Universitatea Tehnică de Construcții din București
Facultatea de Hidrotehnică
Domeniul: Ingineria Sistemelor

Sistem expert pentru optimizarea fișierelor media

Coordonator ştiinţific:
Drd. Ing. Doris Pogăcean

Absolvent:
Gabriel Comănescu

București, 2021

CUPRINS

TABEL DE FIGURI

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

Lucrarea este compusă din 4 capitole și un capitol care concretizează concluziile și perspectivele de viitor ale sistemului expert pentru optimizarea fișierelor media dezvoltat în cadrul acestei lucrări de licență.

În primul capitol au fost prezentate aspectele teoretice referitoare la aplicații de inteligență artificială printre care se numără aplicații ale inteligenței artificiale în diferite domenii, cum ar fi: domeniul medicinei, domeniul mașinilor autonome, domeniul jocurilor și algoritmilor care învață

din experiență. Am vorbit despre diferențele majore între inteligența artificială și machine learning și am descris în detaliu tot procesul de dezvoltare al unui algoritm de machine learning. De asemenea am acoperit o gama largă de detalii despre rețelele neuronale, de la rețele neuronale simple, la rețele de convoluție și până la rețele generativ adversariale, pe care se bazează și această lucrare.

În cel de-al doilea capitol am discutat tehnologiile și instrumentele software folosite în implementarea proiectului, precum mediul de dezvoltare integrat Visual Studio Code, limbajele de programare Python și JavaScript dar și bibliotecile aferente acestora, limbajele de marcare HTML și CSS, cadrul de dezvoltare Flask.

În al treilea capitol, cel al studiului de caz, am concretizat informații despre cele două rețele neuronale antrenate în cadrul acestui proiect precum arhitectura rețelei, pregătirea setului de date, definirea funcțiilor de pierdere, antrenamentul cât și rezultatele. S-a discutat în detaliu despre modelele folosite pentru antrenamentul fiecărei rețele. Deoarece ambele au o abordare unică, modelul joacă cel mai important rol în dezvoltare și în rezultatele obținute.

Al patrulea capitol constă într-o demonstrație a utilizării celor două rețele în mai multe ipostaze pentru a demonstra funcționalitatea lor cât și adaptabilitatea la noi provocări. S-au dat exemple care rezolvă probleme reale și care pot îmbunătăți și ușura depanarea activităților în diferite domenii.

În capitolul final, cel al concluziilor, am dezbătut impactul pe care îl poate avea acest proiect asupra diferitelor domenii de muncă, cât și impactul pe care l-a avut asupra mea. Am vorbit despre lucrurile pe care le-am învățat pe parcurs și adresez problemele de viitor și potențiale îmbunătățiri.

1. ASPECTE TEORETICE REFERITOARE LA APLICAȚII DE INTELIGENȚĂ ARTIFICIALĂ

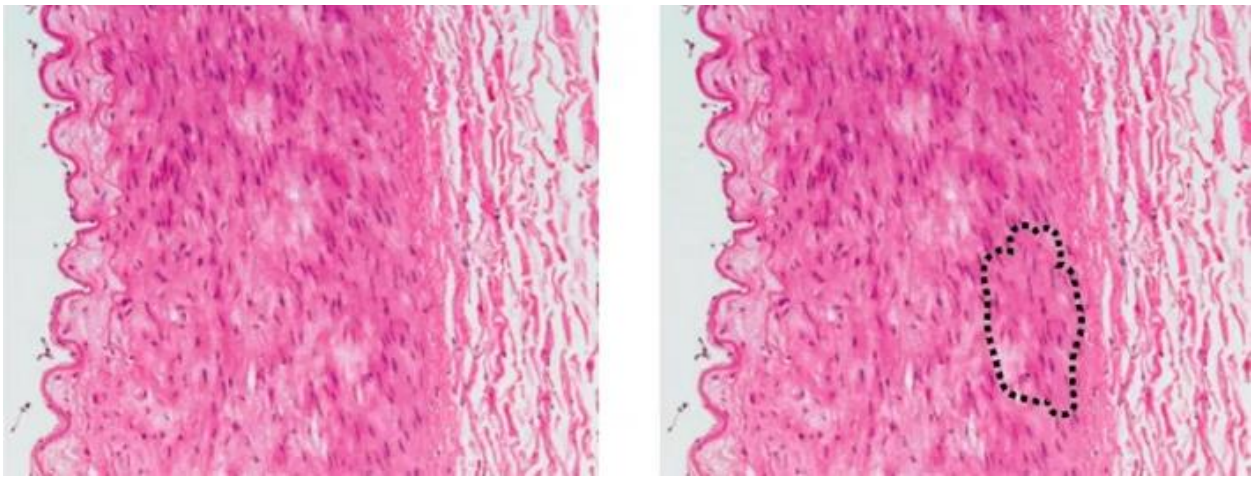
[2]Inteligenta Artificiala (AI) se refera la simularea inteligentei umane in mașinării care sunt programate sa gândească si sa copieze acțiunile oamenilor. Termenul se poate aplica oricărei mașinării care prezinta trăsături asociate cu mintea umana, cum ar fi învățarea si rezolvarea de probleme.

[2]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

1.1.1. Aplicări ale inteligenței artificiale în domeniul medicinii:

- [1] 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.

- [1]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.

1.1.2. Mașini care se conduc singure:

[3]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...

[3]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.

[4]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.

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

Figură 1.2: Tesla Model 3 sistem de navigare

[3] [6] [7]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. 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.

Figură 1.3:Robo-taxi lansat în Singapore de compania NuTonomy

1.1.3. Aplicații în industria jocurilor și algoritmi care învață din experiență:

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.

- [8]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.

[9] [10]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.

[9]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.

[11]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.

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

- [12]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.

[12]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%.

[13]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.

[14]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.

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

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

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

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

1.2. Inteligența artificială și machine learning

[15]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.

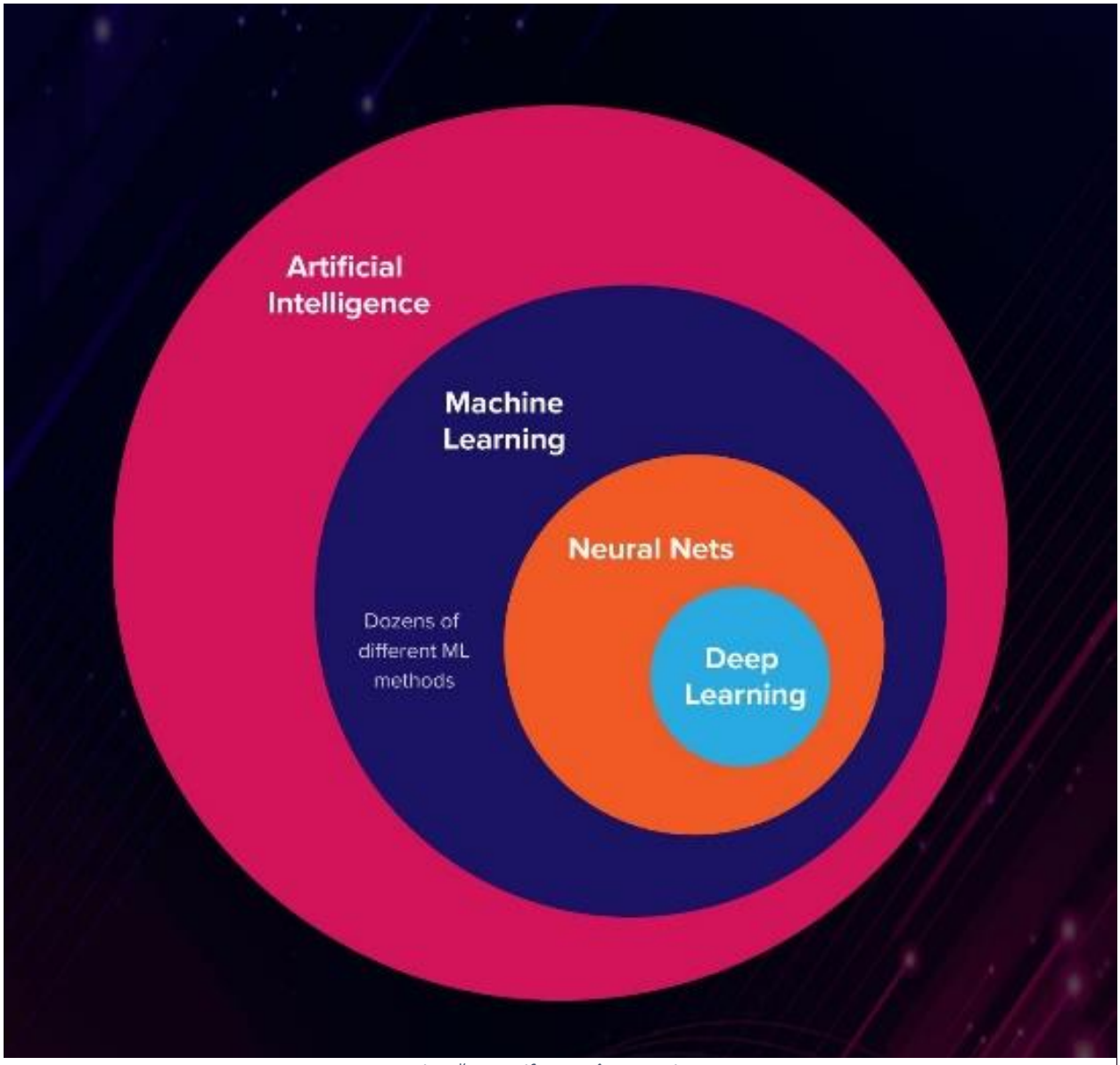
Diferențe cheie între inteligența artificială și machine learning:

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 1.4: Diferența între AI și ML

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

[15] „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”.

1.3. Despre machine learning

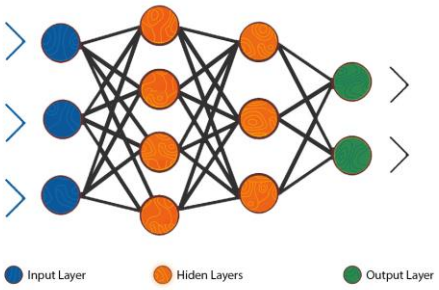
[16]Machine learning funcționează prin intermediul unor algoritmi care învață singuri folosind date. Funcționează numai în domenii specifice cum ar fi dacă creăm un model care detectează imagini cu câini, va oferi rezultate numai pentru imagini cu câini. Dacă ar fi să îi arătăm poze cu pisici ar deveni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

1.3.1. Rețele neuronale artificiale

[17]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.

[18]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 1.5: Exemplu rețea neuronală simplă

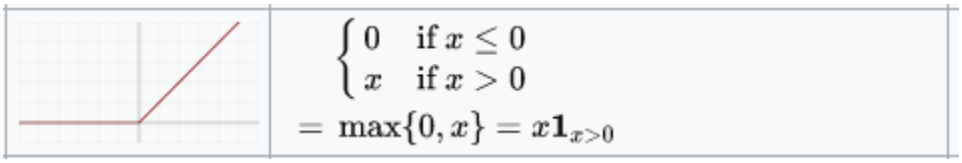
[18]Modul în care rețeaua se antrenează este, mai mult sau mai puțin, un proces de operații matematice, în care, pentru fiecare neuron, fiecare greutate se înmulțește cu valoarea neuronului, se adună toate împreună cu bias-ul și sunt trecute printr-o funcție de activare.

$$Y = \sum (weight * input) + bias$$

Figură 1.6 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.

[19]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.

[20]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.

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

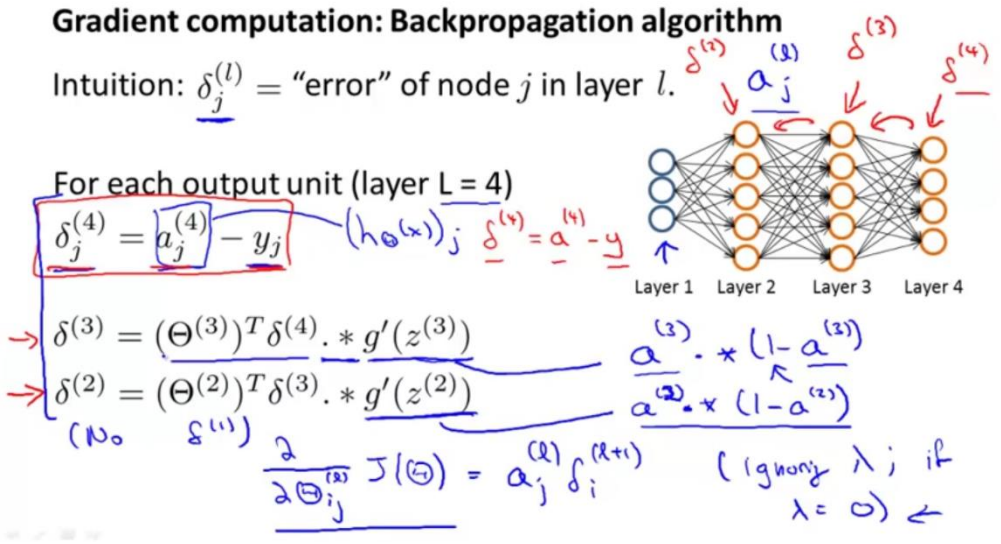
Figură 1.8: Mean Squared Error

[20]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.

$$J = -\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i)\log(1 - h_\theta(x_i))$$

Figură 1.9: Binary Cross Entropy Loss Function

[21]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.



Figură 1.10: Backpropagation

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.

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.

1.3.2. Rețele neuronale convoluționale

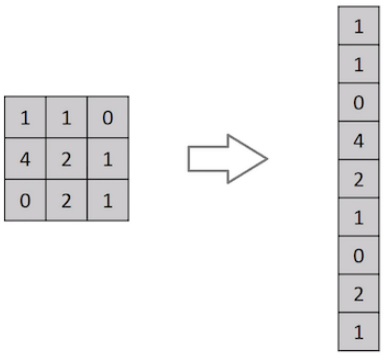
[18]Rețelele convoluționale neuronale se folosesc de filtre, sau „nuclee”(kernels), folosite pentru a extrage trăsături relevante din datele de intrare folosind operații de convoluție. Sunt folosite în mod special în lucrul cu imagini și procesare video.

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?

[23]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.

[23]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.



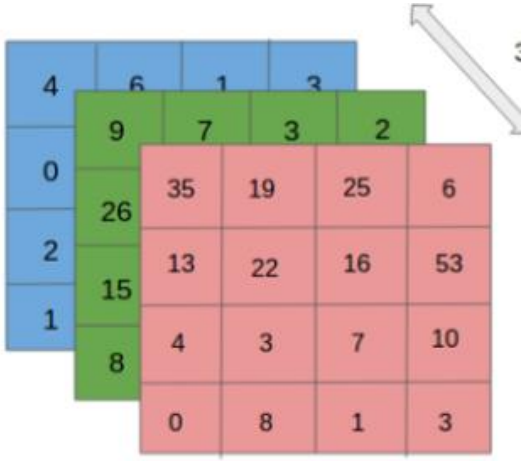
Figură 1.11: filtrarea unei imagini 3x3 într-un vector 9x1

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.

[23]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.

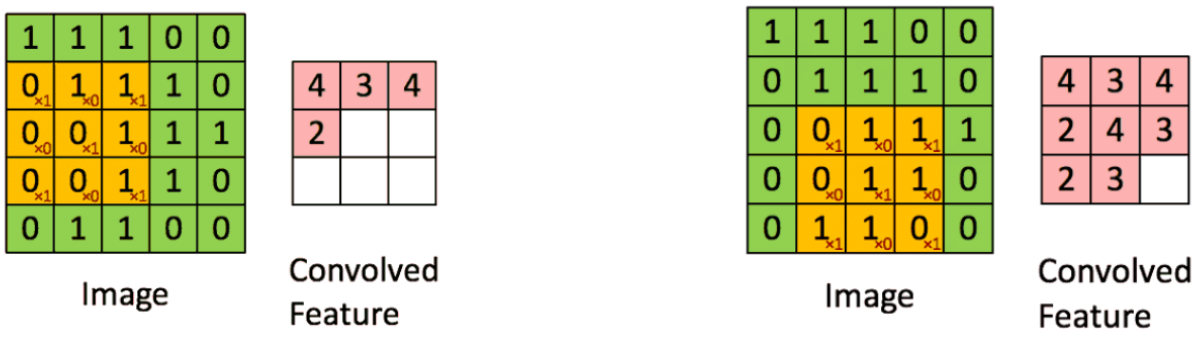
Acest lucru este foarte important în crearea modelului, întrucât trebuie să reușească să se ajusteze și la seturi de date mult mai mari.



Figură 1.12: Impartirea unei imagini in 3 canale de culoare

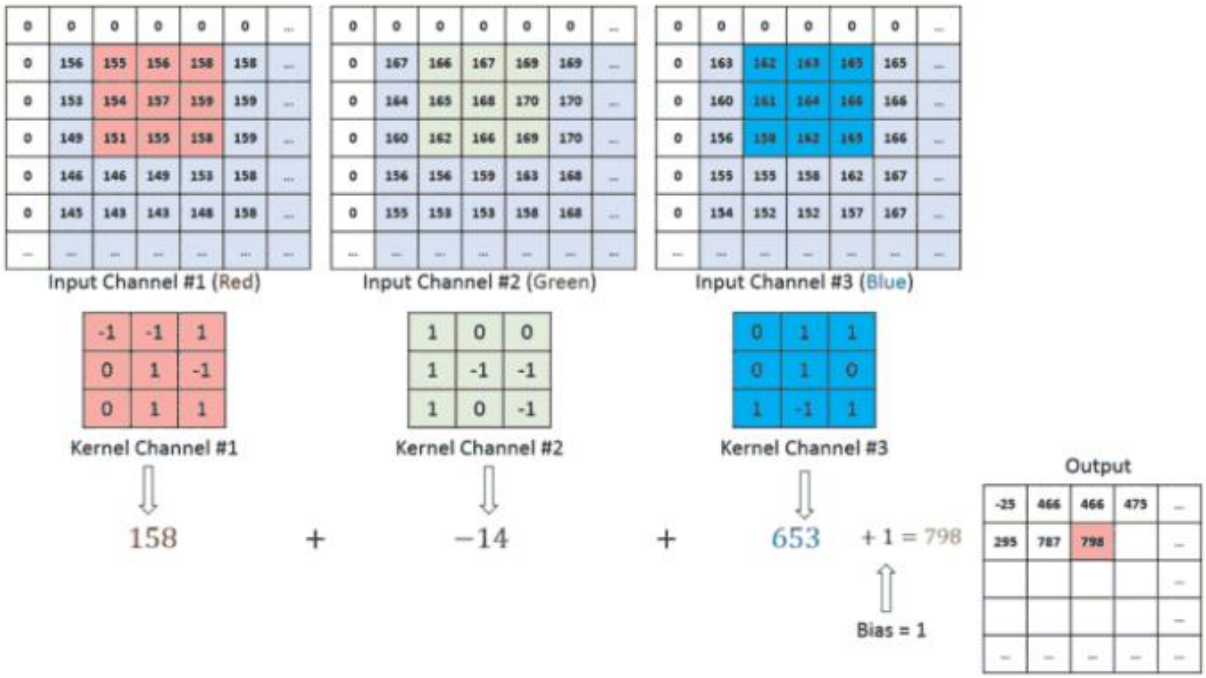
Nucleul (Kernel)

[23]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.



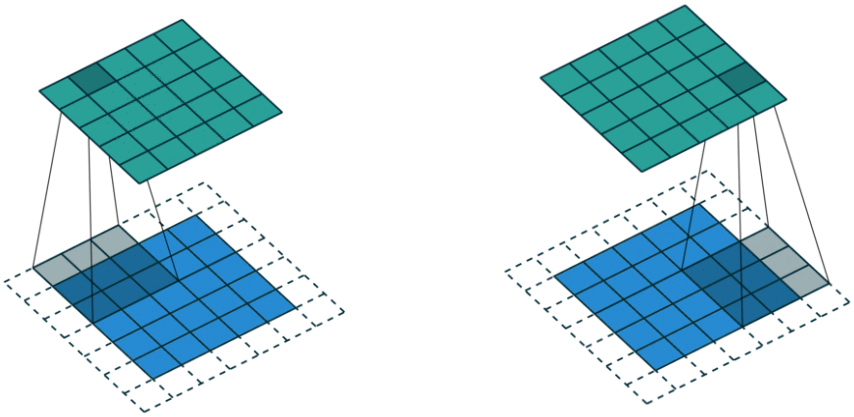
Figură 1.13: Exemplu al aplicării unui filtru

[23]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.



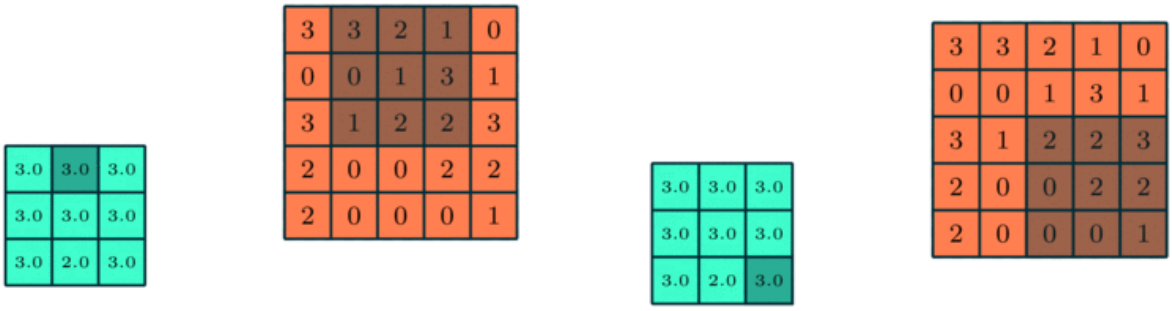
Figură 1.14: Aplicarea unui filtru pe mai multe canale de culoare

[23]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

[23]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

1.3.3. Tipuri de machine learning

- **Supervised learning**:

[24]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**:

[25]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**

[26]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.

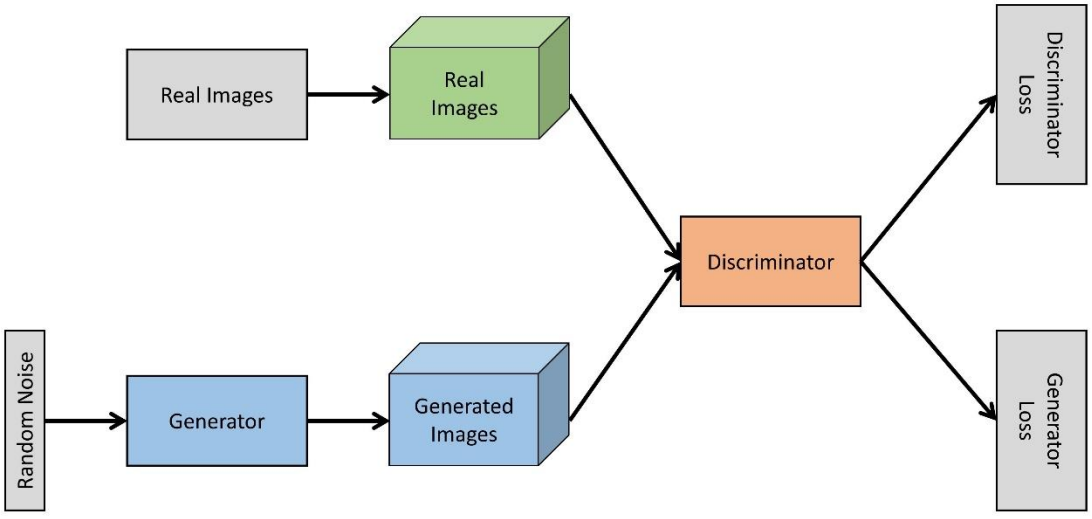
Proiectul meu de licență este bazat pe acest tip de machine learning. Mai exact pe o subcategorie a acestui tip numită Rețele Generativ Adversariale.

1.3.4. Rețele Generativ Adversariale (GAN)

[27]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.

[28]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.

[28]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

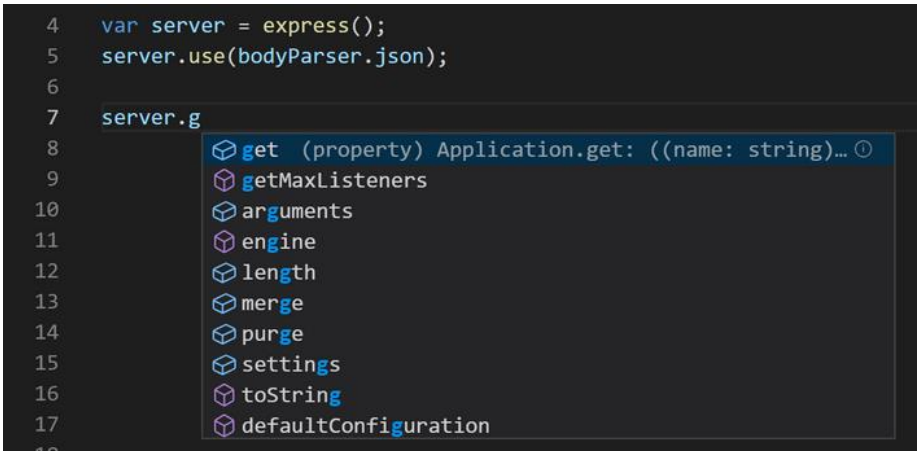
2. TEHNOLOGII ȘI INSTRUMENTE SOFTWARE FOLOSITE ÎN IMPLEMENTAREA PROIECTULUI

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

[29]Visual Studio Code este un editor de cod „simplu”, dar puternic creat de Microsoft. Acesta rulează pe desktop și este potrivit pentru Windows, MacOS și Linux. Are suport încorporat pentru JavaScript, TypeScript și Node.js și oferă un ecosistem extins foarte bogat pentru alte limbaje de programare (cum ar fi C ++, C #, Python, Java, PHP).

[30]Într-un sondaj din 2019 al platformei StackOverflow, peste 50% din 87.317 dezvoltatori au votat să utilizeze Visual Studio Code, care este cel mai popular mediu de dezvoltare.



Figură 2.1 Visual Studio Code

[30]În loc de o structură a proiectului, permite utilizatorilor să deschidă unul sau mai multe directoare, care pot fi apoi salvate în spațiul de lucru. Aceasta permite scrierea în mai multe limbaje de programare având mai multe ferestre de cod deschise. Fișierele inutile pot fi excluse din proiect prin intermediul setărilor. De asemenea, utilizatorii își pot personaliza experiența în Visual Studio Code instalând diverse teme sau extensii care modifică aspectul programului și a codului scris.

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

2.2. Python

[31] Python este un limbaj de programare interpretat, orientat pe obiecte și de nivel înalt care utilizează semantică dinamică. Structurile de date predefinite și dezvoltarea dinamică de cod îl fac să fie o alegere excelentă pentru crearea rapidă de aplicații. Sintaxa ușor de învățat oferă o ușurință în citire și reduce costurile și nevoile de întreținere a unui program. Python acceptă module și pachete, lucru care încurajează modularitatea programului și reutilizarea codului.

[32]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

[32]Python nu folosește funcții de bază, dar este conceput pentru a fi extrem de extensibil folosind module. Modularitatea compactă îl face foarte popular ca instrument pentru adăugarea de interfețe programabile la aplicațiile existente.

[32]Python se străduiește să ofere o sintaxă mai simplă și mai concisă, iar programatorii își pot alege propria metodă de codare. Filozofia adoptată de limbajele de programare este „ar trebui să existe una, de preferință singura și evidentă modalitate de a o rezolva". Alex Martelli, un angajat al Python Software Foundation și autorul cărții Python, a scris: „Descrierea ceva ca fiind inteligent nu este un compliment în cultura Python."

[32]Folosesc Python ca principalul meu limbaj de dezvoltare deoarece toți algoritmii de machine learning sunt scriși în Python. În plus, partea de backend a interfeței web este creată folosind Python.

2.3. PyTorch

[33]PyTorch este o bibliotecă open source de învățare automată bazată pe biblioteca Torch pentru diverse aplicații, cum ar fi viziunea computerizată și procesarea limbajului natural. A fost inițial dezvoltat de Facebook, un laborator de cercetare a inteligenței artificiale și este gratuit și open source. Deși interfața Python este mai îngrijită și principalul obiectiv al dezvoltării, PyTorch are și o interfață C ++.

[33]Multe proiecte de învățare profundă sunt dezvoltate folosind PyTorch, cum ar fi Pilotul automat Tesla și Pyro de la Uber.

[33]PyTorch folosește o clasă numită Tensor pentru a stoca și a efectua operațiuni pe tablouri multidimensionale similare. Tensorul este similar cu un tablou NumPy, dar poate fi procesat și pe o placă video Nvidia care acceptă tehnologia CUDA.

2.4. NumPy

[34]NumPy este o bibliotecă Python care adaugă suport pentru matrici și vectori imens multi-dimensionali, precum și o colecție mare de funcții matematice avansate pentru manipularea acestor matrice.

[34]Limbajul Python nu a fost inițial conceput pentru calcule numerice, dar a atras atenția oamenilor de știință. Și inginerii. Un grup a fost format în 1995 pentru a defini pachetele software de limbaj de programare și pentru a extinde sintaxa pentru ușurință în utilizare. Mai târziu, un nou pachet numit Numarray a fost dezvoltat ca o alternativă mai flexibilă la Numeric. Funcționează mai repede pe vectori mari, dar mai lent pe vectori mici, astfel încât cei doi sunt utilizați în paralel. În 2005, s-a dprit unificarea comunității în jurul unui pachet software. Prin urmare, prin combinarea celor două, se ajunge la NumPy.

[34]NumPy își propune să implementeze referința CPython ca interpret. Utilizând NumPy, puteți obține funcții echivalente cu Matlab, deoarece ambele sunt interpretate. Atâta timp cât majoritatea operațiilor folosesc matrice în loc de scalare, ambele permit utilizatorilor să dezvolte programe rapide.

2.4. Pandas

[35]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.

[35]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.

[35]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

[36]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

[37]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.

[37]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

[38]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.

[38]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.

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

2.4. CSS

[39]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.

[39]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

[40]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.

[40]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.

[40]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.

[40]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

[41]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:

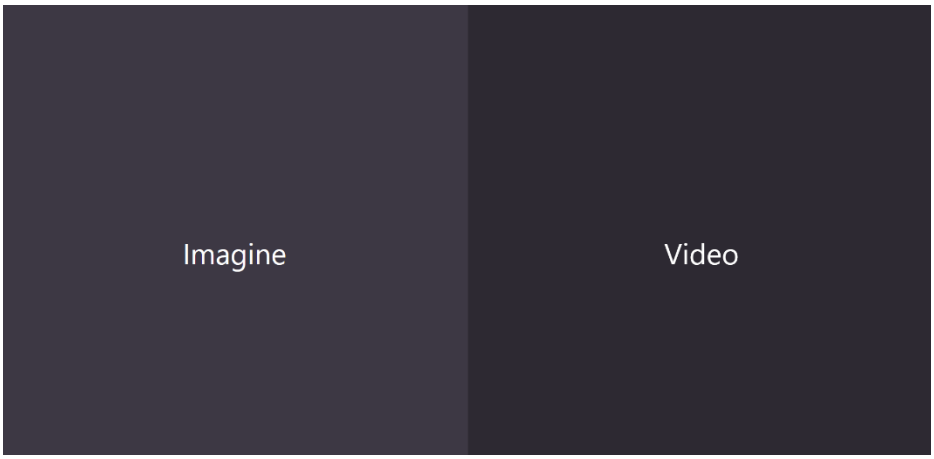
- Werkzeug este o bibliotecă utilitară menită pentru folosința cu Python. În mare parte este un gateway către un server care poate crea articole software pentru cereri, răspunsuri și funcții utile.
- Jinja este un motor șablon pentru programarea în Python și se aseamănă cu șblonul din Django.

Aplicații care folosesc Flask includ Pinterest și Linkedin.

3. STUDIU DE CAZ „SISTEM EXPERT PENTRU OPTIMIZAREA FIȘIERELOR MEDIA”

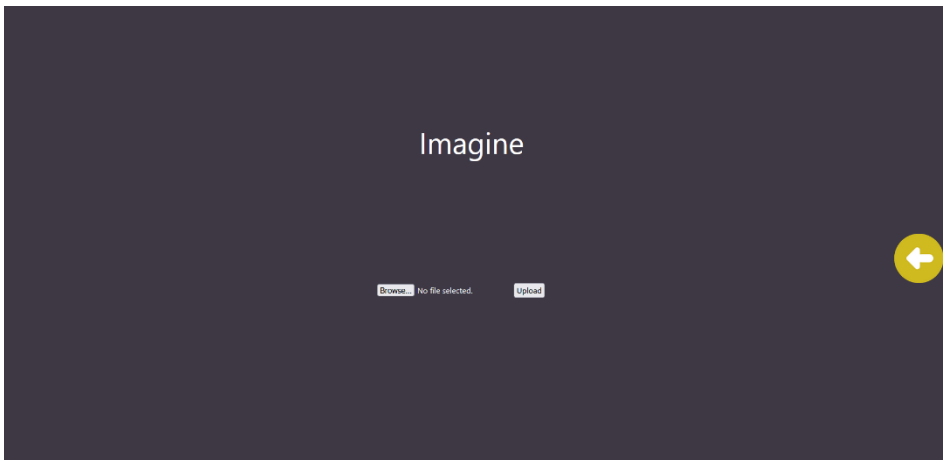
Proiectul meu este împărțit în două rețele neuronale diferite: una dublează rezoluția imaginilor, făcându-le mai clare, cealaltă aplică interpolarea cadrelor pe videoclipuri și le face mai fluide.

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

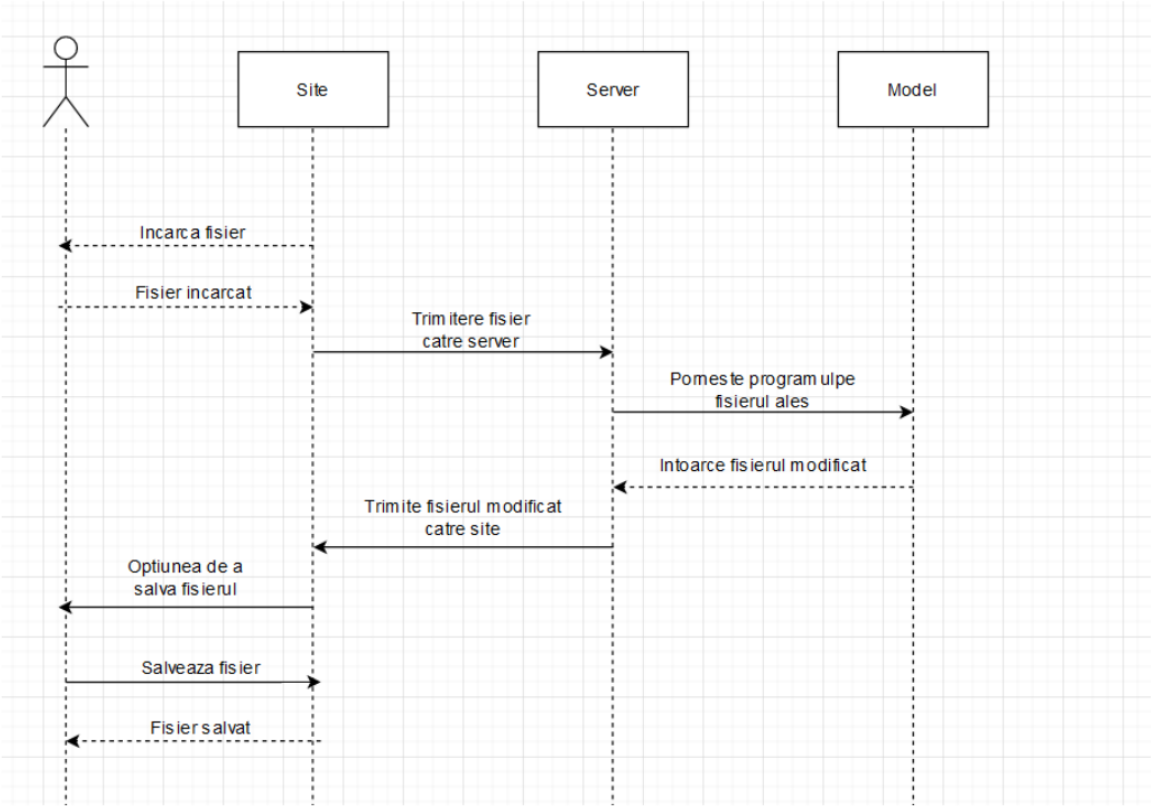
Selectând una dintre cele două va oferi posibilitatea de a încărca fișierul dorit pentru optimizare. În cazul în care încărcăm o imagine, va acționa prima rețea neuronală, altfel va acționa cea de-a doua rețea neuronală.



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 3.3 Diagrama secventiala

[42]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.

3.1. 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.

3.1.1. Introducere

Problema mare prezentă în estimarea rezoluției înalte (HR) ale unei imagini din rezoluția ei slabă (LR) și se referă la ea ca și super rezoluție (SR). SR a primit o atenție substanțială din partea comunității de cercetare a viziunii computerizate și are un domeniu larg de aplicații.

[43]Problema legată de determinarea super rezoluției este în mod particular pronunțată pentru factorii de scalare înalți, pentru care detaliile texturii sunt, în mod general, absente. Ținta de optimizare pentru algoritmii SR supravegheați este, de obicei, minimizarea erorii scoase din funcția de eroare MSE între imaginea generată de generator și imaginea reală. Acest lucru este extrem de convenabil astfel că minimizarea erorii în același timp maximizează rația dintre semnal și zgomot, ceea ce este o mărime de măsură comună folosită în evaluarea și compararea algoritmilor SR. Totuși, abilitatea acestor funcții de a captura diferențe perceptuale relevante, cum ar fi detaliu texturilor de calitate înaltă, este foarte limitată, astfel că sunt definite bazate pe diferențele dintre pixeli în imagini. Diferența perceptuală între imaginile SR și originalul înseamnă că imaginea recuperată nu este fotorealistă.

În acest subcapitol este vorba despre o soluție folosind o rețea generativ adversarială pentru care folosesc un rețea reziduală (REsNet). Este diferită de lucrări anterioare prin definirea unei funcții de pierdere folosind o hartă de caracteristici a rețelei VGG combinată cu un discriminator care încurajează soluții perceptuale, greu de distins de imaginile de referință HR.

3.1.2. Design-ul rețelei neuronale de convoluție

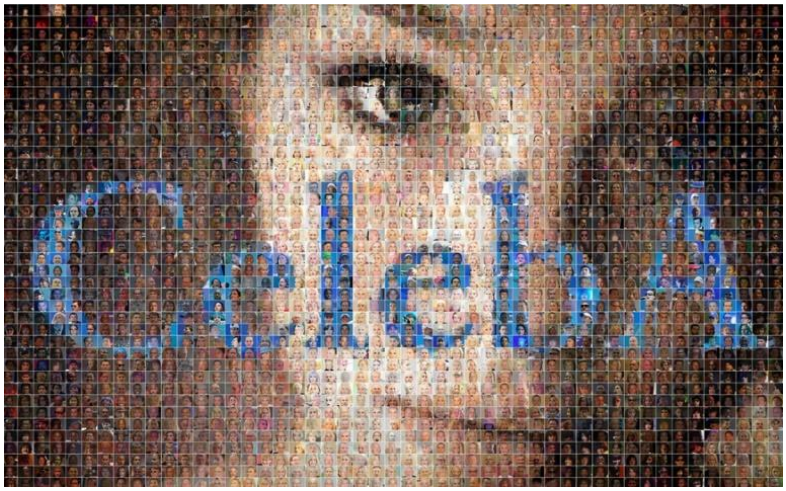
Stadiul tehnic actual al multor probleme legate de viziune computerizată este setat de către rețele neuronale de convoluție special proiectate pentru sarcinile respective.

A fost demonstrat faptul că arhitecturile de rețele neuronale adânci pot fi dificil de antrenat, însă au potențialul să mărească substanțial acuratețea rețelei deoarece permit modelarea unor hărți de complexitate înaltă. Pentru a antrena în mod eficient aceste rețele adânci, normalizarea datelor este foarte des folosită și, de obicei, utilizată împreună cu schimbarea variabilelor. Un design foarte puternic care ușurează antrenamentul rețelelor neuronale de convoluție adânci este introducerea recentă a conceptului de grupuri reziduale și conexiuni sărite. Conexiunile sărite ușurează arhitectura rețelei de la modelarea hărților de identitate, ceea ce reprezintă un lucru trivial, totuși,

potențial de a nu fi atât de trivial în momentul în care este necesar să fie reprezentat folosind nuclee de convoluție.

3.1.3. Pregătirea setului de date

[44]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.

3.1.4. 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.

Funcția de pierdere la nivel de pixeli, la fel ca și MSE, se străduiește să facă față incertitudinii inerente în recuperarea detaliilor de frecvență mare pierdute cum ar fi texturile. Minimizarea erorii încurajează descoperirea mediilor la nivel de pixeli a soluțiilor plauzibile care sunt, de obicei, foarte fine și astfel au o calitate perceptuală slabă.

În alte publicații autorii au abordat problema aceasta prin implementarea rețelelor generativ adversariale pentru aplicările de generare de imagini. Rețelele generativ adversariale au fost de asemenea folosite pentru învățarea reprezentativă nesupravegheată. Ideea folosirii unui GAN pentru a învăța punerea în hartă de la un colector la altul a fost discutată, iar MSE a fost minimizată în spațiile funcționale ale modelului VGG19.

O idee de funcție de pierdere ar fi bazată pe distanța euclidiană calculată în spațiul funcțional al rețelei neuronale în combinație cu antrenamentul adversarial. Este demonstrat cum pierderea propusă permite vizualizarea unei imagini generate superioare și poate fi folosită pentru a rezolva problema decodificării caracteristicilor non-liniare reprezentate. Similar, o idee a fost propusă prin folosirea caracteristicilor extrase dintr-o rețea VGG19 pre-antrenată în locul unei măsurări de eroare la nivel de pixel.

```python
from torchvision.models.vgg import vgg19


class GeneratorLoss(nn.Module):
    def __init__(self):
        super(GeneratorLoss, self).__init__()
        vgg = vgg19(pretrained = True)
        loss_network = nn.Sequential(*list(vgg.features)[:31]).eval()
        for param in loss_network.parameters():
            param.requires_grad = False
        self.loss_network = loss_network
        self.mse_loss = nn.MSELoss()
        self.tv_loss = TVLoss()

    def forward(self, out_labels, out_images, target_images):
        adversarial_loss = torch.mean(1 - out_labels)
        perception_loss = self.mse_loss(self.loss_network(out_images), self.loss_network(target_images))
        image_loss = self.mse_loss(out_images, target_images)
        tv_loss = self.tv_loss(out_images)
        return image_loss + 0.001 * adversarial_loss + 0.006 * perception_loss + 2e-8 * tv_loss
```

Figură 3.4 Cod, functiile de pieredere a)

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.

```
class TVLoss(nn.Module):
    def __init__(self, tv_loss_weight=1):
        super(TVLoss, self).__init__()
        self.tv_loss_weight = tv_loss_weight

    def forward(self, x):
        batch_size = x.size()[0]
        h_x = x.size()[2]
        w_x = x.size()[3]
        count_h = self.tensor_size(x[:,:,1:,:])
        count_w = self.tensor_size(x[:,:,:,1:])
        h_tv = torch.pow((x[:,:,1:,:] - x[:,:,:h_x - 1,:]),2).sum()
        w_tv = torch.pow((x[:,:,:,1:] - x[:,:,:,:w_x - 1]),2).sum()
        return self.tv_loss_weight * 2 * (h_tv / count_h + w_tv / count_w) / batch_size

    @staticmethod
    def tensor_size(t):
        return t.size()[1] * t.size()[2] * t.size()[3]
```

Figură 3.5 Cod, funcții de pierdere b)

3.1.4.1. Funcția de pierdere perceptuală

[45]Definiția funcției de pierdere conceptuală este critică pentru performanța rețelei generator. În timp ce funcția este, de obicei, bazată pe MSE, în acest model a fost concepută o funcție care adresează o soluție cu respect pentru caracteristicile perceptuale relevante. Funcția este definită ca și suma pierderii conținutului și suma componentei adversariale ca fiind următoarea:

$$l^{SR} = \underbrace{\underbrace{l_{\mathrm{X}}^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}}$$

Figură 3.6 Funcția de pierdere perceptuală pentru rețeaua VGG

3.1.4.2. Funcția de pierdere a conținutului

[45]Funcția de pierdere MSE la nivel de pixeli este calculată în felul următor:

$$l_{MSE}^{SR} = \frac{1}{r^2WH}\sum_{x=1}^{rW}\sum_{y=1}^{rH}(I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2$$

Figură 3.7 Funcția de pierdere MSE la nivel de pixeli

Această funcție este cea mai des întâlnită optimizare pentru o imagine SR. Totuși, în timp ce atingerea unei acurateţe favorabile, soluția optimizării MSE întâmpină probleme de conținut cu frecvență mare, ceea ce rezultă în niște soluții perceptual nesatisfăcătoare cu texturi nu foarte bine definite.

[45]În loc de a ne baza pe pierderi calculate la nivel de pixeli, am folosit o funcție de pierdere care este cel mai apropiat de similaritate perceptuală. Am definit funcția de pierdere VGG bazată pe straturi de activare ReLU a unei rețele VGG19 pre-antrenate. Pierderea VGG este definită ca și distanța euclidiană între reprezentările caracteristicilor imaginii reconstruite și imaginea de referință.

$$l_{VGG/i.j}^{SR} = \frac{1}{W_{i,j}H_{i,j}}\sum_{x=1}^{W_{i,j}}\sum_{y=1}^{H_{i,j}}(\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

Figură 3.8 Funcția de pierdere VGG

3.1.4.3.Funcția de pierdere adversarială

[45]În adiție la pierderea de conținut descrisă anterior, se adaugă, de asemenea, și o componentă generativă ale rețelei la pierderea perceptuală. Acest lucru încurajează rețeaua să favorizeze soluțiile care sunt prezente în colectorul natural de imagini prin încercarea de a păcălii discriminatorul. Funcția de pierdere generativă este definită ca fiind bazată pe probabilitățile pe care le are discriminatorul asupra tuturor datelor de antrenament.

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

Figură 3.9 Funcția de pierdere adversarială

3.1.5. Arhitectura rețelei generativ adversariale

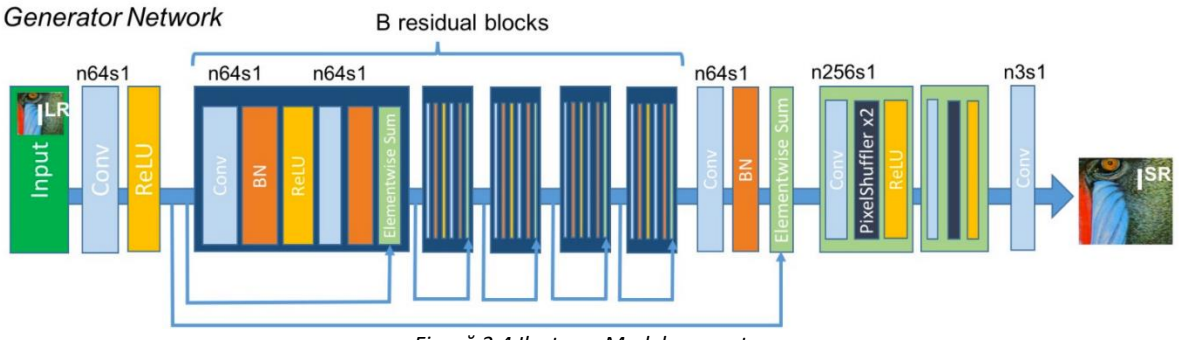
[45]Rețeaua discriminatorului este optimizată într-o manieră alternativă împreună cu rețeaua generatorului pentru a rezolva problema adversarială de min-max:

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{\text{train}}(I^{HR})}[\log D_{\theta_D}(I^{HR})] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))]$$

Figură 3.10 problema min-max a rețelei adversariale

Ideea generală din spatele acestei formulări este faptul că permite antrenamentul generativ al modelului de generator cu scopul de a păcăli discriminatorul care este antrenat să facă diferența între imaginile reale și cele generate. Folosind această abordare, generatorul poate învăța să creeze soluții care sunt extrem de similare cu imaginile reale, astfel făcând foarte dificil pentru discriminator să clasifice. Acest lucru încurajează soluțiilor superior perceptuale care se află în subspațiul colectorului imaginilor naturale.

3.1.5.1. Modelul generatorului



Figură 3.4 Ilustrare Model generator

[45]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.

```
class ResidualBlock(nn.Module):
    def __init__(self, channels):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(channels,channels, kernel_size=3,padding=1)
        self.bn1 = nn.BatchNorm2d(channels)
        self.prelu = nn.PReLU()
        self.conv2 = nn.Conv2d(channels, channels, kernel_size=3,padding=1)
        self.bn2 = nn.BatchNorm2d(channels)

    def forward(self,x):
        residual = self.conv1(x)
        residual = self.bn1(residual)
        residual = self.prelu(residual)
        residual = self.conv2(residual)
        residual = self.bn2(residual)

        return x + residual
```

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.

[46]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.

```
# UPSAMPLE BLOCK

class UpsampleBlock(nn.Module):
    def __init__(self,in_channels, up_scale):
        super(UpsampleBlock, self).__init__()
        self.conv = nn.Conv2d(in_channels,in_channels * up_scale ** 2, kernel_size=3, padding=1)
        self.pixel_shuffle = nn.PixelShuffle(up_scale)
        self.prelu = nn.PReLU()

    def forward(self, x):
        x = self.conv(x)
        x = self.pixel_shuffle(x)
        self.prelu(x)
        return x
```

Figură 3.6 Cod, grupul de upsample

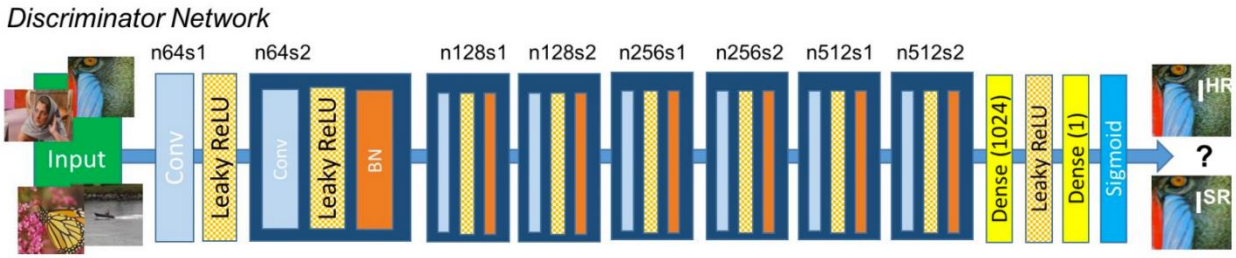
Puse toate cap la cap, Funcția care definește clasa generatorului arată astfel

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

```
class Generator(nn.Module):

    def __init__(self, scale_factor):
        upsample_block_num = int(math.log(scale_factor,2))

        super(Generator, self).__init__()

        self.block1 = nn.Sequential(nn.Conv2d(3,64,kernel_size=9,padding=4),nn.PReLU())
        self.block2 = ResidualBlock(64)
        self.block3 = ResidualBlock(64)
        self.block4 = ResidualBlock(64)
        self.block5 = ResidualBlock(64)
        self.block6 = ResidualBlock(64)
        self.block7 = nn.Sequential(nn.Conv2d(64,64,kernel_size=3,padding=1), nn.BatchNorm2d(64))
        block8 = [UpsampleBlock(64, 2) for _ in range(upsample_block_num)]
        block8.append(nn.Conv2d(64,3,kernel_size=9,padding=4))
        self.block8 = nn.Sequential(*block8)

    def forward(self, x):
        block1 = self.block1(x)
        block2 = self.block2(block1)
        block3 = self.block3(block2)
        block4 = self.block4(block3)
        block5 = self.block5(block4)
        block6 = self.block6(block5)
        block7 = self.block7(block6)
        block8 = self.block8(block1 + block7)

        return (torch.tanh(block8) + 1) / 2
```

Figură 3.7 Cod, Clasa Generatorului

3.1.5.2. Modelul discriminatorului



Figură 3.8 Ilustrare model discriminator

[45]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.

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

```
# DISCRIMINATOR

class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator,self).__init__()
        self.net = nn.Sequential(nn.Conv2d(3,64,kernel_size=3,padding=1),
                                 nn.LeakyReLU(0.2),

                                nn.Conv2d(64,64,kernel_size=3,stride=2,padding=1),
                                nn.BatchNorm2d(64),
                                nn.LeakyReLU(0.2),

                                nn.Conv2d(64,128,kernel_size=3,padding=1),
                                nn.BatchNorm2d(128),
                                nn.LeakyReLU(0.2),

                                nn.Conv2d(128,128,kernel_size=3,stride=2,padding=1),
                                nn.BatchNorm2d(128),
                                nn.LeakyReLU(0.2),

                                nn.Conv2d(128,256,kernel_size=3,padding=1),
                                nn.BatchNorm2d(256),
                                nn.LeakyReLU(0.2),

                                nn.Conv2d(256,256,kernel_size=3,stride=2,padding=1),
                                nn.BatchNorm2d(256),
                                nn.LeakyReLU(0.2),

                                nn.Conv2d(256,512,kernel_size=3,padding=1),
                                nn.BatchNorm2d(512),
                                nn.LeakyReLU(0.2),

                                nn.Conv2d(512,512,kernel_size=3,stride=2,padding=1),
                                nn.BatchNorm2d(512),
                                nn.LeakyReLU(0.2),

                                nn.AdaptiveAvgPool2d(1),
                                nn.Conv2d(512,1024,kernel_size=1),
                                nn.LeakyReLU(0.2),
                                nn.Conv2d(1024,1,kernel_size=1)
                                )

    def forward(self, x):
        batch_size = x.size(0)
        return torch.sigmoid(self.net(x).view(batch_size))
```

Figură 3.9 Cod, Clasa Discriminatorului

3.1.6. 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.

```
#UPDATE D network
netD.zero_grad()
real_out = netD(real_img).mean()
fake_out = netD(fake_img).mean()
d_loss = 1 - real_out + fake_out
d_loss.backward(retain_graph=True)


#UPDATE G network
netG.zero_grad()
g_loss = generator_criterion(fake_out, fake_img, real_img)
g_loss.backward()

fake_img = netG(z)
fake_out = netD(fake_img).mean()

optimizerG.step()

optimizerD.step()
```

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.

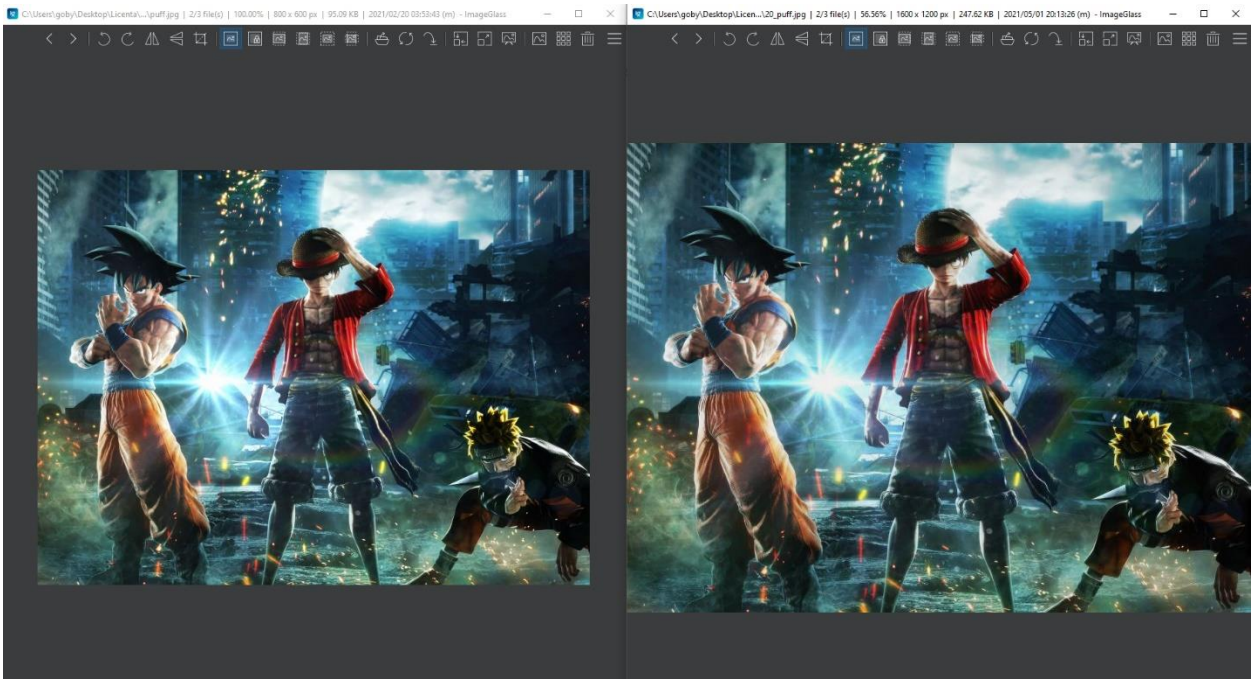
La finalul fiecărei epoci am afișat câteva date ajutătoare, incluzând pierderea fiecărui model și o imagine de test pentru a observa evoluția modelului în timp.

3.1.7. 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:

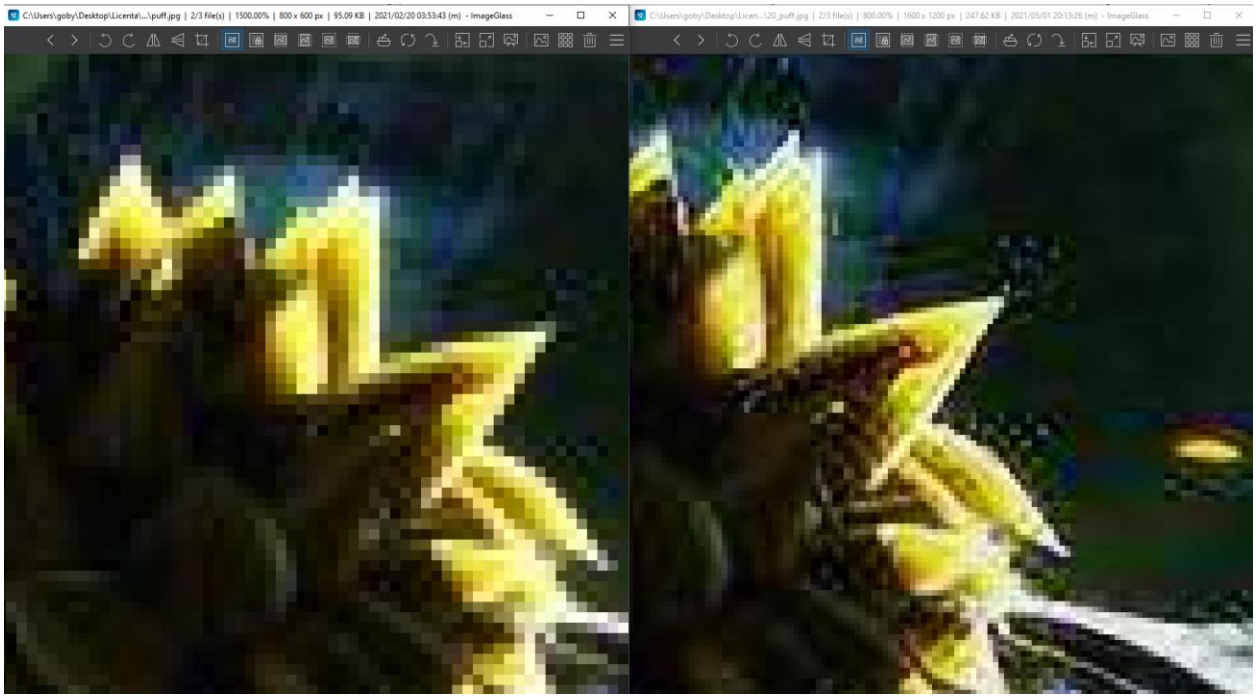


Figură 3.11 Exemplu imagine, inainte și după să fie trecută prin rețeaua neuronală a)

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

3.2. 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.

3.2.1. 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

[47]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.

3.2.2. 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.

```python
def conv_wo_act(in_planes, out_planes, kernel_size=3, stride=1, padding=1, dilation=1):
    return nn.Sequential(
        nn.Conv2d(in_planes, out_planes, kernel_size=kernel_size, stride=stride,
                  padding=padding, dilation=dilation, bias=True),
    )

def conv(in_planes, out_planes, kernel_size=3, stride=1, padding=1, dilation=1):
    return nn.Sequential(
        nn.Conv2d(in_planes, out_planes, kernel_size=kernel_size, stride=stride,
                  padding=padding, dilation=dilation, bias=True),
        nn.PReLU(out_planes)
    )
```

Figură 3.14 Cod rețea IFNet a)

```python
#Block pt retea
class IFBlock(nn.Module):
    def __init__(self, in_planes, scale=1, c=64):
        super(IFBlock, self).__init__()
        self.scale = scale
        self.conv0 = nn.Sequential(
            conv(in_planes, c, 3, 2, 1),
            conv(c, 2*c, 3, 2, 1),
            )
        self.convblock = nn.Sequential(
            conv(2*c, 2*c),
            conv(2*c, 2*c),
            conv(2*c, 2*c),
            conv(2*c, 2*c),
            conv(2*c, 2*c),
            conv(2*c, 2*c),
        )
        self.conv1 = nn.ConvTranspose2d(2*c, 4, 4, 2, 1)

    def forward(self, x):
        if self.scale != 1:
            x = F.interpolate(x, scale_factor=1. / self.scale, mode="bilinear",
                              align_corners=False)
        x = self.conv0(x)
        x = self.convblock(x)
        x = self.conv1(x)
        flow = x
        if self.scale != 1:
            flow = F.interpolate(flow, scale_factor=self.scale, mode="bilinear",
                                 align_corners=False)
        return flow
```

Figură 3.13 Cod rețea IFNet b)

```python
#reteaua ifnet
class IFNet(nn.Module):
    def __init__(self):
        super(IFNet, self).__init__()
        self.block0 = IFBlock(6, scale=8, c=192)
        self.block1 = IFBlock(10, scale=4, c=128)
        self.block2 = IFBlock(10, scale=2, c=96)
        self.block3 = IFBlock(10, scale=1, c=48)

    def forward(self, x, scale=1.0):
        if scale != 1.0:
            x = F.interpolate(x, scale_factor=scale, mode="bilinear", align_corners=False)
        flow0 = self.block0(x)
        F1 = flow0
        F1_large = F.interpolate(F1, scale_factor=2.0, mode="bilinear", align_corners=False) * 2.0
        warped_img0 = warp(x[:, :3], F1_large[:, :2])
        warped_img1 = warp(x[:, 3:], F1_large[:, 2:4])
        flow1 = self.block1(torch.cat((warped_img0, warped_img1, F1_large), 1))
        F2 = (flow0 + flow1)
        F2_large = F.interpolate(F2, scale_factor=2.0, mode="bilinear", align_corners=False) * 2.0
        warped_img0 = warp(x[:, :3], F2_large[:, :2])
        warped_img1 = warp(x[:, 3:], F2_large[:, 2:4])
        flow2 = self.block2(torch.cat((warped_img0, warped_img1, F2_large), 1))
        F3 = (flow0 + flow1 + flow2)
        F3_large = F.interpolate(F3, scale_factor=2.0, mode="bilinear", align_corners=False) * 2.0
        warped_img0 = warp(x[:, :3], F3_large[:, :2])
        warped_img1 = warp(x[:, 3:], F3_large[:, 2:4])
        flow3 = self.block3(torch.cat((warped_img0, warped_img1, F3_large), 1))
        F4 = (flow0 + flow1 + flow2 + flow3)
        if scale != 1.0:
            F4 = F.interpolate(F4, scale_factor=1 / scale, mode="bilinear", align_corners=False) / scale
        return F4, [F1, F2, F3, F4]
```

Figură 3.15 Cod rețea IFNet c)

Schimbările majore ale acestui model față de cele precedente sunt:

- 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.

3.2.3. 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.

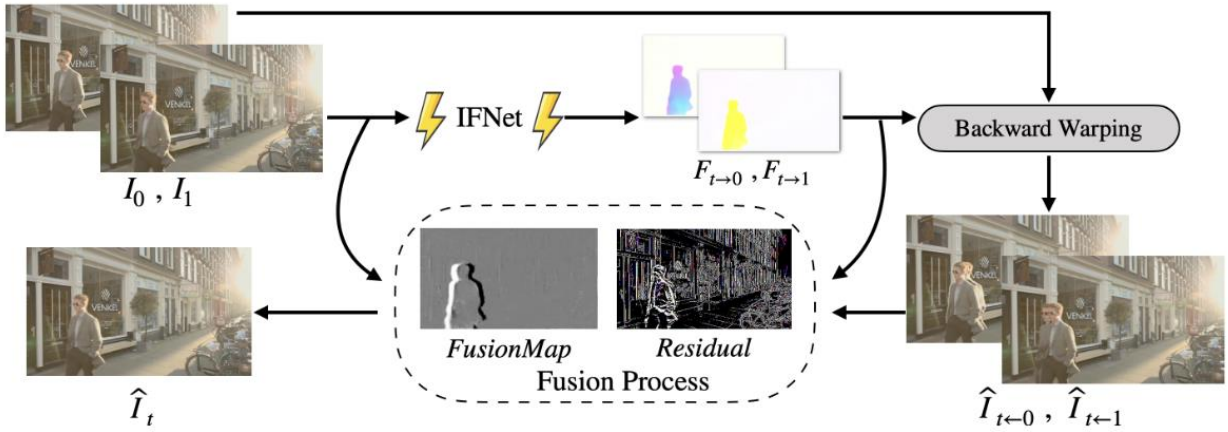
3.2.4. Evoluția interpolării cadrelor video

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.

3.2.5. Modelul RIFE

[47]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 3.16 Ilustrarea funcționalității modelului RIFE

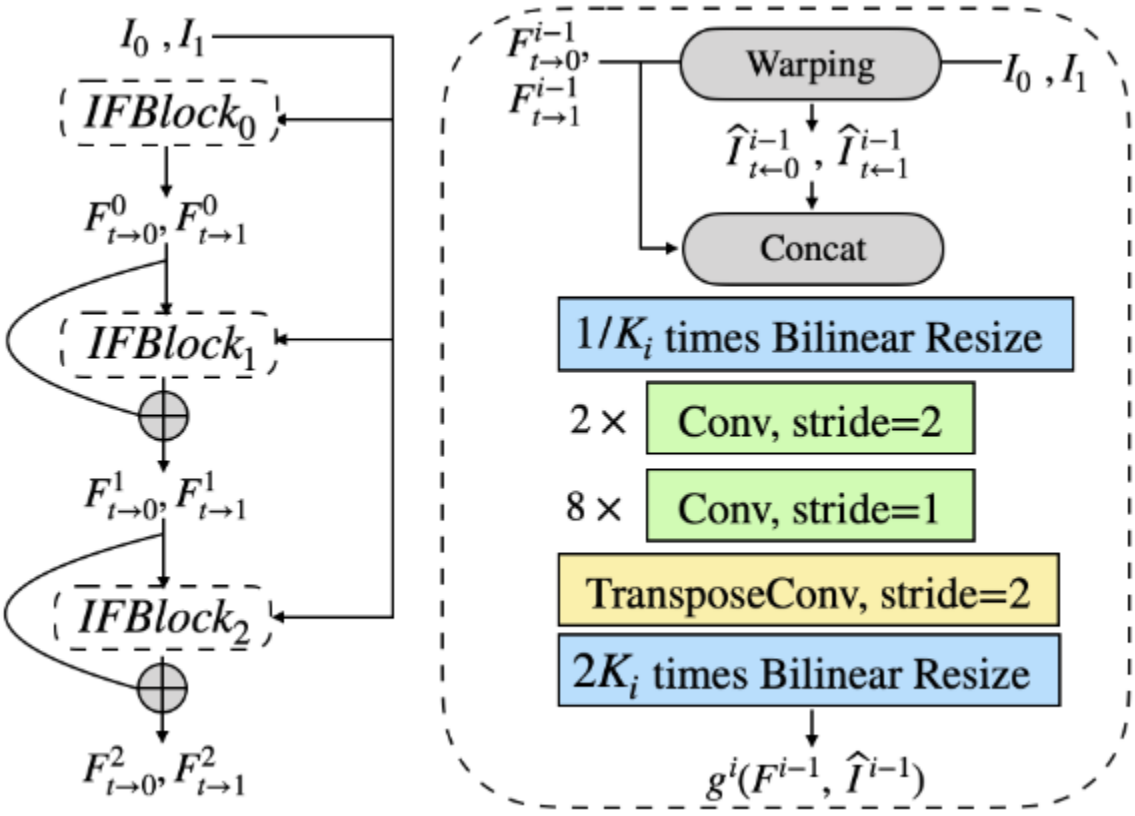
RIFE are două componente majore:

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

3.2.5.1. 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.

[47]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.



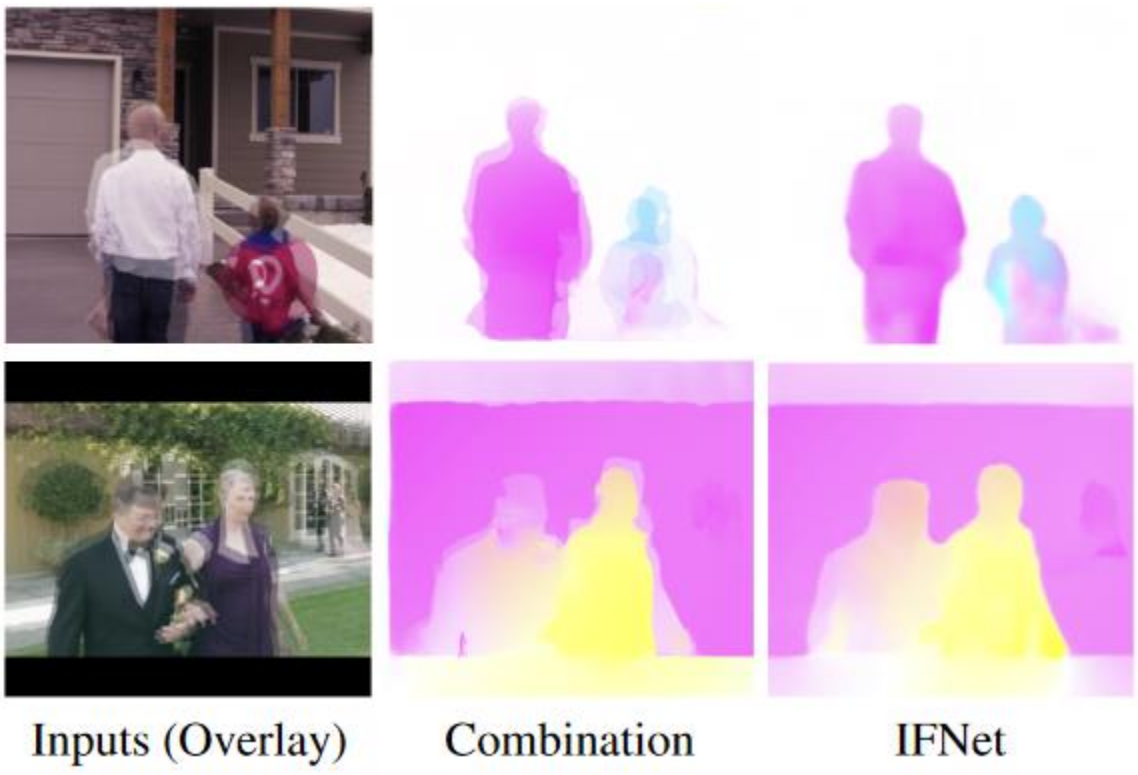
Figură 3.17 Structura modelului IFNet

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.

$$F^i = F^{i-1} + g^i(F^{i-1}, \widehat{I}^{i-1}),$$

Figură 3.18 Formula estimării fluxului intermediar

[47]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 3.19 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.

3.2.5.2. Procesul de fuziune

[47]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.

$$\widehat{I}_t = M \odot \widehat{I}_{t\leftarrow 0} + (1 - M) \odot \widehat{I}_{t\leftarrow 1} + \Delta,$$

Figură 3.20 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.

```
def conv(in_planes, out_planes, kernel_size=3, stride=1, padding=1, dilation=1):
    return nn.Sequential(
        nn.Conv2d(in_planes, out_planes, kernel_size=kernel_size, stride=stride,
                  padding=padding, dilation=dilation, bias=True),
        nn.PReLU(out_planes)
    )


def deconv(in_planes, out_planes, kernel_size=4, stride=2, padding=1):
    return nn.Sequential(
        torch.nn.ConvTranspose2d(in_channels=in_planes, out_channels=out_planes,
                                 kernel_size=4, stride=2, padding=1, bias=True),
        nn.PReLU(out_planes)
    )

def conv_woact(in_planes, out_planes, kernel_size=3, stride=1, padding=1, dilation=1):
    return nn.Sequential(
        nn.Conv2d(in_planes, out_planes, kernel_size=kernel_size, stride=stride,
                  padding=padding, dilation=dilation, bias=True),
    )
```

Figură 3.21 Cod procesul de fuziune a)

```
class Conv2(nn.Module):
    def __init__(self, in_planes, out_planes, stride=2):
        super(Conv2, self).__init__()
        self.conv1 = conv(in_planes, out_planes, 3, stride, 1)
        self.conv2 = conv(out_planes, out_planes, 3, 1, 1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        return x

c = 32
```

Figură 3.23 Cod procesul de fuziune b)

```
class ContextNet(nn.Module):
    def __init__(self):
        super(ContextNet, self).__init__()
        self.conv0 = Conv2(3, c)
        self.conv1 = Conv2(c, c)
        self.conv2 = Conv2(c, 2*c)
        self.conv3 = Conv2(2*c, 4*c)
        self.conv4 = Conv2(4*c, 8*c)

    def forward(self, x, flow):
        x = self.conv0(x)
        x = self.conv1(x)
        flow = F.interpolate(flow, scale_factor=0.5, mode="bilinear", align_corners=False) * 0.5
        f1 = warp(x, flow)
        x = self.conv2(x)
        flow = F.interpolate(flow, scale_factor=0.5, mode="bilinear",
                             align_corners=False) * 0.5
        f2 = warp(x, flow)
        x = self.conv3(x)
        flow = F.interpolate(flow, scale_factor=0.5, mode="bilinear",
                             align_corners=False) * 0.5
        f3 = warp(x, flow)
        x = self.conv4(x)
        flow = F.interpolate(flow, scale_factor=0.5, mode="bilinear",
                             align_corners=False) * 0.5
        f4 = warp(x, flow)
        return [f1, f2, f3, f4]
```

Figură 3.22 Cod procesul de fuziune c)

```
class FusionNet(nn.Module):
    def __init__(self):
        super(FusionNet, self).__init__()
        self.conv0 = Conv2(10, c)
        self.down0 = Conv2(c, 2*c)
        self.down1 = Conv2(4*c, 4*c)
        self.down2 = Conv2(8*c, 8*c)
        self.down3 = Conv2(16*c, 16*c)
        self.up0 = deconv(32*c, 8*c)
        self.up1 = deconv(16*c, 4*c)
        self.up2 = deconv(8*c, 2*c)
        self.up3 = deconv(4*c, c)
        self.conv = nn.ConvTranspose2d(c, 4, 4, 2, 1)

    def forward(self, img0, img1, flow, c0, c1, flow_gt):
        warped_img0 = warp(img0, flow[:, :2])
        warped_img1 = warp(img1, flow[:, 2:4])
        if flow_gt == None:
            warped_img0_gt, warped_img1_gt = None, None
        else:
            warped_img0_gt = warp(img0, flow_gt[:, :2])
            warped_img1_gt = warp(img1, flow_gt[:, 2:4])
        x = self.conv0(torch.cat((warped_img0, warped_img1, flow), 1))
        s0 = self.down0(x)
        s1 = self.down1(torch.cat((s0, c0[0], c1[0]), 1))
        s2 = self.down2(torch.cat((s1, c0[1], c1[1]), 1))
        s3 = self.down3(torch.cat((s2, c0[2], c1[2]), 1))
        x = self.up0(torch.cat((s3, c0[3], c1[3]), 1))
        x = self.up1(torch.cat((x, s2), 1))
        x = self.up2(torch.cat((x, s1), 1))
        x = self.up3(torch.cat((x, s0), 1))
        x = self.conv(x)
        return x, warped_img0, warped_img1, warped_img0_gt, warped_img1_gt
```

Figură 3.24 Cod procesul de fuziune d)

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.

3.2.6. 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.

[47]Funcția de pierdere distilată este definită astfel:

$$\mathcal{L}_{dis} = ||F_{t\to 0} - F_{t\to 0}^{Leak}||_1 + ||F_{t\to 1} - F_{t\to 1}^{Leak}||_1.$$

Figură 3.25 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.

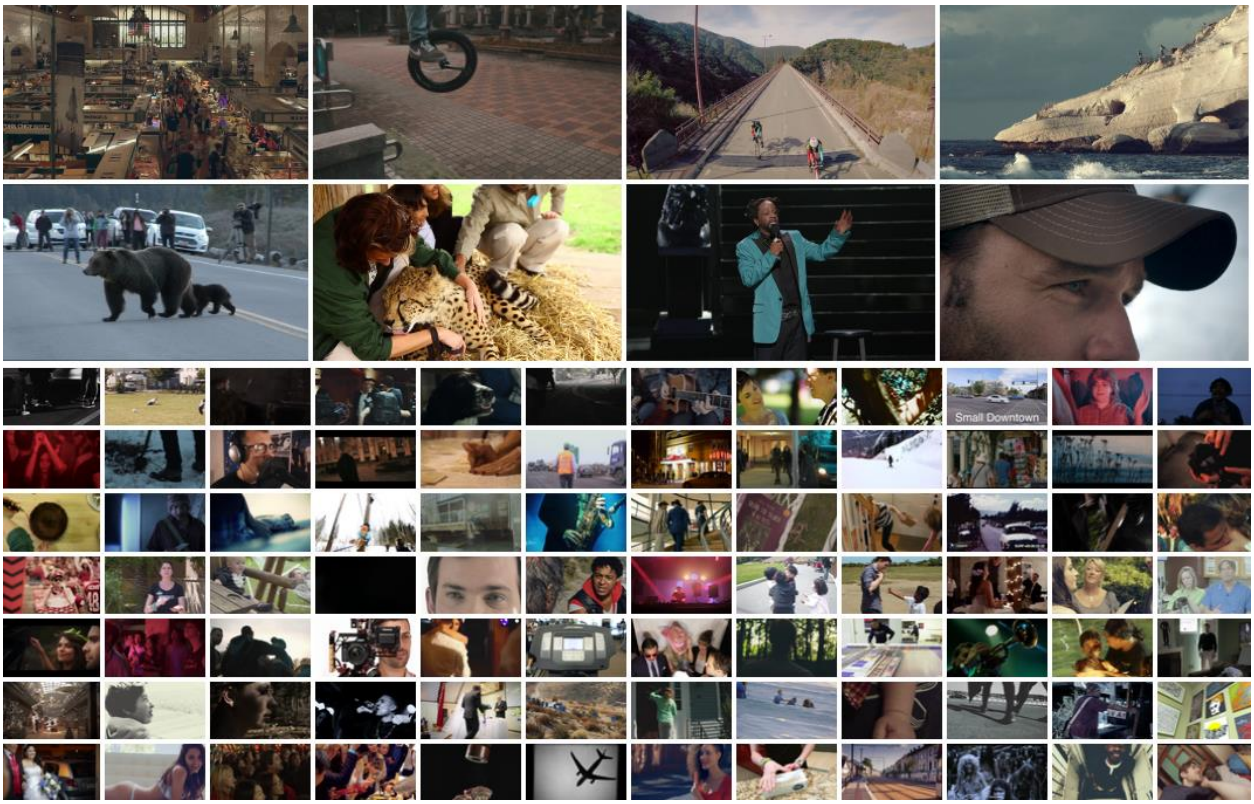
3.2.7. Pregătirea setului de date

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.

[48]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.

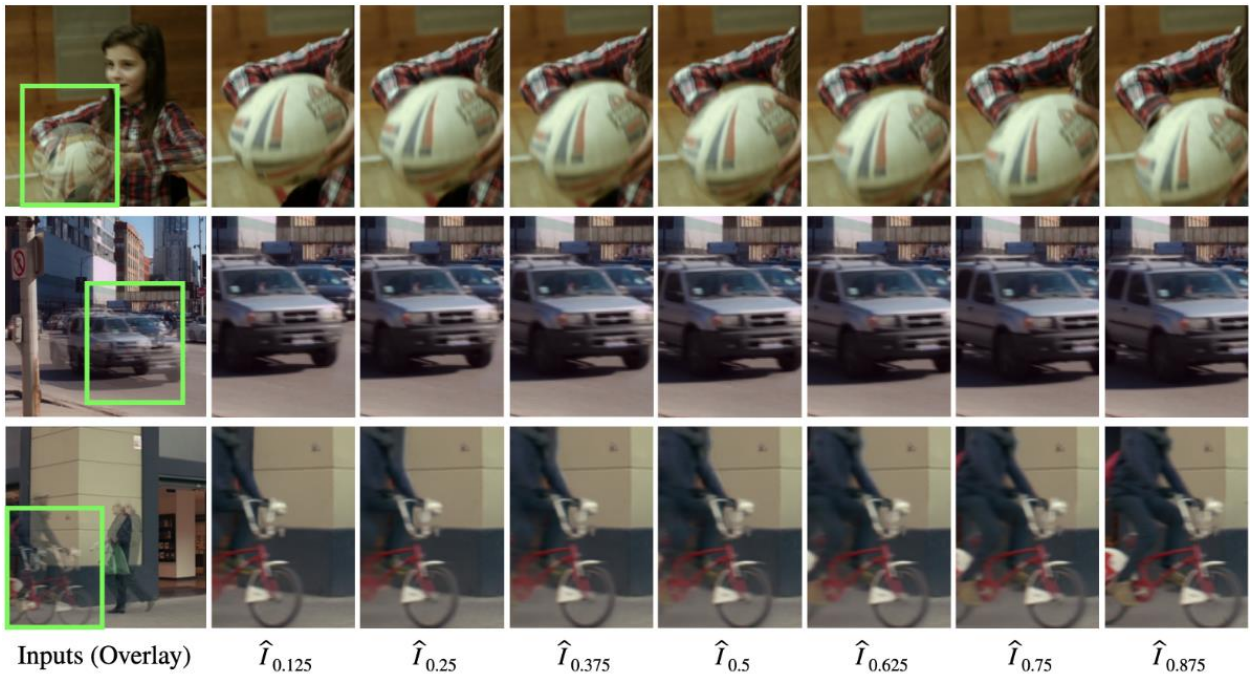


Figură 3.26 Set de date Vimeo-90k

3.2.8. Generarea de cadre multiple

Pentru interpolarea multiplă de cadre la timpuri diferite putem aplica algoritmul RIFE într-un mode recursiv. Adică, folosind două cadre de intrare, aplicăm algoritmul RIFE pentru a face rost de cadrul intermediar la timpul t=0.5. Introducem apoi ca date de intrare cadrul la timpul t=0 și cadrul la timpul t=0.5 pentru a obține cadrul intermediar la timpul t=0.25. Putem repeta acest proces recursiv pentru a face rost de mai multe cadre intermediare. Pentru a demonstra această abilitate, în imaginea de mai jos sunt exemplificate rezultate vizuale pentru setări duble, cvadruple, etc pe imagini cu mișcări mari culese din setul de date Vimeo90k.

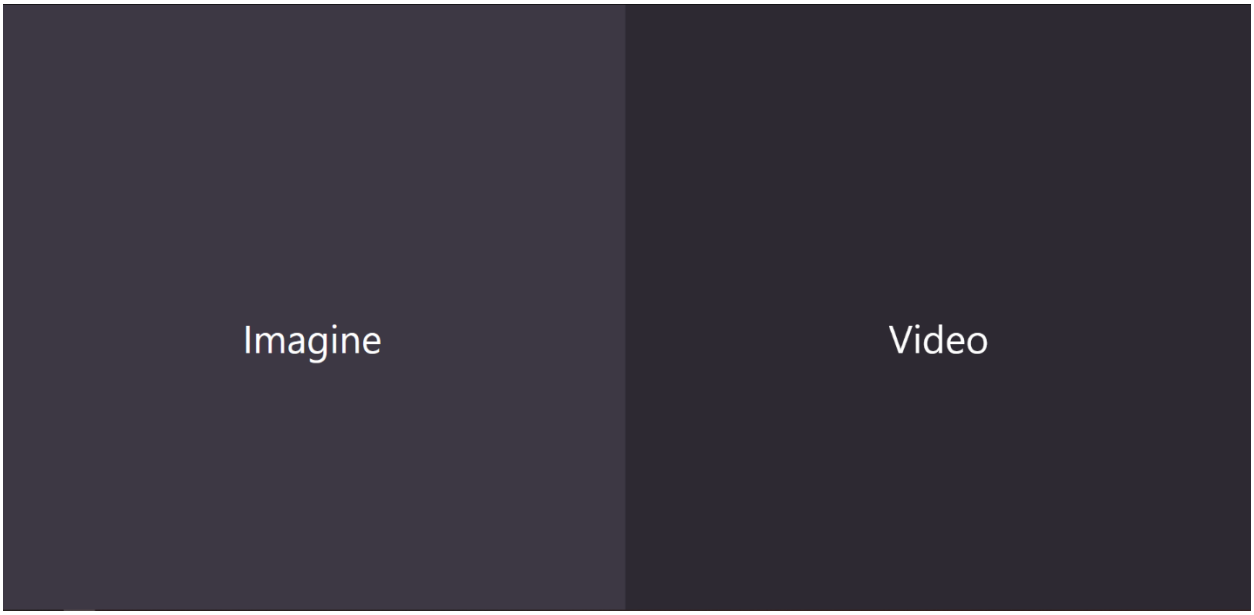
Putem observa cum RIFE produce cu succes cadre clare și o mișcare fluidă



Figură 3.27 Interpolarea cadrelor multiple folosind algoritmul RIFE în mod recursiv

4. SCENARII DE UTILIZARE ALE SISTEMULUI

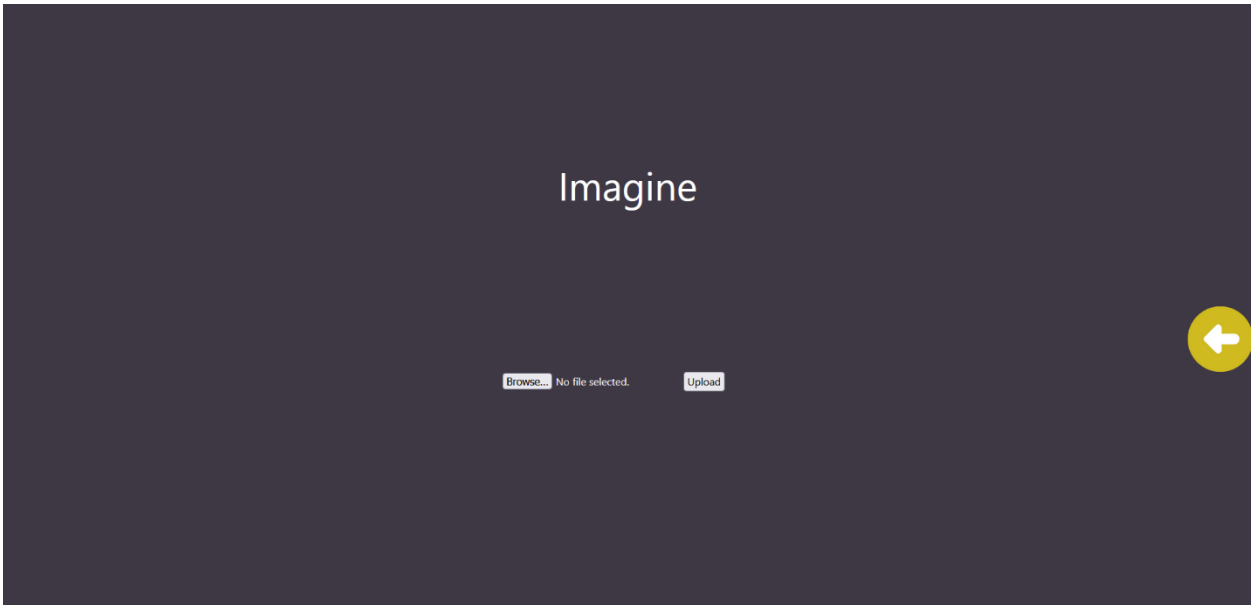
În acest capitol voi prezenta câteva scenarii de utilizare pentru rețelele neuronale create. În cazul folosirii oricărei dintre cele două rețele, utilizatorul este întâmpinat de aplicația web. Acolo are posibilitatea de a alege care rețea neuronală își dorește să folosească.



Figură 4.1 înterfața web

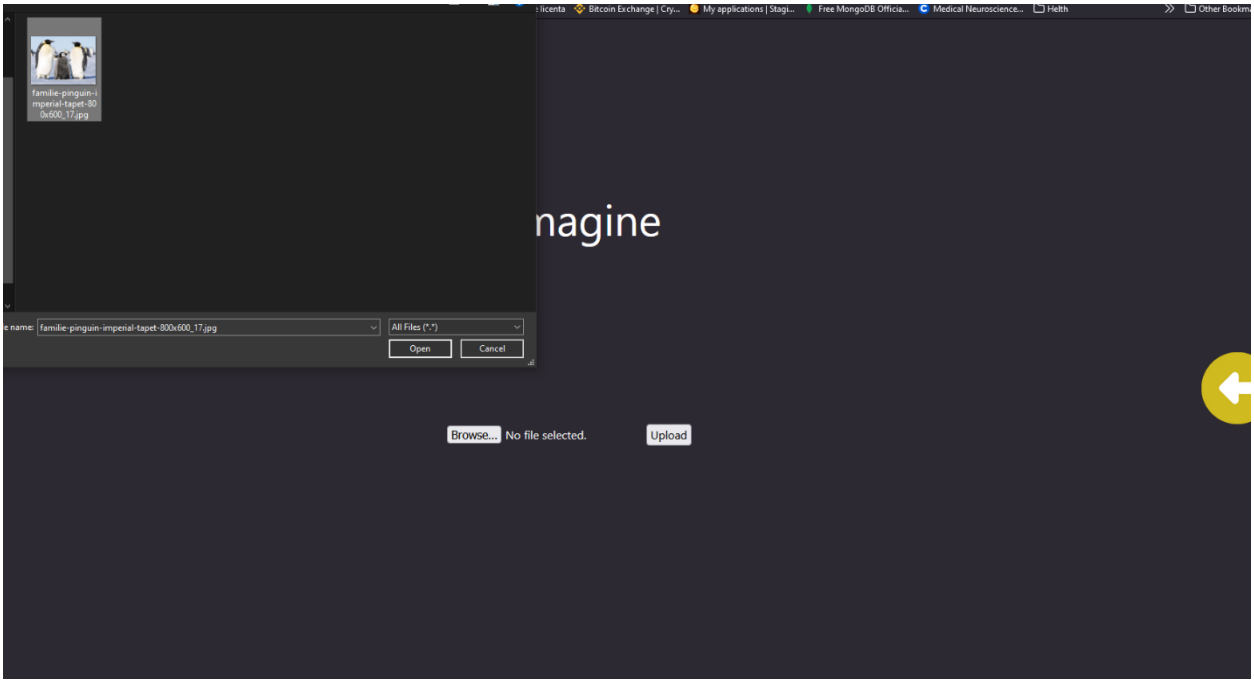
După cum am prezentat și mai sus, utilizatorul este întâmpinat cu alegerea de a încărca o imagine, lucru care ar duce la trecerea acesteia prin prima rețea neuronală. Sau încărcarea unui videoclip, lucru care duce la trecerea lui prin a doua rețea neuronală. Conversia fișierului media este computată pe server, astfel că utilizatorul nu are nevoie să își facă griji pentru resurse.

Odată ce utilizatorul alege operațiunea dorită, tot ecranul va fi acoperit de partea aleasă și poate mai departe să încarce fișierul media dorit. Săgeata galbenă reprezintă butonul de înapoi. Odată apăsat se întoarce la pagina principală unde se alege operațiunea. Folosind butonul de Browse, își alege din memoria locală fișierul pe care dorește să îl încarce. Apoi butonul de Upload încarcă fișierul care va trece prin rețeaua neuronală și pornește, totodată și procesul de conversie. Odată ce procesul este încheiat, fișierul rezultat va fi întors pentru salvare.



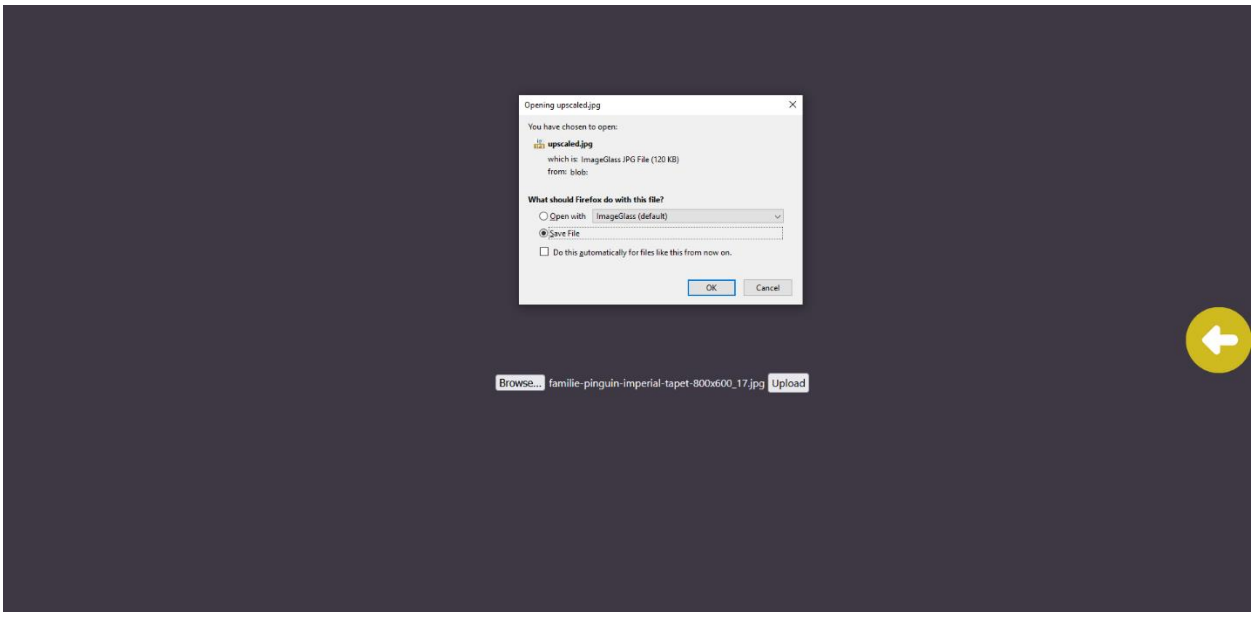
Figură 4.3 Interfața web, operațiunea aleasă

Următorul pas este alegerea fișierului dorit și încărcarea lui prin butoanele Browse și Upload.



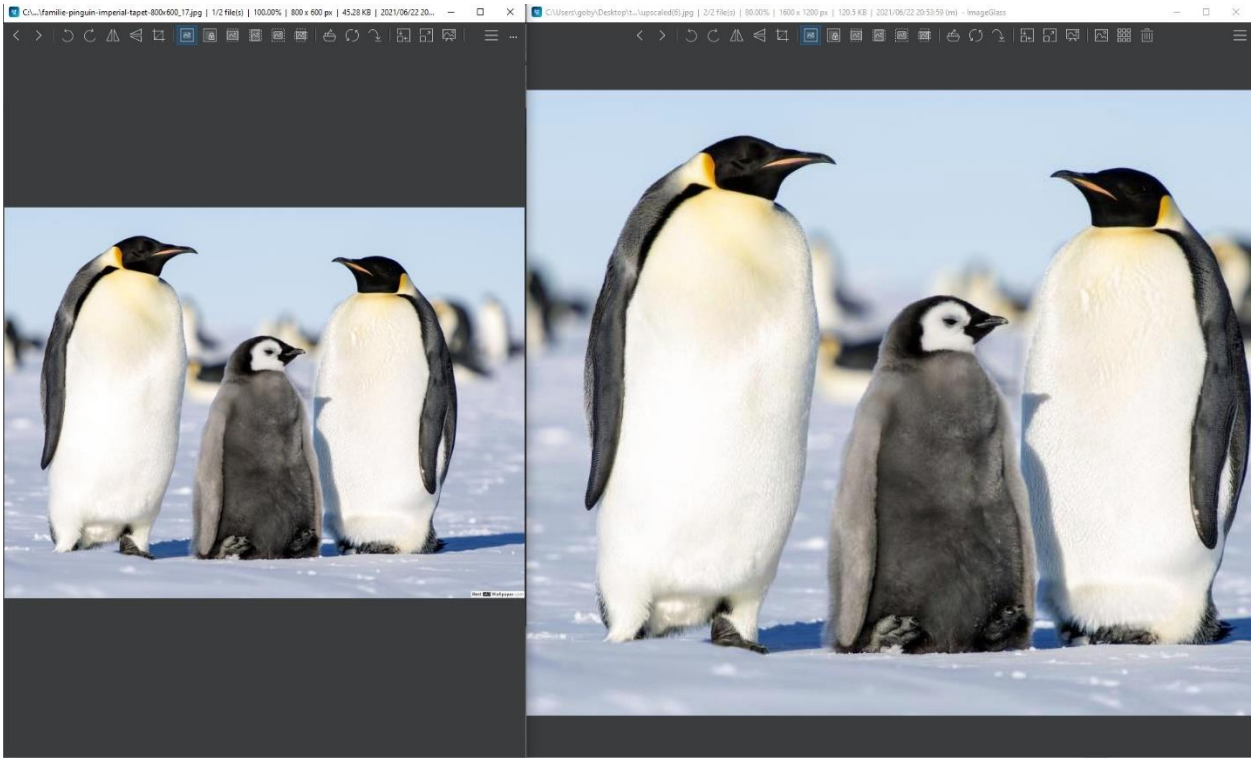
Figură 4.2 Alegerea fișierului dorit

Odată ce fișierul este încărcat și se apasă butonul de „Upload”, procesul de machine learning este pornit și în scurt timp utilizatorul va avea opțiunea de a descărca fișierul rezultat.



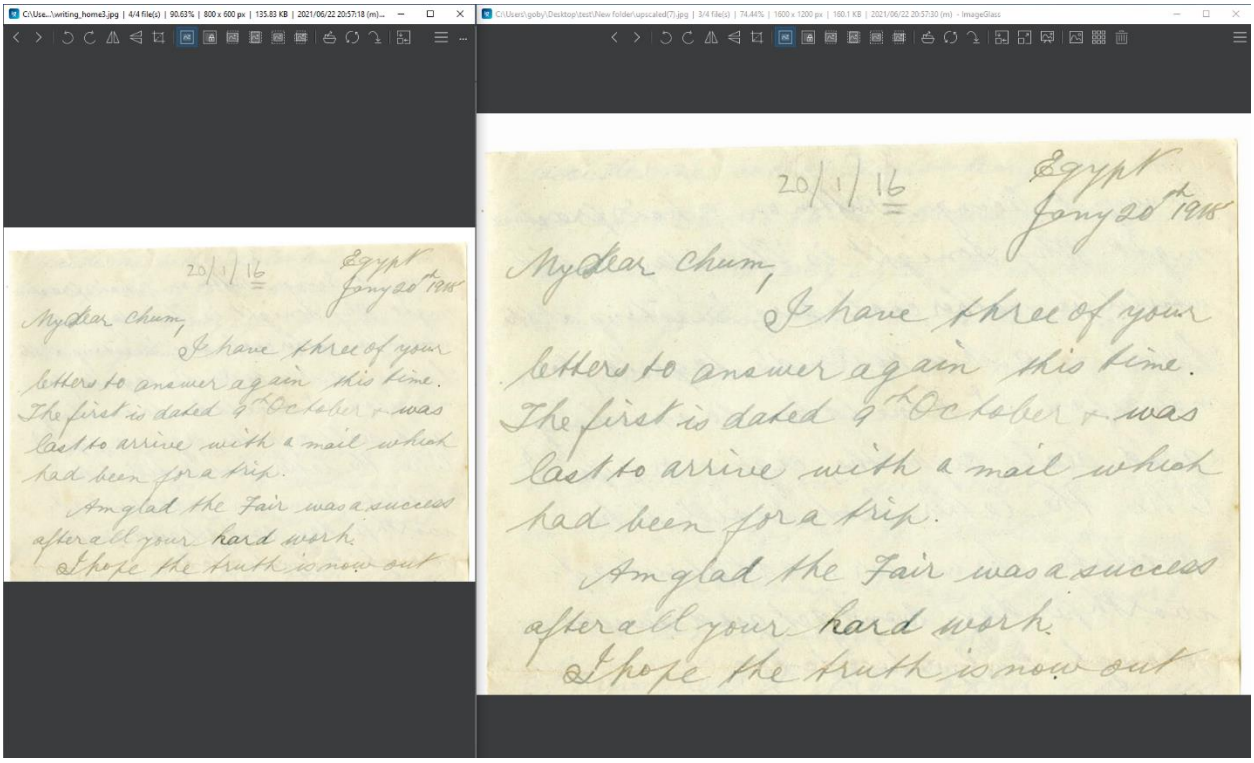
Figură 4.4 Întoarcerea fișierului pentru descărcare

Pentru acest exemplu poza folosită este o poză cu o familie de pinguini, iar în imaginea de mai jos voi atașa diferența dinainte și după.



Figură 4.5 Comparație rezultat înainte si după conversie

Modelul funcționează pe mai multe tipuri de imagini care conțin mai multe tipuri de informații. Performanțele rezultate sunt foarte bun și atunci când introducem o imagine care conține mult scris. Lucru care are deja o mulțime de aplicări în mai multe domenii unde transmisia de date este foarte importantă.



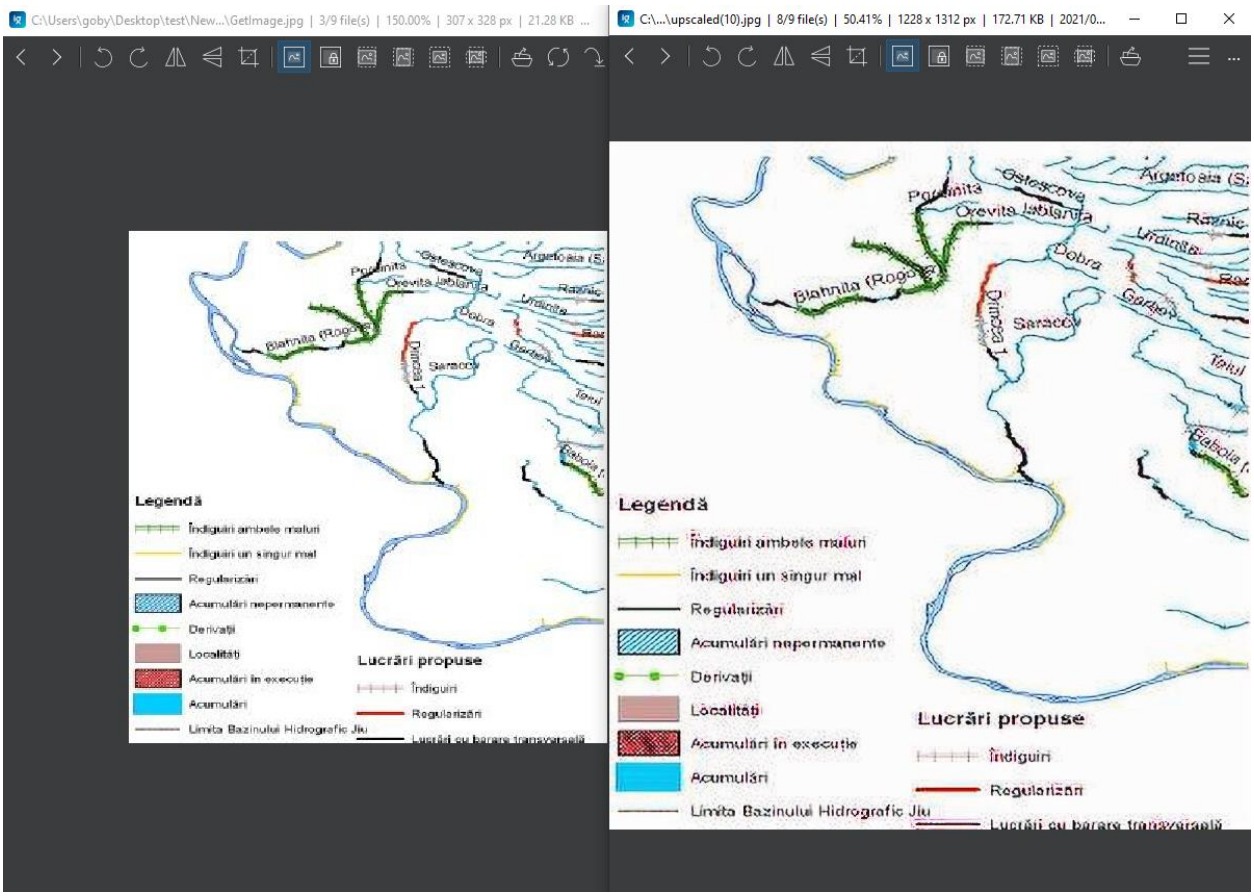
Figură 4.6 Comparație rezultate înainte și după conversie scris

Într-un scenariu în care atenția la detalii poate fi foarte semnificativă, cum ar fi în domeniul criminalisticii, unde singura bucată de dovadă este o înregistrare sau o poză făcută de o cameră de supraveghere. Calitatea acelei camere este în general foarte slabă. Aici poate intervenii modelul și poate îmbunătății calitatea imaginii.

În industria jocurilor, super rezoluția este un lucru căutat și implementat de ani de zile. Calcularea unei imagini superioare poate îmbunătății experiența utilizatorului în momentul jocului.

De asemenea, în industria de hidrotehnică, persoanele care lucrează în domeniu au de a face cu interacțiunea cu hărți hidrotehnice. În aceeași ordine de idei și persoanele care lucrează în domeniul geodeziei au de a face cu cititul, interpretatul și folositul a multor hărți, unde nu toate au o calitate a imaginii superioare. Iar încercarea de a înțelege unde se află anumite lucruri și ce se află acolo poate deveni o provocare mare.

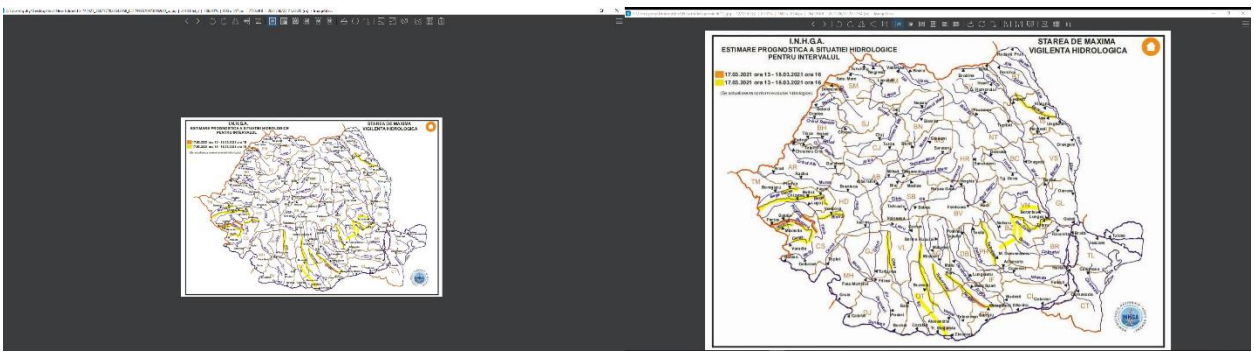
Astfel modelul se descurcă foarte bine și la îmbunătățirea calității unei hărți și a evidențierii detaliilor care sunt extrem de importante. Exemplu de rezultat al unei hărți trecute prin rețea în imaginea de mai jos.



Figură 4.7 Comparație înainte și după conversie hartă

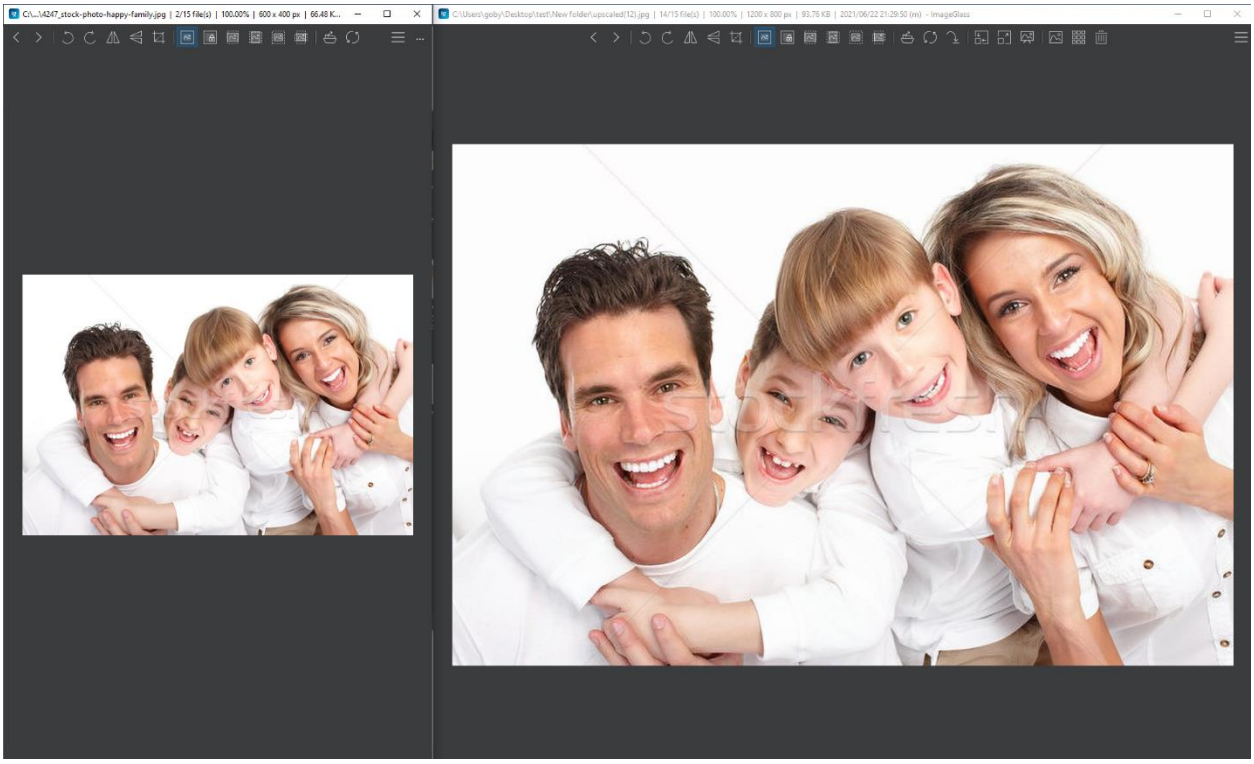
Comparația de mai sus reprezintă diferența dintre imaginea originală și imaginea introdusă de 2 ori consecutiv în rețea. Astfel ne putem baza și pe introducerea multiplă a unei imagini pentru a mării calitatea de mai multe ori.

În următoarea poză se află o comparație reală între mărimile complete ale imaginilor înainte și după conversie.



Figură 4.8 Comparație între rezoluțiile pozelor înainte și după conversie

O altă utilizare foarte importantă este în domeniul familiei. Oamenii vor să păstreze amintiri de o calitate cât mai bună prin intermediul fotografiilor și videoclipurilor. Fotografiile făcute acum mulți ani pot avea o calitate mai slabă decât cele mai noi. Folosind modelul neuronal, amintirile pot fi reconstituite și păstrate la calitatea pe care le merită.



Figură 4.9 Comparație poze cu familia

Diferența în detaliu este vizibilă cu ochiul liber atunci când ne uităm mai aproape

Figură 4.10 Comparație detaliu înainte și după conversie

Pentru a doua rețea neuronală nu pot arăta exemple cu videoclipuri. Însă modelul este în stare să genereze mai multe cadre intermediare pentru două imagini consecutive, creând astfel un mic clip.

În imaginea următoare sunt cadrele de intrare în rețea, iar modelul va umple 15 cadre intermediare.

Figură 4.13 Cadrele de intrare pentru rețea

Figură 4.20 Cadru intermediar 1

Figură 4.12 Cadru intermediar 2

Figură 4.19 Cadru intermediar 3

Figură 4.18 Cadru intermediar 4

Figură 4.17 Cadru intermediar 5

Figură 4.16 Cadru intermediar 6

Figură 4.15 Cadru intermediar 7

Figură 4.14 Cadru intermediar 8

Figură 4.11 Cadru intermediar 9

Figură 4.26 Cadru intermediar 10

Figură 4.25 Cadru intermediar 11

Figură 4.24 Cadru intermediar 12

Figură 4.23 Cadru intermediar 13

Figură 4.22 Cadru intermediar 14

Figură 4.21 Cadru intermediar 15

Rețeaua funcționează de asemenea foarte bine și cu obiecte mari aflate în mișcare.

Voi prezenta în imaginile următoare ce a reușit să creeze modelul urmărind un vehicul în mișcare.

Figură 4.27 Cadrele de intrare

În imaginea de deasupra sunt ilustrate cadrele introduse în rețea.

În imaginile următoare sunt prezentate cadrele intermediare create de model, reușind să păstreze un nivel de detaliu și să creeze fluiditate.

Figură 4.42 Cadru intermediar 1

Figură 4.41 Cadru intermediar 2

Figură 4.40 Cadru intermediar 3

Figură 4.39 Cadru intermediar 4

Figură 4.38 Cadru intermediar 5

Figură 4.37 Cadru intermediar 6

Figură 4.36 Cadru intermediar 7

Figură 4.35 Cadru intermediar 8

Figură 4.34 Cadru intermediar 9

Figură 4.33 Cadru intermediar 10

Figură 4.32 Cadru intermediar 11

Figură 4.31 Cadru intermediar 12

Figură 4.30 Cadru intermediar 13

Figură 4.29 Cadru intermediar 14

Figură 4.28 Cadru intermediar 15

CONCLUZII

In concluzia acestui proiect ambele rețele antrenate pot aduce o mulțime de beneficii în multe domenii diferite și pot oferi confort și stabilitate. Am reușit să aduc ambele modele într-un stagiu satisfăcător în care am atins toate punctele pe care mi le-am propus și am învățat o mulțime de concepte noi pe parcurs.

Fiecare dintre cele două rețele pot fi folosite singure, și este, de fapt, mult mai fiabil să fie așa. Am dorit să ofer o posibilitate gratuită de folosire a unor tehnologii fără a îmi face griji de resursele pe care le am.

Am demonstrat aplicările modelelor în diferite domenii și am prezentat rezultatele antrenamentului. Capacitatea celor două rețele mi-a întrecut așteptările prin acuratețea obținută și eficiența în situații reale.

Prin intermediul acestui proiect, m-am dezvoltat pe plan profesional și am dobândit mai multe cunoștințe și înțelegeri despre domeniul abordat. Am învățat ce înseamnă să lucrezi la un proiect de scară largă și cum ar trebui să îmi administrez timpul pentru a duce la bun sfârșit ce mi-am propus.

Tehnologiile pe care le-am învățat și folosit consider că sunt ceea ce se caută în piață și că este un pas în direcția bună către angajare.

Planuri de viitor

Pentru optimizarea și mai departe a modelelor, în primul rând pentru rețeaua care mărește rezoluția imaginilor, ar trebui antrenată mai mult pe o placă video mai bună pentru a ajunge la un rezultat cât mai apropiat de perfecțiune. Aș antrena rețeaua pe mai multe poze cu detalii mici, cum ar fi hărți și texte deoarece prezintă rezultate mai puțin satisfăcătoare pe acest plan.

Pentru cea de-a doua rețea, modelul ar trebui, de asemenea, antrenat mai mult. Deocamdată, luând in considerare resursele hardware de care dispune calculatorul meu nu este în stare să genereze încă videoclipuri. Un aspect de viitor ar fi implementarea acestui lucru.

Pentru platforma web, se are în vedere tranziția pe un server dedicat care să dispună de resursele necesare pentru a produce rezultate la un nivel comercial pentru orice cerință a unui utilizator.

BIBLIOGRAFIE

[1] https://sitn.hms.harvard.edu/flash/2019/artificial-intelligence-in-medicine-applications-implications-and-limitations/. [Accesat 14 Iunie 2021].

[2] „Artificial Intelligence (AI),” [Interactiv]. Available: https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp. [Accesat 14 Iunie 2021].

[3] „History of self-driving cars,” [Interactiv]. Available: https://en.wikipedia.org/wiki/History_of_self-driving_cars. [Accesat 14 Iunie 2021].

[4] „Autopilot and Full Self-Driving Capability,” [Interactiv]. Available: https://www.tesla.com/support/autopilot. [Accesat 14 Iunie 2021].

[5] „How Capable Is Tesla's Autopilot Driver-Assist System? We Put It to the Test,” [Interactiv]. Available: https://www.caranddriver.com/news/a35839385/tesla-autopilot-full-self-driving-autonomous-capabilities-tested-explained/. [Accesat 14 Iunie 2021].

[6] „Soon Singapore can hail out to nuTonomy's 'robo-taxi',” [Interactiv]. Available: https://techseen.com/singapore-nutonomy-robo-taxi/. [Accesat 14 Iunie 2021].

[7] „Robotaxi,” [Interactiv]. Available: https://en.wikipedia.org/wiki/Robotaxi. [Accesat 14 Iunie 2021].

[8] „AlphaGo,” [Interactiv]. Available: https://deepmind.com/research/case-studies/alphago-the-story-so-far. [Accesat 15 Iunie 2021].

[9] „AlphaGo,” [Interactiv]. Available: https://en.wikipedia.org/wiki/AlphaGo. [Accesat 15 Iunie 2021].

[10] „Lee Sedol,” [Interactiv]. Available: https://en.wikipedia.org/wiki/Lee_Sedol. [Accesat 15 Iunie 2021].

[11] „AlphaZero Crushes Stockfish In New 1,000-Game Match,” [Interactiv]. Available: https://www.chess.com/news/view/updated-alphazero-crushes-stockfish-in-new-1-000-game-match. [Accesat 15 Iunie 2021].

[12] „OpenAI Five,” [Interactiv]. Available: https://openai.com/projects/five/. [Accesat 15 Iunie 2021].

[13] „OpenAI Five,” [Interactiv]. Available: https://openai.com/blog/openai-five/. [Accesat 15 Iunie 2021].

[14] „OpenAI Five,” [Interactiv]. Available: https://en.wikipedia.org/wiki/OpenAI_Five. [Accesat 15 Iunie 2021].

[15] „Difference between Artificial intelligence and Machine learning,” [Interactiv]. Available: https://www.javatpoint.com/difference-between-artificial-intelligence-and-machine-learning. [Accesat 16 Iunie 2021].

[16] „Machine Learning,” [Interactiv]. Available: https://www.investopedia.com/terms/m/machine-learning.asp. [Accesat 16 Iunie 2021].

[17] „Neural Network,” [Interactiv]. Available: https://www.investopedia.com/terms/n/neuralnetwork.asp. [Accesat 16 Iunie 2021].

[18] „https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/,” [Interactiv]. Available: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/. [Accesat 16 Iunie 2021].

[19] „Neural Networks Bias And Weights,” [Interactiv]. Available: https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da. [Accesat 16 Iunie 2021].

[20] „Activation function,” [Interactiv]. Available: https://en.wikipedia.org/wiki/Activation_function. [Accesat 16 Iunie 2021].

[21] „Understanding Loss Functions in Machine Learning,” [Interactiv]. Available: https://www.section.io/engineering-education/understanding-loss-functions-in-machine-learning/. [Accesat 16 Iunie 2021].

[22] „Lecture 9.2 — Neural Networks Learning | Backpropagation Algorithm — [Machine Learning | Andrew Ng],” [Interactiv]. Available: https://www.youtube.com/watch?v=x_Eamf8MHwU. [Accesat 16 Iunie 2021].

[23] „A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,” [Interactiv]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accesat 16 Iunie 2021].

[24] „Supervised Learning,” [Interactiv]. Available: https://www.ibm.com/cloud/learn/supervised-learning. [Accesat 17 Iunie 2021].

[25] „Reinforcement learning,” [Interactiv]. Available: https://www.geeksforgeeks.org/what-is-reinforcement-learning/. [Accesat 17 Iunie 2021].

[26] „Unsupervised Machine Learning: What is, Algorithms, Example,” [Interactiv]. Available: https://www.guru99.com/unsupervised-machine-learning.html. [Accesat 17 Iunie 2021].

[27] „A Gentle Introduction to Generative Adversarial Networks (GANs),” [Interactiv]. Available: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/. [Accesat 17 Iunie 2021].

[28] „Introduction to Generative Adversarial Networks (GANs),” [Interactiv]. Available: https://medium.com/@kraken2404/introduction-to-generative-adversarial-networks-gans-89095151cd3a. [Accesat 17 Iunie 2021].

[29] „Documentation for Visual Studio Code,” [Interactiv]. Available: https://code.visualstudio.com/docs. [Accesat 18 Iunie 2021].

[30] „Visual Studio Code,” [Interactiv]. Available: https://en.wikipedia.org/wiki/Visual_Studio_Code. [Accesat 18 Iunie 2021].

[31] „What is Python? Executive Summary,” [Interactiv]. Available: https://www.python.org/doc/essays/blurb/. [Accesat 18 Iunie 2021].

[32] „Python (programming language),” [Interactiv]. Available: https://en.wikipedia.org/wiki/Python_(programming_language). [Accesat 18 Iunie 2021].

[33] „PyTorch,” [Interactiv]. Available: https://en.wikipedia.org/wiki/PyTorch. [Accesat 18 Iunie 2021].

[34] „NumPy,” [Interactiv]. Available: https://en.wikipedia.org/wiki/NumPy. [Accesat 18 Iunie 2021].

[35] „pandas (software),” [Interactiv]. Available: https://en.wikipedia.org/wiki/Pandas_(software). [Accesat 18 Iunie 2021].

[36] „Matplotlib,” [Interactiv]. Available: https://en.wikipedia.org/wiki/Matplotlib. [Accesat 18 Iunie 2021].

[37] „Anaconda (Python distribution),” [Interactiv]. Available: https://en.wikipedia.org/wiki/Anaconda_(Python_distribution). [Accesat 18 Iunie 2021].

[38] „HTML,” [Interactiv]. Available: https://en.wikipedia.org/wiki/HTML. [Accesat 18 Iunie 2021].

[39] „CSS,” [Interactiv]. Available: https://en.wikipedia.org/wiki/CSS. [Accesat 18 Iunie 2021].

[40] „JavaScript,” [Interactiv]. Available: https://en.wikipedia.org/wiki/JavaScript. [Accesat 18 Iunie 2021].

[41] „Flask (web framework),” [Interactiv]. Available: https://en.wikipedia.org/wiki/Flask_(web_framework). [Accesat 18 Iunie 2021].

[42] *Note de Curs "Analiza sistemelor informaționale și proiectarea sistemelor informatice".*

[43] L. T. F. H.́. J. C. A. C. A. A. A. A. T. J. T. Z. W. W. S. Christian Ledig, „Photo-Realistic Single Image Super-Resolution Using a Generative AdversarialNetwork,” p. 19.

[44] „Large-scale CelebFaces Attributes (CelebA) Dataset,” [Interactiv]. Available: https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html. [Accesat 20 Iunie 2021].

[45] K. Ahirwar, Generative Adversarial Networks Projects: Build next-generation generative models using TensorFlow and Keras, 2019.

[46] „PixelShuffle,” [Interactiv]. Available: https://pytorch.org/docs/stable/generated/torch.nn.PixelShuffle.html. [Accesat 21 Iunie 2021].

[47] Z. H. Z. H. S. Z. I. University, „RIFE: Real-Time Intermediate Flow Estimation for Video Frame Interpolation,” p. 10, 2021.

[48] „Video Enhancement with Task-Oriented Flow,” [Interactiv]. Available: http://toflow.csail.mit.edu/. [Accesat 22 Iunie 2021].