheată, unde un model pre-antrenat este folosit pentru a induce o etichetă asupra datelor neetichetate. Cu acces la cadrul țintă, modelul profesor are o perspectivă complet diferită asupra videoclipului studentului. Teoretic, profesorul cauzează o scurgere unde estimatorul de flux poate avea acces la informația cadrului intermediar țintă în timpul antrenamentului și în timpul experimentelor s-a dovedit că această scurgere este extrem de benefică pentru antrenamentul întregului sistem.

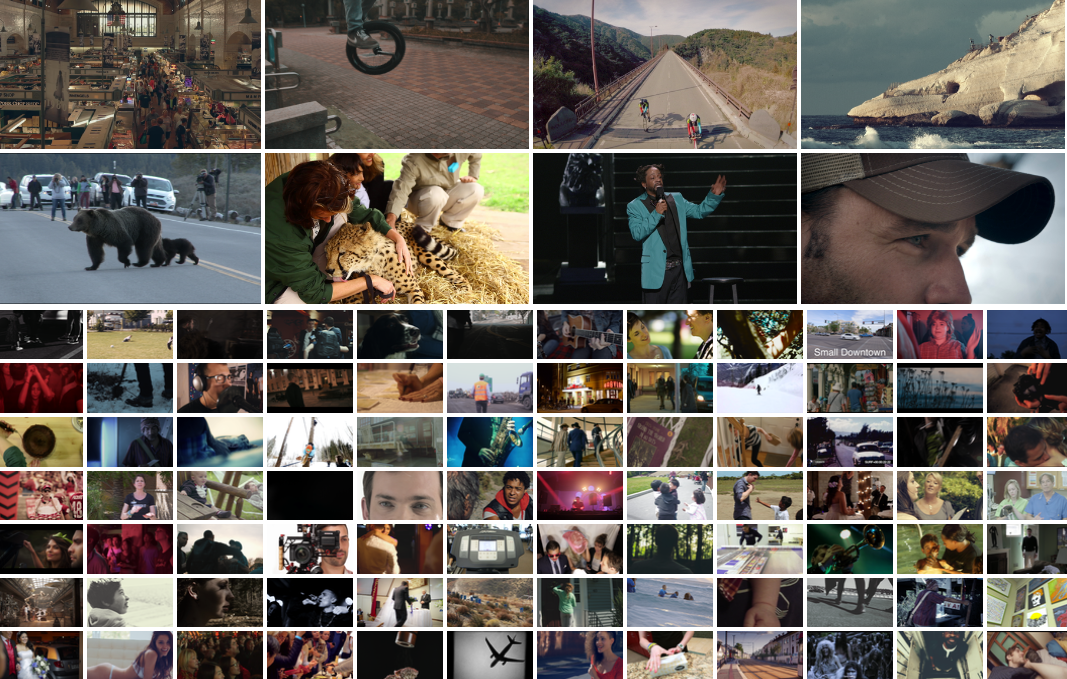
### **Pregătirea setului de date**

Pregătirea setului de date a constat în mai multe operații de rearanjare și decupare a cadrelor existente. Pentru antrenament a fost folosit setul de date vimeo-90k triplet, un set de câte 3 cadre din diferite videoclipuri de pe site-ul vimeo. Setul cuprinde 73.171 de secvențe diferite de câte 3 cadre, extrase din peste 15.000 de videoclipuri diferite. Setul de date a fost creat special pentru antrenamentul de interpolare a videoclipurilor

Toate cadrele sunt aranjate să aibă specific rezoluția de 448x256 și interpolarea este făcută pas cu pas între fiecare două cadre diferite. Modelul poate să dubleze numărul de cadre pe secundă pentru un videoclip HD de 30 de cadre pe o placă video Nvidia 2080TI. Poate, de asemenea, să interpoleze până la 16 cadre intermediare pentru o singură pereche de două imagini.

Datele de intrare au fost învârtite aleator, din când în când, pentru mai multă diversitate în setul de antrenament. Modelul a fost antrenat pe acest set de date. Un model oficial pre-antrenat FlowNet este folosit ca și profesorul care supraveghează și se ocupă de scurgerea de informații.

Modelul este optimizat folosind algoritmul de optimizare AdamW, și antrenat pentru 300 de epoci pe setul de date Vimeo. Am redus rata de învățare puțin câte puțin de la 10^-4 până la zero pe parcursul întregului proces de antrenament.



Figură Set de date Vimeo-90k

### **Generarea de cadre multiple**