Universitatea POLITEHNICA din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Inferența miscării micro-vehiculelor aeriene din secvente video

Proiect de Diplomă

Prezentat ca cerință parțială pentru obținerea
titlului de *Inginer*
în domeniul *Electronică, Telecomunicaţii şi Tehnologia Informaţiei*
programul de studii *MON*

Conducător științific
Ș.L. dr. ing. Cristian Constantin DAMIAN

Absolvent
Nicușor-Cristian SOCOL

Anul 2023

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Program de studiu **MON**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ

a studentului **SOCOL G. Nicuşor-Cristian, 445E**

1. Titlul temei: Inferența miscării micro-vehiculelor aeriene din secvente video

2. Descrierea temei și a contribuției personale a studentului (în afara părții de documentare):
Studentul va implementa o rețea de învățare profundă utilizând o bibliotecă specifică (tensorflow, pytorch), va antrena și testa rețeaua pe un set de date cu secvențe de imagini adnotate cu poziția și unghiul camerei video.
Studentul va realiza o analiză a performațelor rețelei în funcție de structura rețeiei, funcția de eroare și hiperparametri rețelei pentru a adapta rețeaua la secvențe video ale micro-vehiculelor aeriene.

3. Discipline necesare pt. proiect:
Recunoașterea formelor și inteligența artificală, Procesarea imaginilor, Programarea calculatoarelor

4. Data înregistrării temei: 2023-01-20 13:40:45

Conducător(i) lucrare,
Ș.L. dr. ing. Cristian Constantin DAMIAN

Student,
SOCOL G. Nicuşor-Cristian

Director departament,

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **2b82e2a051**

Declaraţie de onestitate academică

Prin prezenta declar că lucrarea cu titlul *Inferența mișcării micro-vehiculelor aeriene din secvente video*, prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* în domeniul Inginerie Electronică şi Telecomunicaţii/ Calculatoare şi Tehnologia Informaţiei, programul de studii *MON* este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor şi măsurătorilor pe care le prezint ca fiind făcute de mine, precum şi metodele prin care au fost obţinute, sunt reale şi provin din respectivele simulări, experimente şi măsurători. Înţeleg că falsificarea datelor şi rezultatelor constituie fraudă şi se sancţionează conform regulamentelor în vigoare.

Bucureşti, Iulie 2023.

Absolvent: Nicușor-Cristian SOCOL

Cuprins

Lista figurilor

Lista tabelelor

Lista acronimelor

CNN = Convolutional Neural Networks (Rețea neuronală convoluționlă)
CSV = Comma Separated Values (Valori separate prin virgulă)
CPU = Central Processing Unit (Unitate centrală de procesare)
CUDA = Compute Unified Device Architecture (Calcularea arhitecturii unificate a dispozitivelor)
GPS = Global Positioning System (Sistem de poziționare globală)
GPU = Graphics Processing Unit (Unitate de procesare grafică)
LIDAR = Light Detection and Ranging (Detectarea și măsurarea luminii)
ML = Machine Learning (Învățare automată)
NON-ML = Non Machine Learning (Învatare ne automată)
RAM = Random Access Memory (Memorie cu acces aleator)
RNN = Recursive Neural Network (Rețea neuronală recursivă)
VO = Visual Odometry (Odometrie vizuală)

Introducere

0.1 Motivație

În ultimii ani, micro-vehiculele aeriene au început să atragă atenția cercetătorilor, inginerilor, dar și a organizațiilor militare, prin multitudinea de sarcini pe care le pot realiza în multiple domenii. Câteva din aceste sarcini sunt: servicii de livrare, realizarea hărților și a traseelor optime pentru diferite sarcini din industria de transport și realizarea unei supravegheri mai ușoare și rapide a diferitelor puncte strategice sau de interes. În toate aceste sarcini pe care micro-vehiculele aeriene le pot realiza, estimarea corectă a micro parametrilor de translație și rotație, este esențială pentru a putea controla cu o precizie ridicată și cu maximă siguranță prin evitarea obstacolelor. Anterior, au fost create modele care se bazează pe algoritmi matematici de o complexitate ridicată și o serie de tehnici pentru a controla multitudinea senzorilor. Aceste metode au dovedit că pot obține rezultate bune, dar cu anumite limitări. Una dintre aceste limitări este faptul că sunt predispuse la erori și necesită o putere de calcul sporită. Aceste erori provin din diferiți factori precum, zgomotul din circuitele senzorilor, diferite situații din mediile înconjurătoare sau condițiile dinamice de vreme care pot apărea în timpul de rulare al micro-vehiculului aerian.

În această lucrare urmărim realizarea unei variante alternative pentru estimarea micro parametrilor menționați. Această variantă alternativă propune estimarea acestor parametrii folosind tehnici de învățare adâncă precum, rețelele neuronale convoluționale. Această abordare se mai numește și odometrie vizuală sau visual odometry.

Rețele neuronale convoluționale au dovedit capabilități impresionante în ceea ce privește sarcinile care implică imagini. Au dovedit că sunt competitive sau chiar superioare tehnicilor de învățare standard, care nu implică învățare automată. Chiar dacă acestea s-au dovedit foarte utile, au rămas încă sarcini în care tehnicile standard rămân superioare sau încă nu au fost implementate pentru acele sarcini. Abilitatea de a extrage caracteristici utile din imagini, elimină nevoia de a avea un inginer care să se ocupe de această sarcină și se mulează perfect pe scenariul prezentat în această lucrare, mai exact estimarea mișcării de translație și rotație a vehiculului. Aceste estimări se pot realiza ușor și rapid, doar prin folosirea imaginilor unei camere montate la bord, ceea ce reduce semnificativ nevoia unei echipări cu o multitudine de senzori și echipament complex.

Un alt aspect în care acest tip de rețea neuronală reușește să fie mai performantă decât variantele standard propuse, este abilitatea de a se adapta la condițiile exterioare, care în unele cazuri, se pot schimba extrem de rapid și imprevizibil. Aceste rețele reușesc acest aspect prin antrenarea anterioara a rețelei cu o multitudine de condiții și situații, de unde rețeaua învață să extragă doar caracteristicile utile și astfel se reduc erorile cauzate de factorii externi.

Pentru a implementa o astfel de rețea și pentru a realiza partea de antrenare, am folosit 2 baze de date ce ne oferă imagini extrase din secvențe video, iar fiecare pereche de imagini are alocate măsurători precise ale mișcări de translație și rotație, obținute cu ajutorul unor platforme specializate și a tehnicilor care nu implică învățare automată. Aceste perechi de imagini și adnotări, stau la baza antrenării și testării modelului realizat. În această lucrare o să fie explicate aspecte precum, modalitatea de antrenare si testare, setarea parametrilor potriviți ai rețelei, concluzii legate de performantele obținute și viitoare posibilități de a sporii performanțele.

0.2 Obiectivele stabilite

Ținând cont de importanța dezvoltării unei astfel de rețele neuronale convoluționale, care să se poată folosii în sarcina de odometrie vizuală, obiectivele principale sunt următoarele:

- Încărcarea datelor provenite de la baze de date și prelucrarea acestora, pentru a putea deveni compatibile cu rețeaua neuronală;
- Proiectarea și implementarea unei rețele neuronale convoluțională care să realizeze estimarea mișcării de rotație și translație a unui micro-vehicul aerian;
- Testarea și evaluare celor mai bune tehnici de antrenare, relative la bazele de date disponibile și evidențierea punctelor puternice și slabe ale acestei alternative de odometrie vizuală;

0.3 Structura lucrării

Capitolul 1 prezintă partea teoretică a acestei lucrări. Sunt prezentate informații esențiale în ceea ce privește arhitectura unei rețele neuronale simple sau a unei rețele neuronale convoluționale. De asemenea acest capitol prezintă informații legate de modalități de realizare a tehniciilor de odometrie vizuală convențională, dar și prin intermediul CNN.

Capitolul 2 prezintă tehnologiile utilizate în realizarea practică a modelului, precum limbajul de programare Python, cadrul de lucru Pytorch și echipamentul folosit în antrenarea acestei rețele. În finalul capitolului sunt menționate și explicate modalitățile de prelucrarea a datelor realizate de echipele de dezvoltare pentru cele 2 baze de date care au stat la baza antrenării modelului.

Capitolul 3 este alcătuit din 2 subcapitole, care prezintă pregătirea imaginilor pentru a putea fi încărcate și folosite de către model, prelucrarea datelor de la sol, iar în final este prezentată arhitectura finală a modelului.

Capitolul 4 prezintă rezultatele experimentale obținute după antrenarea modelului cu 3 combinații diferite de date, pentru a putea vizualiza ce variantă se potrivește mai bine pentru sarcina propusă.

Ultimul capitol, intitulat Concluzii, prezintă concluziile finale legate de alegerea celui mai potrivit model și aspecte următoare pentru a dezvolta și a sporii în continuare performanțele modelul.

Capitolul 1
Aspecte teoretice

1.1 Învățare adâncă(Deep Learning)

Învățarea adâncă este un subdomeniu al inteligenței artificiale care se concentrează pe crearea unui model de rețea neuronală capabilă să producă decizii precise bazate pe informațiile prezentate. Această abordare se pliază foarte bine în situațiile în care informația este una foarte complexă și există o bază de date foarte mare.

Învățarea adâncă a apărut prin cercetarea inteligenței artificiale și a învățării automate. Această tehnică se bazează pe atribuirea unor funcții cu rolul de cartografiere deterministă. Conceptul de funcție este foarte importantă pentru învățarea adâncă și se definește ca fiind o cartografiere între datele de intrare și cele de ieșire. Funcțiile se pot reprezenta ca simple operații aritmetice, funcții liniare, neliniare sau reprezentări mult mai complexe ale datelor. Scopul acestui tip de învățare este chiar determinarea acestor funcții provenite din informațiile oferite sistemului.

Rețelele neuronale sunt o modalitate de a reprezenta aceste funcții. Cum am menționat anterior, învățarea adâncă se bazează pe crearea unui model de rețea neuronală, mai exact, structura extrasă din seturile de date sunt funcții care se pot reprezenta ca o rețea neuronală. Structura unei astfel de rețea se poate împarți în locații de memorie, unde datele de intrare sunt stocate și neuronii care implementează fiecare câte o funcție. Datele de intrare sunt trimise primului strat de neuroni, iar datele de la ieșirea acelui strat este trimisă ca date de intrare în următorul strat până la final unde putem afla datele finale de ieșire. Numărul de neuroni și dimensiunea rețelei, depind de complexitatea obiectivului pe care îl avem de realizat și resursele disponibile.

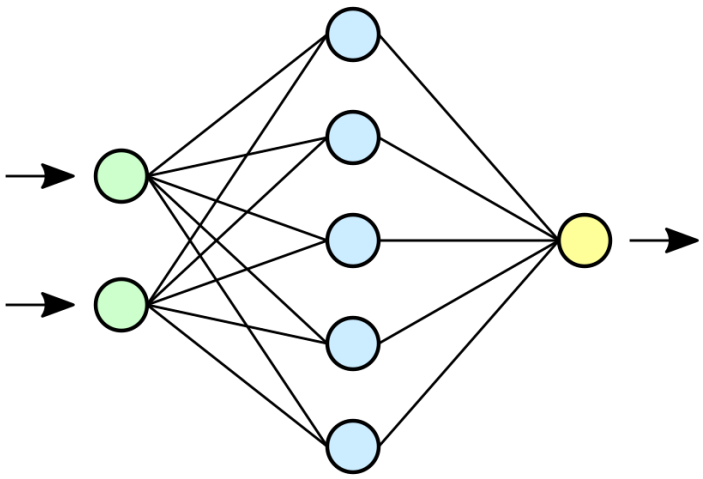

Figura 1.1: Rețea neuronală ; [1]

Pentru a învață o funcție, rețelele neuronale folosesc o strategie de tipul împarte și cucerește. O funcție simplă este învățată de câte un neuron din rețea, iar funcția complexă definită de

rețea este creată prin combinația funcțiilor simple învățate individual de neuronii rețelei.

Utilitatea extragerii unei funcții dintr-un anumit set de date este dată de faptul că, după ce funcția a fost definită, putem aplica aceeași funcție pe un set nou de date, pentru a extrage datele utile rezolvării problemelor din setul nou de date. Trebuie menționat faptul că noul set de date trebuie să se afle din același domeniul de interes pentru a putea extrage date relevante.[5]

În domeniul sarcinii de procesare a imaginilor există 2 paradigme, învățarea supervizată și învățarea nesupervizată.

- Învățarea supervizată reprezintă învățarea sistemului folosind date de intrare etichetate. Pentru fiecare antrenament care este făcut asupra sistemului, o să avem un set de date, care de obicei sunt sub formă de vector și o valoare de ieșire asociată. Obiectivul final este de a reduce eroarea dintre predicțiile modelului și valorile reale ale acestora prin antrenament succesiv;
- Învățarea nesupervizată diferă prin faptul că seturile de antrenament nu includ și etichete pentru date, iar pentru a putea determina rata de succes, trebuie analizată funcția cost asociată pentru a vedea daca aceasta scade sau se mărește. Este de menționat faptul că pentru majoritatea sarcinilor bazate pe recunoașterea unor structuri provenite din imagini, se folosește învățarea supervizată; [6]

1.2 Rețele neuronale convoluționale (CNN)

1.2.1 Generalitați

Rețelele neuronale convoluționale sunt asemănătoare cu rețele neuronale artificiale tradiționale, prin faptul că sunt formate din neuroni care se optimizează automat prin învățare. Datele de intrare o să fie trimise primului strat de neuroni și se vor realiza diferite funcții liniare sau neliniare. Pornind de la un set de date cu imagini care sunt transformate în vectori, se ajunge la clasificarea lor sau la oricare din sarcinile sistemului, iar sistemul o să aibă o singură funcție de scor, care o să fie memorată în scopul reproduceri fiecărei funcții îndeplinită de neuroni. Această funcție de scor are la bază greutățile rețelei. Ultimul strat de neuroni o să conțină și funcția de pierdere, definită la începutul modelului pentru a ajuta la reglarea greutăților rețelei.

Rețele neuronale convoluționale și rețele neuronale artificiale tradiționale se aseamănă în foarte multe aspecte, dar diferența majoră între cele 2 este faptul că rețelele convoluționale sunt folosite cel mai mult în sarcini de tipul căutării unui model din imaginile atribuite. Aceste rețele sunt concepute să aibă diferite funcții specifice pentru prelucrarea imaginilor, ceea ce le face să aibă o performanță sporită în sarcinile bazate pe secvențe de imagini. Având funcții și metode specifice de prelucrare a rețelelor convoluționale, avem și mai puțini parametrii de setat la începutul modelului pentru ajunge la un model potrivit sarcinii alese și performant.

Una din diferențele majore ale rețelelor neuronale convoluționale este structura straturilor de neuroni. Fiecare strat conține neuroni aranjați într-o structura cu 3 dimensiuni corespunzătoare înălțimii, lățimii și adâncimii datelor de intrare. Adâncimea reprezentată de a treia dimensiune a stratului nu se refera la numărul de straturi pe care le are rețeaua. Acest parametru reprezintă a treia dimensiune a volumului de activare, ceea ce avantajează extracția caracteristicilor utile pentru sarcinile care includ imagini. [6]

1.2.2 Arhitectura rețelelor neuronale convoluționale

Rețelele neuronale convoluționale sunt alcătuite din 3 tipuri diferite de straturi,cum se poate vedea și pe figura 1.2

- straturi convoluționale(Connvolution Layer);
- straturi de grupare(Max Pooling);
- straturi complet conectate(Fully Connected);

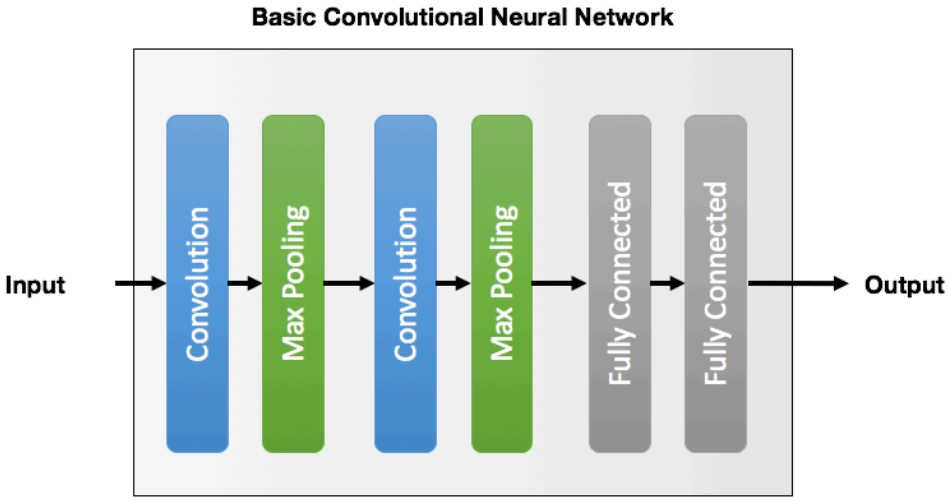

Figura 1.2: Rețea neuronala convoluționala de bază ; [2]

Straturile convoluționare au un rol important în funcționarea rețelelor convoluționare. Acest strat se concentrează pe așa numitele nuclee învățabile. Aceste nuclee sunt împrăștiate de-a lungul adâncimii datelor de intrare. În momentul în care datele de intrare ajung la straturile convoluționare, se realizează o filtrare pe dimensiunea spațiala a intrării, ceea ce creează o hartă de activare cu doar 2 dimensiuni. În timpul în care datele sunt parcurse și convertite în hârți de activare, nucleele pot învață anumite caracteristici în funcție de poziția lor spațiala și sunt cunoscute sub forma de caracteristici de activare. Volumul de ieșire al datelor din stratul convoluțional se formează prin suprapunerea harților de activare, fiecare hartă având corespondent câte un nucleu, iar toată adâncimea datelor de intrare este acoperită.

Straturile convoluționale pot reduce și ajută la complexitatea modelelor folosind câțiva hiperparametrii specifici, care optimizează datele de ieșire din strat. Avem 3 hiperparametrii pe care îi putem modifica:

- Adâncimea(Depth): Adâncimea volumului de ieșire al stratului convolutional este ușor modificabilă, prin ajustarea numărului de neuroni din strat pentru a se putea potrivi pe o regiune specifică din datele de intrare. Prin micșorarea acestui hiperparametru, numărul total de neuroni se poate reduce drastic, dar prin această metoăa pierdem și din capabilitățile de recunoaștere a modelelor.
- Pasul(Stride): Acest hiperparametru este folosit pentru ajustarea câmpurilor din jurul dimensiunilor spațiale de interes care captează informații. Dacă alegem un pas mic, o sa avem o suprapunere mare, rezultând în foarte multe caracteristici de activare. În sens opus, un pas mare reduce suprapunerea, dar o să reducă și dimensiunile spațiului de interes.
- Zero-Umplere(Zero-Padding): Acestă tehnică se rezumă la adăugarea de valori de 0 la marginea datelor de intrare pentru a putea modifica ușor dimensiunile spațiului de interes. Este o metodă directă, dar foarte eficientă pentru a controla această dimensiune.

Straturile de grupare au ca obiectiv principal, diminuarea progresivă a dimensiunii datelor de intrare, pentru a reduce complexitatea de calcul și numarul de parametrii al modelului. Straturile de grupare folosesc hărți de activare individuale provenite de la datele de intrare și se folosesc de funcția de maxim pentru a reduce dimensiunile.

Straturile complet conectate sunt alcatuite din neuroni direct conectați la straturile anterioare și posterioare. În interiorul acestui tip de strat, neuronii nu sunt conectați ,iar această structură de neuroni este foarte asemanatoare cu structura convențională de rețea neuronală artificială. [6]

1.3 Odometrie Vizuală (Visual Odometry)

1.3.1 Fundamentele odometriei vizuale

Realizarea unei tehnici de odometrie precisă a unui obiect mobil, este una dintre cele mai importante sarcini din industria de aplicații mobile pentru roboți. Pentru a putea avea control deplin și perfect autonom, roboții trebuie să poată accesa în permanență poziția exactă. Cercetătorii și inginerii au realizat o mulțime de sisteme, tehnici și senzori pentru a se adresa direct acestor provocări. Printre acestea putem enumera:

- Odometrie bazată pe miscarea roților;
- GPS;
- Odometrie folosind ultrasunete ;
- Odometrie vizuală;

Odometriea bazată pe mișcarea roții este o metodă de estimare a poziției foarte des folosită și ușor de realizat. Această tehnică presupune numărarea rotațiilor pe care roata în contact cu terenul le face pentru a realiza estimarea dorită. Având numărul de rotații, se poate face foarte precis conversia la o traiectorie liniară față de teren. Chiar dacă este o metoda des întâlnită și ușor de realizat, aceasta poate avea probleme din cauza momentelor în care roata alunecă pe teren. Acest aspect nu are efect imediat, dar în timp acuratețea scade din cauza deplasării neglijată de alunecarea roții, care duce la erori cumulative și nevoia de resetare a ansamblului.

Tehnologia de localizare GPS are o largă gamă de aplicații. Folosind acest tip de localizare, putem găsi aplicații al acestei tehnologii în agricultura, transportul public, construcții, dar ceea mai folosită aplicatie pentru GPS este navigația. Cel mai mare atribut al tehnologiei GPS este abilitatea de a oferi poziția și navigația precisă, având zero costuri pentru utilizatorii care dispun de un receptor GPS. Acest tip de localizare se folosește de o constelație de 24 de sateliți operaționali care arbitrează în jurul planetei și transmit semnale de radio frecventă codificate. Este capabilă să ofere poziția absolută, cu un nivel de eroare cunoscut. Această tehnologie nu suferă din cauza acumulării de erori de-a lungul timpului, ceea ce îl face foarte stabil pe perioade mari de timp. Dezavantajul major al acestei tehnologii este că nu oferă posibilitatea de navigare în interior cladirilor sau în zone care nu sunt vizibile de către sateliți.

Senzorii ultrasonici sunt folosiți pentru detecția și măsurarea distanței dintre senzor și obiect. Măsurătoarea se realizează utilizând energia acustică. Ansamblul este format din transmițătorul care emite un puls ultrasonic și receptorul care captează pulsul după ce a fost reflectat. Senzorii măsoară timpul dintre momentul în care a fost emis pulsul și momentul în care a fost recepționat. Având această abordare, limitările sunt destul de evidente. Timpul calculat depinde de materialul și orientarea obiectului care reflectă semnalul emis și devin foarte sensibili la zgomotul provocat în mediul înconjurător de alte dispozitive care folosesc același tip de senzori ultrasonici.

Odometria vizuală (VO) este procesul de estimare al poziției și orientării unei camere video montate pe vehicule mobile, oameni sau roboți. Această tehnică se folosește de un flux de date, mai exact, imagini consecutive de la una sau mai multe camerele atașate. Configurația camerelor poate să fie una variată, folosind camere stereo, mono sau orice altă configurație, în funcție de nevoile sarcinii. Dacă comparăm cu metodele tradiționale de odometrie, cum ar fi GPS, odometria cu sonar, odometria vizuală este considerată mult mai bună la nivel de costuri, dar și mai precisă. Eroarea relativă a poziției poate să varieze în funcție de cât de performant este modelul folosit, dar estimativ, aceasta se află între 0.1-2% pentru modele cele mai performante.

Implementarea unui astfel de sistem de odometrie vizuală se poate considera fiabil și prezintă o balanță destul de echilibrată între costurile necesare și complexitatea de implementare. O altă deosebire față de sistemele convenționale de odometrie care sunt deja implementate la nivel global, odometria vizuală nu emite energie detectabilă în mediul înconjurător și nu necesită nici semnale specifice pentru localizare, precum sistemul GPS.

Beneficiile folosirii unor camere video pe diferite structuri mobile, față de senzorii fizici, sunt integrarea relativ simplă a diverșilor algoritmi bazați pe datele vizuale, cu un cost redus. Majoritatea senzorilor convenționali, necesită calibrarea acestora după o perioadă de timp, iar acestă este un beneficiu pentru tehnicile de odometrie vizuală care nu necesită calibrare sau calibrarea se poate realiza automat și de la distanță. [7]

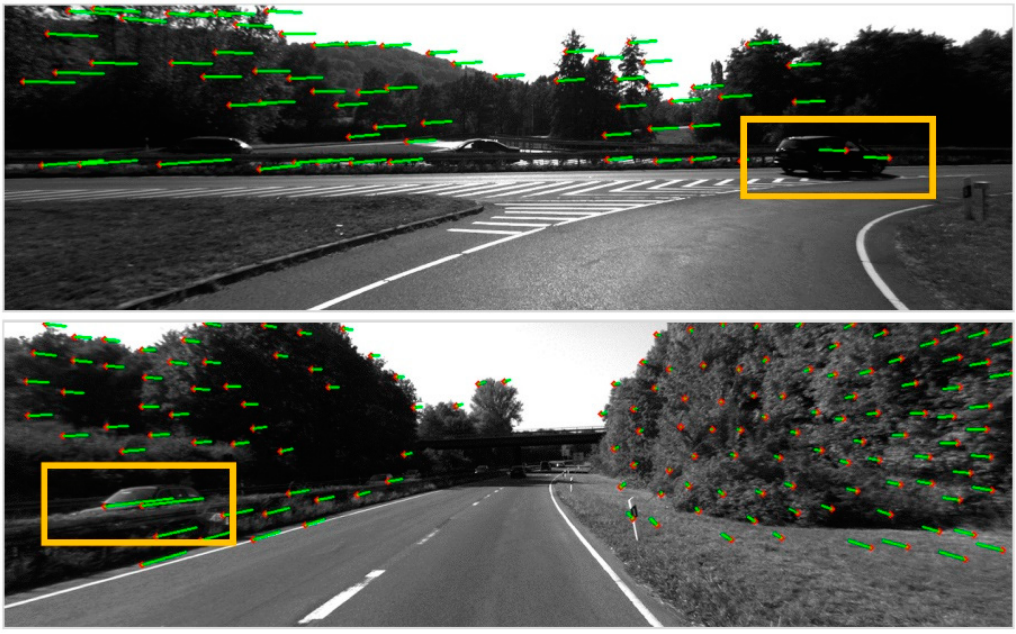

Figura 1.3: Extracție de caracteristici vizuale ; [3]

1.3.2 Odometrie vizuală folosind rețele neuronale convoluționale

Rețelele neuronale convoluționale au dovedit o eficientă impresionantă în sarcinile care aveau ca problemă principală, imagini și modalități în care calculatoarele să realizeze distincția anumitor caracteristici intre două s-au mai multe imagini. Aceste tipuri de rețele au reușit chiar să devină mai performante decât variantele tradiționale care nu se folosesc de ML, în anumite provocări și sarcini. Există totuși și anumite sarcini în care tehnicile non-ML se descurcă extraordinar, au fost testate de-a lungul anilor și larg folosite pentru diferite sarcini în care s-au dovedit mai precise, cu o nevoie computațională mai mică sau în cele mai multe cazuri, amândouă.

Odometria vizuală implică folosirea unor senzori vizuali, cum ar fi o cameră video, pentru a estima modificarea poziției robotului sau a obiectului mobil folosit. În ultimii 12 ani, această metodă a devenit populară și a devenit o arie destul de mare de cercetare din cauza implicarii directe într-o mulțime de probleme precum:

- Realizarea de mașini perfect autonome care să fie accesibile la preț pentru o gama variată de persoane ;
- Realizarea de roboți mobili, terestrii sau aerieni pentru diferite sarcini de automatizare sau explorare ;
- Îmbunătățirea sistemelor care se ocupă cu realitatea augumentată ;

Rețelele neuronale convoluționale au arătat rezultate foarte bune în testele de recunoaștere a obiectelor din imagini și în extragerea anumitor caracteristici din secvențe de imagini. Aceste rezultate se păstrează chiar dacă apar mișcari de rotație, translație sau în condiții diferite de iluminare. Datorită avansului tehnologic și a nevoilor de putere de calcul marită, foarte multă muncă a fost redirecționată în cresterea vitezei prin paralelizarea proceselor și accelerarea grafică. Rezultatul acestei dezvoltari a fost faptul că, rețelele neuronale convoluționale pot aborda problemele legate de odometria vizuală și pot deveni mai performante și eficiente decat prin abordarea unei sarcini identice cu metode alternative tradiționale, care nu pot beneficia de paralelizare și accelerare grafică. [8]

Capitolul 2
Tehnologii utilizate

2.1 Python

Python este un limbaj de programare interpretat, cu o semantică dinamică și caracteristici avantajoase pentru programarea obiect orientată de nivel înalt. Conexiunile dinamice și tastarea dinamică, împreună cu structurile de date de nivel înalt care sunt încorporate în pachetul de bază sau se pot adaugă foarte ușor, fac acest limbaj de programare să fie foarte popular și printre cele mai alese limbaje de programare pentru dezvoltarea rapidă a oricăror aplicații. Datorită acestor factori, este deseori utilizat pentru scripting sau pentru a unii 2 componente deja existente.

Sintaxa acestui limbaj de programare este una foarte simplă, ușor de citit și de înțeles. Acest lucru ajută persoanele care sunt la început de carieră în programare sau reduce foarte mult costurile de întreținere a programelor scrise de persoanele mult mai experimentate. Un alt avantaj al limbajului Python este suportul pentru module și pachete, ceea ce creează și promovează un stil de a scrie programe modulare și un mediu în care este foarte ușor sa refolosești bucăți de cod. De asemenea, Python oferă interpretorul și librăriile standard gratuit, în diferite forme pentru aproximativ toate platformele. [9]

În paragraful precedent am vorbit despre un mediu de programare optim și trebuie menționat faptul că pentru a avea un astfel de mediu, este nevoie să utilizam librarii și diferite cadre de lucru. Python ne oferă acest mediu optim prin librăriile și cadrele sale de lucru, cu scopul de a reduce timpul de dezvoltare software. Aceste librarii conțin cod deja scris și de cele mai multe ori implementat în numeroase funcții simple și ușor de folosit, care sunt folosite de către dezvoltatori pentru a realiza mai ușor și mai rapid proiectele în care sunt implicați.

Pe măsura ce tehnologia avansează, apar din ce în ce mai multe probleme reale în lume, care sunt dificil de rezolvat, dacă nu folosim o inteligență artificială. Prin simplitatea, consistența de care a dat dovadă de-a lungul anilor și a comunității foarte dedicate și numeroasă, Python este considerat mult bun de folosit în cadrul proiectelor de automatizare cu inteligență artificială, față de alte limbaje de programare. [10]

Având în vedere cele menționate mai sus și introducerea despre această lucrare, consider ca acest limbaj de programare s-a potrivit perfect pentru toate sarcinile pe care le-am avut de realizat în cadrul acestui proiect. În continuare o să prezint cadrul de lucru cel mai important în care am lucrat pentru realizarea și finalizarea proiectului.

2.2 Pytorch

Există o multitudine de cadre de lucru specializate pentru a lucra cu proiecte de deep learning. Tendința acestor cadre a fost de obicei, de a realiza ori un cadru foarte versatil și utilizabil, dar lent, ori opusul acestuia, un cadru de lucru rapid, dar nu foarte versatil și ușor de folosit. Acestea fiind spuse, Pytorch, o librărie specializată pentru machine learning, a demonstrat că cele 2 proprietăți menționate pot să fie maximizate în același cadru de lucru. Pytorch ne oferă un stil de programare imperativ, dar pastrează și stilul normal de python

cu care comunitatea este obișnuită. Tratează codul ca pe un model, a simplificat căutarea și rezolvarea erorilor și este perfect compatibil cu o varietate foarte mare de librarii științifice, care sunt folosite la scală largă. În același timp, rămâne foarte eficientă și suportă accelerare grafică utilizând de exemplu unități de procesare grafice (GPU).

Pytorch este un succes, deoarece a reușit să încorporeze într-un design, câteva caracteristici esențiale care balansează viteza și abilitatea de a fi ușor de folosit. În urma acestui design, creatorii acestei librarii au stabilit patru puncte cheie pentru a putea face alegerile potrivite în legătură cu designul și arhitectura librăriei. Aceste puncte cheie sunt urmatoarele:

- Primul punct important pe care dezvoltatorii l-au implementat, este potrivirea perfectă și naturală cu ecosistemul deja existent din python. Acest ecosistem se bazează foarte mult pe simplitate ,ușoara realizare a obiectivelor setate de utilizator și consistentă;
- Al doilea punct este menținerea unei performanțe ridicate, având capabilitatea de a avea viteze mari de compilare și parcurge a diferitelor modele propuse, dar fără să complice utilizarea, păstrând o utilizare foarte accesibilă și facilă pentru utilizator;
- Al treilea punct este menținerea unei interfețe ușor de folosit de către utilizator, încercând să facă sarcini precum: încărcarea de date, scrierea și dezvoltarea modelelor, crearea de buclee de antrenare și testare foarte ușor de înțeles și lucrat cu ele ;
- Al patrulea punct se referă mai precis la realizarea unei implementări interne mai simplă în cadrul librăriei pentru a realiza un ritm accelerat și mai eficient de îmbunătățiri în acest context al inteligenței artificiale și a învățării adânci ;

Pe măsură ce rețelele neuronale s-au dezvoltat, acestea au devenit mult mai complexe, având printre altele, funcții recursive și numeroase bucle pentru a extrage funcțiile dorite pornind chiar de la cele mai simple secvențe de parcurgere directă. Pytorch reușește să realizeze aceste performanțe, prin păstrarea caracterul imperativ al limbajului python și a abordării meta-programării grafică. Conceptele de bază folosite în programare au rămas active și se aplică în fiecare pas al unui proiect de deep learning , cum ar fi:

- Definirea straturilor ;
- Crearea efectivă a modelului ;
- Încarcarea de date ;
- Paralelizarea procesului de antrenare și testare ;

```
1 class CircleModelV0(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.layer_1=nn.Linear(in_features=2,out_features=5)
5         self.layer_2=nn.Linear(in_features=5,out_features=1)
6     def forward(self,x):
7         return self.layer_2(self.layer_1(x))
8 model_0=CircleModelV0().to(device)
9 model_0
```

Figura 2.1: Definirea unui model simplu în pytorch;

La fel, toate uneltele folosite în python se pot aplica în fiecare pas menționat mai sus. Unelte precum afișarea anumitor parametrii, folosirea de librarii pentru a vizualiza rezultatele sau găsirea și rezolvarea de probleme, se pot folosi fără nicio problemă, la fel de ușor ca în orice alt program scris în python. Având o viteză destul de mare de compilare, este foarte ușor să

analizezi rapid cât de bine se descurcă modelul creat și să verifici dacă datele obținute de la model, sunt precise sau au nevoie de modificări. [11]

2.3 Google Colaboratory

Jupyter Notebook este o platformă browser-based, open source, care adună într-un singur program, interpretorul pentru python, librăriile specifice și unelte de vizualizare a datelor. După ce se creează documentul, nu avem un singur câmp de scris cod. Putem avea nenumărate celule care se pot executa separat sau toate la un loc. Aceste celule pot conține anumite scripturi sau linii de cod simple de executat, iar fiecare celulă are propriul câmp pentru a putea vizualiza rezultatele, afișa imagini, tabele sau grafice. Este foarte ușor să organizezi celulele și să creezi linii de cod modulare pe care să le poți refolosi în viitor, exact după filozofia din ecosistemul python.

Google Colaboratory (Colab) este un proiect care are ca obiectiv principal, de a ajuta persoanele interesate de machine learning să învețe sau să realizeze articole de cercetare în acest domeniu. La baza proiectului este Jupyter Notebook și funcționează ca un document care poate fi accesat de oricine are acces și de oriunde avem o conexiune la internet, pentru a ajuta la colaborarea dintre mai multe persoane. Colab oferă un mediu pre configurat cu versiunea Python 3 și cele mai utilizate librarii. TensorFlow, Keras și Pytorch fiind unele din multiplele librării accesibile. Mașina virtuală care este folosită în momentul rulării se deconectează automat după o anumită perioadă de timp și datele obținute sunt pierdute, dacă nu au fost salvate în prealabil, dar notebook-ul este păstrat. Oferă posibilitatea de a încarca sau stoca date direct din Google Drive sau se poate folosii spațiul alocat direct din mașina virtuală. În plus, putem avea acces și la sesiuni care folosesc GPU pentru a avea timpi de rulare mai mici. Pentru a realiza această lucrare am folosit varianta gratuită pe care o oferă Colab-ul. În această variantă am avut acces la un CPU cu 2 nuclee, memorie RAM disponibilă de 12.7 Gb, iar în materie de GPU avem disponibil în versiunea standard Tesla T4 cu 15 Gb de memorie disponibilă.

Dezavantajul major pe care îl are varianta gratuită este că nu avem afișat timpul rămas disponibil pentru a rula, iar după o rulare mai mare, nu poți știi când mașina virtuală o să se deconecteze. În urma experimentelor realizate de mine, timpul de rulare pe o masină virtuală care oferă GPU este aproximativ 4 sau 5 ore pe zi, dar poate să varieze foarte mult în momentul în care se folosește zilnic această resursă. Cu cât se folosesc resursele mai mult, zilnic, timpul de rulare pentru fiecare zi scade. Acest dezavantaj poate să fie ocolit prin folosirea a multiple conturi care au în comun notebook-ul. [12]

2.4 CUDA

Arhitectura creată de către Nvidia și folosită exclusiv în procesoarele grafice produse de ei, este cunoscută cu numele de CUDA (compute Unified Device Arhitecture). Acestă tehnologie oferă o nouă metodă de folosire a puterii de procesare provenită de la procesoarele grafice, prin crearea unei extensii al cunoscutul limbaj de programare C/C++, numită Cuda C/C++ . Folosind această extensie, dezvoltatorii software pot să se folosească de puterea mare de procesare oferită de către procesoarele grafice. Această putere sporită de calcul se datorează capabilitățile de procesare paralelă prin alocarea a multor fire de execuție să ruleze concomitent pe procesorul grafic.

Arhitectura oferă 128 de nuclee care pot să lucreze împreună pentru a realiza anumite sarcini. Din această cauză, în momentul în care se utilizează o aplicație care necesită mai multe fire de execuție, nu este nevoie să se transfere puterea de procesare către GPU, deoarece

nucleele din interiorul CUDA pot comunica și realiza schimbul de date. Având aceste aspecte în vedere, arhitectura CUDA este foarte utilizată pentru algoritmii care necesită putere mare de procesare paralelă, de aceea dezvoltatorii software pentru algoritmii de ML optează pentru o astfel de arhitectură.

Pentru a reuși să obținem performanțe mărite pentru algoritmii de ML, avem nevoie de un număr foarte mare de fire de execuție, ceea ce în foarte multe cazuri garantează aceste performanțe mărite, dar CUDA poate să nu fie întotdeauna cea mai bună variantă. Cel mai bun exemplu pentru acest caz este orice algoritm serial. În cazul majorității acestor tipuri de algoritmi, nu se poate face împărțirea în mai multe blocuri, care să aibă un număr de fire de execuție, de ordinul miilor. Acești algoritmi se pot transforma în algoritmi de tip paralel, dar această variantă nu este întotdeauna posibilă.

Daca algoritmii pe care îi folosim pot să fie împărțiți în blocuri cu un număr minim de ordinul miilor , arhitectura CUDA este o variantă foarte bună pentru a maximiza performanțele, ceea ce rezultă în cazul algoritmilor de ML, în timpi mai mici de așteptare în momentul antrenării modelelor și rezultate mai bune în privință predicțiilor făcute.

Un alt beneficiu al folosirii acestei tehnologii create de NVIDIA este că nu avem nevoie să scriem tot codul în extensia CUDA C/C++. În momentul dezvoltării unei aplicații complexe, a unui model sau algoritm de ML, majoritatea codului poate să fie scris în orice limbaj de programare dorim. În momentul în care avem nevoie de o putere mare de calcul pentru a realiza diferite operații matematice complexe, putem apela funcțiile CUDA. Această abordare permite folosirea ușoară și mai eficientă a puterii mari de calcul a procesoarelor grafice, fără a fi nevoie de a scrie tot programul în CUDA C/C++. [13]

2.5 The KITTI Vision

Pentru realizarea unui model de început din această lucrare, am folosit baza de date numită KITTI Vision. Echipa din spatele acestei baze de date foloseste o platformă mobilă autonomă, pe care au numit-o Annieway, pentru a crea repere noi în domeniul viziunii computerizată din lumea reală. Aceștia au realizat baze de date pentru diferite obiective, care constau în capturarea de imagini stereo, flux optic, odometrie vizuală sau detecție de obiecte 3D. Pentru realizarea acestor sarcini, echipa din spatele proiectului a folosit autovehiculul menționat anterior, echipat cu 2 camere de rezoluție înaltă care pot filma color sau alb-negru. Parametrii de la sol legați de traiectorie, rotație și translație au fost obținute printr-un sistem de localizare GPS și un scaner laser.

Imaginile surprinse de cele 2 camere montate au adunat secvențe din diferite medii, cum ar fi zone rurale, zone de autostradă sau chiar un oraș de dimensiuni medii. S-a încercat ca în fiecare imagine să existe repere din mediul respectiv, cât mai variate și dificil de prelucrat, pentru sistemele de învățare automată. Aceste repere sunt reprezentate de multiple mașini și pietoni care apar. Datele obținute au fost apoi împărțite ca date de baza, care să cuprindă repere specifice și evaluări metrice pentru fiecare obiectiv menționat mai sus, dar și un site web de evaluare a datelor obținute de persoanele care doresc să folosească aceste baze de date pentru a putea valida eficiența modelelor sau a metodelor alese.

Obiectivul principal al echipei a fost de a aduce mai multe repere cu diferite dificultăți și probleme din lumea reală pentru a ajuta la dezvoltarea viitoarelor tehnici și modele de ML.

În această lucrare am folosit această bază de date, strâns corelată cu odometria vizuală. Reperele din această secțiune constau în 22 de secvențe stereo, color, salvate într-un format cu pierderi minime. Primele 11 secvențe oferă și datele reale de la sol sub forma a 12 parametrii care după anumite prelucrări, pot constitui o matrice de rotație și una de translație. Fiecare rând pe care îl găsim în fișierele atașate bazei de date sunt relative la momentul 0 al secvenței.[14]

2.6 The NTU VIRAL

Cel mai important factor motivator, pe care l-au avut echipele care se ocupă cu crearea de baze de date publice, este de a facilita progresul tehnologiei în diferite domenii pentru a reduce pe viitor nevoia de investiții în hardware. Dacă facem o comparație directă între bazele de date disponibile pentru diferite vehicule autonome de sol și cele create pentru sistemele aeriene autonome, putem observa că pentru cele aeriene nu avem o varietate prea mare de baze de date complexe și pregătite pentru utilizarea în diferite proiecte de dezvoltare a tehnologiei în acest domeniu.

Echipa care a creat baza de date NTU VIRAL, a dorit să acopere această lipsă de date și a reușit să colecteze datele necesare unei baze de date complexe, folosind un set cuprinzător de senzori echipați pe o platformă aeriană. Echipamentul folosit este următorul:

- Doi senzori de tipul 3D LIDAR care folosesc raze laser pentru a măsura distanțe și a crea o reprezentare în 3 dimensiuni a mediului înconjurător ;
- Două camere cu sincronizare globală a obiectivului pentru a putea realiza o reprezentare stereo corectă ;
- Multipli senzori inerțiali pentru a putea măsura cu o eroare minimă, viteza și accelerația platformei ;
- Multipli senzori de bandă foarte largă pentru a putea măsură cu precizie distanțele dintre platformă și reperele stabilite ;

Tot ce am enumerat mai sus se pot găsi pe toate platformele de vehicule de sol autonome, dar pentru o platformă aeriană autonomă, aduce noi probleme și provocări. Prin aceste metode, echipa a reușit să colecteze date precise din diferite locații, unele chiar și în interiorul anumitor clădiri, realizând o multitudine de baze de date care conțin imagini stereo, date legate de calibrare și date precise de la sol, sub forma a 12 parametrii, asemănător cu cei din baza de date menționată în subcapitolul anterior. [15] [16]

Capitolul 3
Implemenarea arhitecturii

3.1 Prelucrarea datelor initiale

Pentru a începe această lucrare și a reuși să creez o rețea complexă de învățare adâncă, primul pas este de a prelucra datele oferite de cele 2 baze de date menționate în capitolul 2.5 si 2.6. Aceste baze de date ne oferă tot ce avem nevoie în materie de date de bază, pentru dezvoltarea corectă și eficientă a rețelei neuronale.

Prima bază de date este realizată pe baza unei platforme mobile terestre, fiind mai ușor să dezvoltăm un model în aceste circumstanțe și a fost folosită pentru antrenarea și testarea inițiala a soluțiilor găsite. A doua bază de date este realizată pe baza unei platforme aeriene și am folosit-o în antrenarea modelului pre antrenat cu prima bază de date, dar s-a încercat și antrenarea modelului doar cu această bază de date.

Prelucrarea datelor a constat în trei etape diferite de prelucrare pe care le voi detalia în următoarele subcapitole.

3.1.1 Pregatirea datelor de la sol

Ambele baze de date oferă același stil pentru datele de la sol înregistrare pentru cele 2 platforme. Acestea vin sub forma a 12 parametrii, pentru fiecare moment de timp t, care semnifică parametrii de rotație și translație. În documentul de tip text oferit de către dezvoltatorii bazelor de date, parametrii vin sub forma următoare: r00, r01, r02, t0, r10, r11, r12, t1, r20, r21, r22,t2.

Folosind acești 12 parametrii putem crea o matrice de forma 4x4, numită matrice de transformare care arată astfel:

$$\begin{pmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

Pentru o ușoară înțelegere și vizualizare, vom reprezenta fiecare matrice separat și o vom analiza.

$$\begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix} \tag{3.2}$$

$$\begin{pmatrix} t_0 \\ t_1 \\ t_2 \end{pmatrix} \tag{3.3}$$

Formula 3.2 reprezintă matricea de rotație pentru orientarea globală, iar formula 3.3 reprezintă mișcarea de translație pe toate cele 3 axe de deplasare. Ambele formule sunt relative la câte un moment de timp t specific.

Datele inițiale pe care le avem la dispoziție au avut nevoie de prelucrare, deoarece fiecare linie cu cei 12 parametrii este relativă la momentul de timp inițial al fiecarei secvențe. Pentru o acuratețe sporită și în încercarea de a estima acești parametrii cât mai generalizat se poate, a fost nevoie să aplicăm următoarea formulă 3.4 pentru a transforma fiecare linie, într-o linie relativă doar la momentul anterior. Vom nota matricea de transformare cu indicativul $T_{X \to Y}$. Acest indicativ reprezintă matricea de transformare de la poziția de timp y, față de momentul de timp x. [17]

$$T_{(N-1)\to(N)} = T_{(0)\to(N)} \times T^{-1}_{(0)\to(N-1)} \tag{3.4}$$

Am folosit scriptul din Anexa 2 pentru a realiza citirea dintr-un document de tip CSV a datelor necesare și rescrierea într-un alt document de tip CSV, pentru a pastra formatul folosit în toate scripturile create în cadrul lucrarii.

3.1.2 Pregatirea imaginilor pentru antrenare și testare

Pregătirea imaginilor, din secvențele video, este una din părțile esențiale din crearea unei rețele neuronale. Pentru a avea rezultate bune și precise, trebuie să avem pregătite pe lângă setul de date, o porțiune de cod care se ocupă cu prelucrarea imaginilor. În lucrarea aceasta, obiectivul principal este să determinam diferențele de caracteristici dintre 2 poze consecutive. Imaginile au fost extrase din materialele video de către echipele care s-au ocupat cu bazele de date, dar această prelucrare nu este suficientă.

Prelucrările realizate în cadrul acestei lucrări sunt următoarele:

- Crearea unui tip de date folosind o structură de tip obiect care să unească 2 imagini, folosind canalul specific culorilor ;
- Crearea unei normalizări generale pentru fiecare imagine, pentru a avea date cu aceeași dimensiune și augmentare ;
- Transformarea datelor provenite de la imagini în tipul de dată specific pytorch sau orice alt cadru de lucru, adică tensor;

Pentru a putea lucra cu 2 imagini secvențiale și a găsi anumite caracteristici între ele, avem nevoie să le unim într-o anumită structură. Capitolul 3.1.3 o să aducă mai multe explicații legate de modalitatea de unire a imaginilor cu etichetele aferente. Ideea de bază este că imaginile color sunt create prin suprapunerea a 3 canale în formatul RGB. Pentru unirea lor, am folosit acest canal pentru a suprapune cele 2 imagini, ceea ce a rezultat în crearea unui tip de imagine cu 6 canale de culoare.

Înainte ca aceste 2 imagini să se suprapună pe canalul de culoare, avem nevoie să realizăm câteva transformări ale imaginilor. Pentru acestă lucrare, imaginile au fost decupate în formatul 376x376 și s-a aplicat o operație de redimensionare a imaginii pentru a obține în final dimensiunea de 128x128, cu scopul de a păstra cât mai multe caracteristici din imagini. În încercarea de a nu depinde foarte mult de condițiile de mediu, s-a aplicat o augmentare generală în care se setează luminozitatea, contrastul, saturația culorilor și nuanța culorilor la o valoare medie de 0.4 . Pe lângă aceste transformări, aplicăm și o normalizare a datelor de intrare prin scăderea mediei setate și împărțirea la abaterea medie standard.

Imaginile din cadrul bazei de date NTU VIRAL, nu au fost capturare color, iar pentru că arhitectura rețelei neuronale a fost proiectată inițial pentru a accepta imagini color, a fost nevoie de crearea unei transformări care să copieze canalul de culoare existent, în așa fel încât pe toate cele 3 canale de culori să fie aceeași informație, realizând o imagine care rămâne alb-negru, dar se poate folosi cu o arhitectură gândită pentru imagini color.

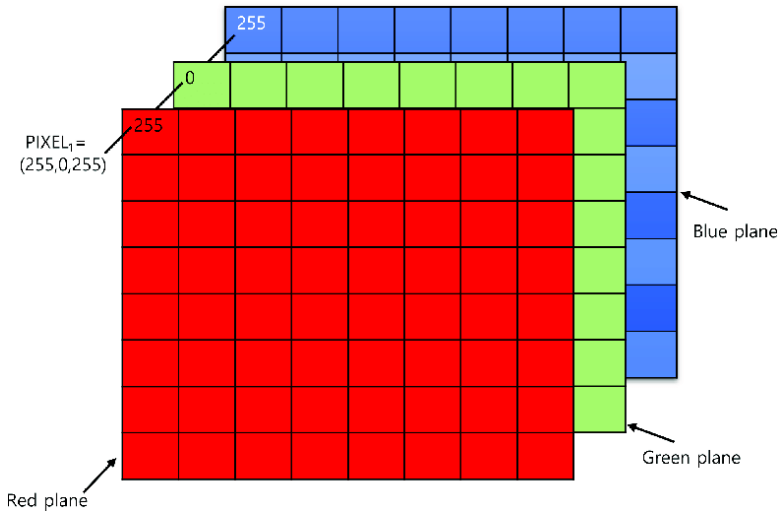

Figura 3.1: Imagine RGB în care sunt evidențiate cele 3 straturi; [4]

```
class GreyToRGB(object):
    def __call__(self, tensor):
        tensor = torch.cat([tensor, tensor, tensor], dim=0)
        return tensor
```

Transformarea în tensori reprezintă trecerea datelor într-un format pe care rețelele neuronale să le poată prelucra. Tensorii reprezintă matrici multi-dimensionale care reprezintă datele într-o anumită formă accesibilă cadrelor de lucru specializate pentru ML. Tensorul este tipul de dată fundamental pentru ML și poate reprezenta date de intrare, date de ieșire sau parametrii din modelele de învățare.

3.1.3 Unirea celor două tipuri de date pentru realizarea unei structuri de tip obiect

În capitolele anterioare am discutat separat despre prelucrarea imaginilor și a etichetelor, dar pentru a realiza rețeaua neuronală propusă, avem nevoie să combinam cele 2 tipuri de date într-o singură structură de tip obiect. În acest mod, am creat clasa CustomDataset prezentă în Anexa 1. Clasa primește ca parametrii, locația directoarelor unde se află imaginile, un document de tip Excel în care sunt precizate numele imaginilor care vor alcătui o pereche, cei 12 parametrii folosiți ca etichete și modalitatea de transformare a imaginilor.

Documentul de tip Excel oferă cele mai multe informații și este realizat individual pentru fiecare bază de date și fiecare mediu.Primele 2 coloane din documentul Excel oferă informații despre pereche de poze, mai exact numele imaginilor care vor forma perechea, iar următoarele 12 coloane oferă parametrii matricei de rotație și translație. În interiorul acestei clase, avem implementate mai multe metode pentru a prelucra datele. Metoda principală obține imaginile din fișierul Excel menționat, încarcă fiecare imagine aferentă unei perechi, separat și aplică transformările dorite. Imaginile sunt apoi suprapuse folosindu-se canalul de culoare, iar metoda ne returnează o imagine cu 6 canale de culori și etichetele perechii de imagini.

```

class CustomDataset(Dataset):
    def __init__(self, directory, csv_file, transform=None):
        self.directory = directory
        self.transform = transform
        self.labels = pd.read_csv(csv_file,header=None)

    def __len__(self):
```

```
        return len(self.labels)

    def __getitem__(self, idx):
        img1_name = os.path.join(self.directory, self.labels.iloc[idx, 0])
        img2_name = os.path.join(self.directory, self.labels.iloc[idx, 1])

        label = [self.labels.iloc[idx, i] for i in range(2, 14)]
        label = torch.tensor(label, dtype=torch.float32)

        img1 = Image.open(img1_name)
        img2 = Image.open(img2_name)

        if self.transform:
            img1 = self.transform(img1)
            img2 = self.transform(img2)
        img = torch.cat([img1, img2], dim=0)

        return img,label
```

3.2 Definirea modelului ales

Una dintre cele mai importante parți din realizarea unei sarcini care implică rețele neuronale adânci de învățare, este de a defini un model suficient de bun pentru a reuși extracția datelor dorite. În cazul acestei lucrări, avem nevoie să extragem anumite caracteristici, dintr-o pereche de imagini, deci avem nevoie de un model care să poată prelua și prelucra astfel de date de intrare pentru a obține rezultatele dorite.

Am ales să folosesc o arhitectură de model pentru o rețea neuronală convoluțională, care a dovedit de-a lungul anilor că este eficientă și poate să ne ajute în extragerea datele esențiale sarcinii dorite în această lucrare. Numele arhitecturii este ResNet-50. ResNet este o prescurtare pentru rețele reziduale, a fost introdusă în anul 2015 și este creată pentru a se folosi în rețele neuronale adânci de învățare. A fost creată pentru a rezolva o problemă pe care arhitecturile tradiționale o întâmpinau constant de-a lungul antrenării, adică dispariția gradienților în timpul antrenării. Dezvoltatorii acestei arhitecturi au introdus un bloc nou, de tip rezidual, care ajută modelul să găsească scurtături, folosite pentru a putea să ocolească anumite straturi și implementarea funcțiilor reziduale printre neuronii rețelei.

Rețeaua pe care am preluat-o se numește ResNet-50, deoarece include 50 de straturi, care pot sa fie:

- Straturi convolutionale;
- Straturi de grupare;
- Straturi complet conectate;
- Straturi cu blocuri reziduale;

Aceste straturi facilitează implementarea sarcinilor care implică extracția caracteristicilor din imagini și în general orice problemă din mediul viziunii computerizată. Modalitatea de funcționare a acestui tip de arhitectură se bazează pe conceptul de învățare reziduală, ceea ce implică faptul că funcțiile reziduale învățate sunt combinate cu datele de intrare, pentru a produce datele de ieșire dorite. Această funcție se realizează folosind blocurile reziduale menționate anterior, care rețin scurtături între conexiunile rețelei. În interiorul acestei arhitecturi se realizează anumite normalizări ale lotului și unități liniare de rectificare pentru a accelera și îmbunătății antrenarea și a evita supra adaptarea datelor. [18]

```
(features): ResNet(
    (conv1): Conv2d(6, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): Bottleneck(
```

```
        (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
```

Modelul realizat în cadrul acestei lucrari este prezentat în Anexa 1 și prezintă modalitatea de preluare a arhitecturii și ajustările realizate pentru a adapta rețeaua la sarcina pe care dorim să o realizăm. Pentru început, se preia modelul pre antrenat, iar pentru a realiza conversia de la o singură imagine acceptată de model, către o pereche de imagini, am modificat primul strat al rețelei pentru a accepta 6 canele de culoare. Modelul preluat conține 50 de straturi care sunt împărțite în 4 straturi principale, care la rândul lor sunt împărțite în straturi numite Bottleneck în care se realizează normalizarea lotului, rectificarea unităților liniare și unde se găsesc straturi convoluționale. Anterior am exemplificat straturile modificate și straturile dintr-un bloc de BottleNeck.

Deoarece arhitectura este gândită pentru a realiza și rezolva sarcini de clasificare, a fost nevoie să adăugam un strat adițional, pentru a realiza porțiunea de regresie necesară predicțiilor celor 12 parametrii. Acest strat are ca parametru de intrare 1000, aceasta fiind numărul de neuroni de ieșire al arhitecturii ResNet. Acest număr de neuroni este apoi crescut treptat până la pragul de 4096 de neuroni pe strat, păstrat constant pentru 2 straturi, iar apoi este redus treptat cu puterile numărului 2 pana la 512 de neuroni. Ultimul strat al regresiei este cel care ne va returna parametrii doriți, ceea ce înseamnă că avem 512 neuroni de intrare și 12 de ieșire. Putem vedea acest bloc creat și în codul urmator.

```
(regression): Sequential(
    (0): Linear(in_features=1000, out_features=2048, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=2048, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=4096, bias=True)
    (7): ReLU(inplace=True)
    (8): Dropout(p=0.5, inplace=False)
    (9): Linear(in_features=4096, out_features=4096, bias=True)
    (10): ReLU(inplace=True)
    (11): Dropout(p=0.5, inplace=False)
    (12): Linear(in_features=4096, out_features=2048, bias=True)
    (13): ReLU(inplace=True)
    (14): Dropout(p=0.5, inplace=False)
    (15): Linear(in_features=2048, out_features=1024, bias=True)
    (16): ReLU(inplace=True)
    (17): Dropout(p=0.5, inplace=False)
    (18): Linear(in_features=1024, out_features=512, bias=True)
    (19): ReLU(inplace=True)
    (20): Dropout(p=0.5, inplace=False)
    (21): Linear(in_features=512, out_features=12, bias=True)
  )
```

Printre straturile menționate, avem implementată și funcția de activare ReLu pentru aplicarea rectificării unităților liniare, pentru a reduce consumul de memorie al unității grafice și resursele computaționale. De asemenea folosim și tehnica de regularizare Dropout care setează procentul de elemente care sunt reduse la 0, pentru a forța modelul să învețe caracteristici utile și solide, nu anumite caracteristici regăsite și legate de zgomotul din perechea de imagini. Procentul setat este de 50%.

Un aspect important de menționat este că în momentul în care modelul se află în starea de

testare, funcția se comportă diferit față de modul de antrenare. Această tehnică este oprită și datele care trec prin straturi sunt scalate la numărul setat pentru a asigura rezultate consistente și comparabile cu cele din timpul testării, unde funcțiile de Dropout funcționează.

Pentru a asigura faptul că modelul reușește să învețe și să implementeze funcția dorită pentru a finaliza sarcina inițială, a trebuit să realizăm operația de activarea a gradienților din ultimele 2 straturi ale arhitecturii ResNet, care în mod implicit sunt dezactivați, pentru ca modelul, în partea de antrenare să poată realiza actualizarea gradienților în pasul de propagare inversă prin model.

În final realizăm trecerea datelor prin arhitectura ResNet, apoi avem nevoie să realizăm reducerea hărții de tensori într-o singură dimensiune pentru a asigura compatibilitatea cu următorul pas din model, cel de regresie liniară. În final realizăm pasul de regresie liniară, care ne va oferii cei 12 parametrii doriți. Putem vedea intreaga arhitectură în figura urmatoare 3.2.

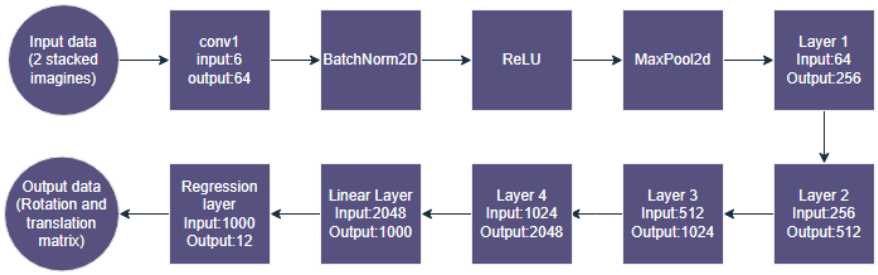

Figura 3.2: Diagramă simplificată a arhitecturii implementate;

Capitolul 4
Rezultate experimentale

4.1 Aspecte inițiale și mențiuni

În cadrul acestei lucrări, s-au folosit 2 baze de date pentru a se încerca maximizarea performaței unei rețele neuronale de tip CNN. În urma implementării modelului și a celorlalți pași explicați în capitolul 3, urmează să se evidențieze dimensiunea datelor folosite în antrenare și testare, dar și modalitatea de antrenare și testare. După ce aceste aspecte sunt prezentate și explicate, în continuare o să existe multiple subcapitole care prezintă performanțele din timpul antrenării folosind cele 3 combinații alese:

- Kitti;
- NTU Viral;
- Kitti și NTU Viral;

În final o să fie prezentate performanțele obținute după un număr fix de epoci ales, pentru a nu avea o perioadă prea lungă de antrenare și a face imposibilă finalizarea în timp a lucrării, dar și punctele slabe ale rețelei și posibilitațile de îmbunatațire.

4.2 Prezentare date folosite

În primul rând trebuie menționate câteva aspecte despre modalitatea și selectarea datelor din cele 2 baze de date. În urma experimentelor realizate și cu ajutorul articolului [17], s-a demonstrat faptul că rețele neuronale de tipul CNN nu se descurcă cu generalizarea datelor din medii necunoscute. După cum am experimentat la începutul realizării acestei lucrări, dar și după articolul menționat anterior, pentru a putea realiza un model care chiar reușește să învețe și să reducă funcția de pierdere, avem nevoie ca împărțirea datelor în 2 subcategorii, mai exact testare si antrenare, să se facă din același mediu. Dacă încercam să antrenăm modelul cu date dintr-un anumit mediu, iar testarea se face cu date dintr-un alt mediu, complet diferit, modelul nu reușește să generalizeze și o să fie blocat în aproximativ același punct, indiferent de câte iterații de epoci avem.

Pentru a avea acces ușor la date, am folosit stocarea internă, pe care ne-o pune la dispoziție mediul de lucru Google Colaboratory și Google Drive. După ce mediile specifice bazei de date au fost alese și documentul Excel menționat în capitolul 3.1.3 a fost realizat, am creat câte o arhivă pentru fiecare bază de date în care am adăugat imaginile dorite și încă o arhivă pentru documentele Excel. Aceste arhive au fost încărcate pe platforma Google Drive, iar de acolo folosind codul următor, au fost descărcate și dezarhivate în mediul de stocare intern al platformei Google Colaboratory. Citirea datelor direct din Google Drive este foarte lentă, ceea ce conduce la timpi de rulare foarte mari. Viteza de descărcare a datelor de pe Google Drive în Google Colab este mare, iar dezarhivarea se face relativ rapid. În medie acest proces durează aproximativ 5 minute, dar apoi datele sunt ușor și rapid accesibile de către rețeaua neuronală.

```
! gdown --id 1tJqVqzTObuBVFNmWLDgyyi7Sy6AS1vjy
! gdown --id 1vcBD4AHI6J6HIX9-28cY42PGzyciKd4L
!unrar x dl_eee.rar
!unrar x drones.rar
```

Având acest aspect în vedere, am procedat în felul următor. Am ales 2 medii din baza de date Kitti, respectiv 3 din baza de date NTU. Cum am explicat și în capitolul 3.1.3 , imaginile sunt încărcate din baza de date , într-un set de date specific pentru fiecare mediu ales, care conține structuri simple alcatuite din 2 imagini secvențiale și cei 12 parametrii de rotație și translație aferenți imaginilor. Din aceste seturi de date au fost preluate la întâmplare 20% date pentru testare și 80% pentru antrenarea modelului. În final cele 80% din fiecare set de date au fost concatenate pentru a crea o structură unică pentru antrenare, respectiv 20% pentru testare, numită dataloader. Această structură este direcționată către modelul creat cu ușurința și ne ajută să avem datele compacte și ușor de accesat.

În cazul primei baze de date, Kitti, am selectat imagini din 2 medii diferite. Primul mediu ne pune la dispoziție 4541, iar cel de al doilea 4661, având în total un mediu de antrenare și testare alcătuit din 9202 imagini. Desigur, după crearea primei structuri de seturi de date, o să avem 4540 de perechi de imagini din primul mediu și 4660 din al doilea, datorită combinării imaginilor în stilul 1-2 2-3 3-4 etc. După cum am menționat anterior, aceste perechi de imagini sunt asociate unuia dintre cele 2 tipuri de date, antrenare sau testare. Din acest punct, ultimul pas este să cream structura de date numită dataloader. Pentru crearea acestei structuri de date, avem nevoie de următorul parametru numit, dimensiunea lotului, pentru a crește viteza de parcurgere și rulare pentru fiecare epocă. Dimensiunea lotului se poate definii în cazul nostru, ca fiind numărul de perechi de poze pentru fiecare dataloader. Pentru a balansa memoria folosită si viteza de realizare a unei epoci, am ales valoarea 64. În final au rezultat 115 dataloaderi, a câte 64 de perechi de poze pentru antrenare și 29 dataloaderi cu câte 64 de perechi pentru testare.

Datorită faptului că aceste perechi de poze sunt selectate și repartizare la întâmplare de un algoritm, avem nevoie să realizăm încă 2 aspecte pentru a asigura o desfășurare corectă. La nivelul de valori luate la întâmplare pentru un calculator, nu putem vorbi despre o alegere perfect la întâmplare. În spatele unor numere alese la întâmplare, stă un algoritm care generează aceste numere, deci putem manipula această alegere de numere în așa fel încât să ne genereze întotdeauna aceleași numere, pentru a asigura repetabilitatea numerelor din lucrare, prin setarea unui seed cu o valoare fixă. Valoarea aleasă este 27. Prin setarea acestui parametru care asigură repetabilitatea acestor numere, putem să fim siguri că aveam întotdeauna aceleași perechi de imagini pentru cele 2 tipuri de date. Al doilea aspect pe care a trebuit să îl realizăm este de a crea câte un dataloader pentru fiecare mediu, pentru a putea exporta valorile generate de model și a le compara cu valorile reale.

În cazul NTU Viral, am procedat în același stil. Avem 3 medii diferite cu care realizăm testarea sau antrenarea, cu câte 3986, 3209 respectiv 1813 de imagini. Seturile de date constau în 3985, 3208 și 1812 perechi de imagini, iar dataloaderul de antrenare are aceeași dimensiune a lotului, adică 64. Avem pentru antrenare 113 dataloaderi de câte 64 de perechi, iar pentru testare 29 de dataloader tot cu 64 de perechi. Am setat același parametrul pentru algoritmul care generează numerele la întâmplare și am realizat încă 3 dataloaderi pentru fiecare mediu ales.

4.3 Prezentare modalitate de antrenare

Antrenarea și evaluarea modelului prezentat în capitolul 3.2 este prezentată în finalul anexei 1. Pentru a realiza o secvență de cod folosită la antrenarea oricărui model de învățare adâncă, mai întâi trebuie aleși 3 parametrii esențiali.

Primul parametru setat este funcția de pierdere, care se referă la o funcție matematică folosită pentru a calcula diferența dintre datele reale și cele produse de către rețeaua neuronală. Valoarea returnată de funcție este direct proporțională cu cât de bine se descurcă rețeaua în a generaliza și a învața noi caracteristici. Există o varietate destul de mare de funcții de pierdere oferite de librăria Pytorch, folosite pentru diferite probleme de ML. Modelul creat de mine trebuie să producă 12 parametrii care compun matricea de rotație și translație, ceea ce rezultă într-o problemă de regresie. Pentru o astfel de problemă, cele mai bune funții de pierdere sunt: [19]

- Eroarea medie absolută(MAE);
- Eroarea medie patratică(MSE);
- Eroarea Huber;

Am ales să folosesc eroarea medie pătratică pentru a creste viteza de convergență a modelului. Folosind această funcție de pierdere, modelul este pedepsit cu un grad mult mai ridicat când rezultatele oferite de model sunt departe de rezultatele dorite, ceea ce ajută dacă modelul este foarte departe inițial de valorile reale sau la un moment dat, evoluția acestuia se îndreaptă într-o direcție greșită față de cea dorită. Formula de bază folosită de această funcție de pierdere este 4.1.

$$\text{Functia de pierdere} = \frac{1}{N}\sum_{i=1}^{N}(y_{i,adevarat} - y_{i,estimat})^2 \tag{4.1}$$

Al doilea parametru pe care trebuie să îl setăm este optimizatorul. Acesta face mici ajustări în funcțiile fiecarui neuron din interiorul modelului, cu obiectivul de a ajunge la o eroare cât mai mică. La fel ca în cadrul funcțiilor de pierdere, în librăria Pytorch avem disponibili o mulțime de optimizatori care se pot potrivi în orice problemă de ML. Algoritmul din spatele optimizatorului ales pentru această lucrare este algoritmul ADAM. Este o combinație între 2 algoritmi populari, RMSProp și AdaGrad. Algoritmul oferă posibilitatea de a seta un pas de învățare, dar acest pas este adaptabil dinamic pentru fiecare parametru bazat pe gradienți. Pasul de învățare ales în urma experimentelor realizate este de 0.0001. Algoritmul oferă o varietate mult mai complexă de parametrii pe care îi poți modifica, de exemplu parametrii impuls de ordin 1 sau 2.

Al treilea parametru pe care trebuie să îl setam este numărul de epoci pe care dorim să îl realizăm. Desigur, acesta se poate seta cu o valoare mai mica și rerula codul de câte ori dorim, dar ideal este să îl setam cel puțin la jumătate din cât consideram că vom rula în total. În aceasta lucrare, am decis ca un număr de 250 de epoci pentru fiecare model, sunt suficiente să arătăm capabilitățile modelului de a învăța, a se adapta cerințelor și nici nu are o durata de rulare foarte mare. Folosind platforma Google Colaboratory, explicată în capitolul 2.3, amintim că platforma are destule limitări, ceea ce rezultă în timpi destul de mari de rulare pentru fiecare model.

O altă caracteristică pe care am implementat-o în această etapă de antrenare este tehnica de early stopping. Această tehnică constă în setarea unui parametru, numit răbdare la o valoare fixă, în cazul nostru fiind de 30 de epoci,o valoare foarte mare pentru parametrul care stochează cea mai bună valoare a funcției de pierdere și un numărător de epoci. Această caracteristică este utilă pentru a evita supra adaptarea la zgomot a sistemului. Mai exact, în timpul antrenarii, sistemul nu mai învață caracteristici utile și învăța doar din zgomotul imaginilor, care este un comportament nedorit. Modul de funcționare este unul destul de simplu, de la o iterație la alta, dacă funcția de pierdere pentru datele de testare nu scade, numărătorul este incrementat cu valoarea 1, iar bucla de rulare este întreruptă în momentul în care numărătorul ajunge la valoarea parametrului de răbdare setată. Chiar dacă bucla de rulare se întrerupe, avem construit un mecanism care salvează cel mai bun model întâlnit, iar în momentul în care

bucla se întrerupe, modelul cel mai bun este preîncărcat în variabila supranumită model și este pregătită să se repornească bucla pentru a se încerca continuarea învațării caracteristicilor noi.

Înainte de a începe bucla de rulare, este la fel de important să ne putem folosii la maxim de resursele oferite, ceea ce înseamnă că trebuie să transferam modelul și toate informațiile pe care acesta le deține, pe procesorul grafic și mai exact să ne folosim de arhitectura Cuda, explicată în capitolul 2.4 . Setarea acestor parametrii se poate vedea în urmatoarele linii de cod sau în Anexa 1.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model_load
model.to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
num_epochs=50

best_loss = 1000
patience = 30
counter = 0
```

Cea mai importantă parte din acest subcapitol este bucla de antrenare și testare. Înainte de a porni bucla, folosind funcția de timer, putem porni un cronometru pentru a evalua cât de mult a durat toată perioada de rulare. Folosim și o unealtă care ne arată progresul fiecărei epoci sub forma unei bare de progres. Această unealtă ne oferă și o idee despre cât durează fiecare iterație a buclei și realizează media de timp pe parcursul a tuturor buclelor.

Bucla principală are ca interval de rulare numărul de epoci. În interiorul acestei bucle se află alte două bucle, una pentru antrenare și una pentru testare. Înainte de a porni bucla de antrenare, trebuie să setăm modelul pentru a fi în modul de antrenare, ceea ce activează funcții și caracteristici specifice antrenării modelelor. După ce acest mod este activat, inițializăm variabila care stochează valoarea funcției de pierdere cu valoarea 0 pentru a avea un estimat corect pentru fiecare epocă.

Bucla de antrenare se realizează în gama dimensiunii structurii de dataloader, disponibili pentru antrenare, iar în această buclă se va prelua fiecare lot de date disponibil și se va trece prin următorul proces:

1. Mutarea imaginilor și a etichetelor pe dispozitivul disponibil, de obicei fiind unitatea grafică;
2. Resetarea gradienților la valoarea 0 pentru fiecare lot de date;
3. Trecerea imaginilor prin întreaga rețea neuronală pentru a obține predicțiile dorite;
4. Ștergerea unei dimensiuni create de funcția de pierdere pentru a ne asigura că predicțiile realizate de model sunt în concordanță cu dimensiunea etichetelor reale;
5. Calcularea funcției de pierdere dintre datele obținute la ieșirea rețelei neuronale și datele reale încărcate prin etichete;
6. Realizarea propagării inverse a valorii obținute în pasul anterior pentru a calcula gradienții tuturor parametrilor din model;
7. După acest tip de propagare inversă, este realizat pasul de optimizare folosind algoritmul setat la început, care modifică pe baza gradienților, parametrii interni ai modelului;
8. La final se realizează suma tuturor funcțiilor de pierdere realizată pentru cei 12 parametrii, din fiecare lot, iar după ce bucla de antrenare s-a terminat, realizăm ultima operație, împărțirea valorii funcției de pierdere obținută în bucla de antrenare, la dimensiunea structurii de dataloader de antrenare;

Procesul de testare este asemănător, diferența majoră este că în bucla de testare nu se realizează nici o adaptare a parametrilor modelului și este folosită strict pentru a observa cum se descurca modelul cu alte date, diferite de cele de antrenare. În această buclă se realizează

și salvarea celui mai bun model și incrementarea numărătorului pentru a verifica dacă modelul avansează și reușește să învețe.

La finalul buclei principale, avem și o linie de cod care ne printează rezultatele obținute în legătura cu funcția de pierdere, după fiecare epocă, pentru a putea urmări evoluția modelului. Datele afișate la nivelul acelei linii de cod au fost salvate și folosite pentru a realiza graficele cu evoluția modelelor.

4.4 Capabilitați de invatare și performanțe obținute în timpul antrenarii

În acest subcapitol, urmează să fie prezentate cele 3 modele antrenate. O să fie explicate pe rând performanțele obținute, evoluția în timp a modelelor, iar pentru o vizualizare mai ușoară, vor fi expuse graficele și tabelele rezultate.

4.4.1 Rezultate obtinute în urma unui model antrenat cu baza de date Kitti

Datorită dificultății de realizare al unui astfel de model, am optat să nu folosesc de la început o bază de date care provine de la o platformă aeriană autonomă. De aceea modelul de început pe care l-am realizat în această lucrare este bazat pe baza de date numita KITTI. Așa cum am menționat anterior în capitolul 2.5, această bază de date este obținută de la o platforma mobilă de sol, care navighează prin diferite medii. Având o bază de date relativ mare, am ales să folosesc cele mai mari 2 secvențe din cele 11 medii oferite. Această alegere a fost luată pe baza unor experimente anterioare în care s-a constatat un timp destul de mare de parcurgere a unei epoci.

Dimensiunile bazei de date au fost menționate în capitolul 4.2, dar o să fie prezentate pe scurt și aici. Avem 2 secvențe, fiecare cu 4541, respectiv 4661 de imagini. Avem deci la dispoziție un număr destul de mare de imagini pe care să le folosim pentru antrenare și testare. Numărul de epoci în care modelul a fost lăsat să învețe este de 250 epoci.

În urma antrenării acestui model, timpul de rulare pentru fiecare epocă a variat, dar folosind din librăria Tqdm, bara de progresie, am putut obține o estimare pentru valoarea medie a unei epoci. Această valoare a fost de 355 de secunde, adică aproximativ 6 minute pentru o epocă. Având în vedere numărul de epoci, putem calcula durata medie estimată a antrenării acestui model ca fiind de aproximativ 25 de ore de rulare. În timpul acestei rulări, valorile funcției de pierdere au fost salvate într-un document de tip Excel și au fost folosite pentru a realiza un grafic pentru o mai bună vizualizare a acestei funcții de-a lungul perioadei de învățare. Acest grafic este prezentat în figura 4.1. Datorită instabilității modelului de la o epocă la alta, valorile obținute au fost filtrare folosind un filtru de tip median cu lungimea de 5 eșantioane, folosind funcția MEDIAN din cadrul programului Excel. În urma acestei prelucrări, am obținut această figură.

Pentru început putem remarca că valorile obținute în prima fază a antrenarii, sunt destul de mari, mai ales funcția de pierdere pentru datele de test. Acest lucru este normal și de așteptat, deoarece modelul a fost preluat de la 0 și este imposibil să realizeze din prima o funcție atât de complexă, ca cea pe care o căutam noi. În continuare ne așteptăm ca aceste 2 funcții să scadă, ceea ce se și întâmplă. În primele 100 de epoci, am avut o scădere a funcțiilor de pierde foarte mare, ceea ce ne spune că modelul începe să înțeleagă ce funcție dorim să realizeze. În subcapitolul 4.3 am menționat faptul că pe parcursul tuturor perioadelor de antrenare, am folosit ca funcție de pierdere, eroarea medie pătratică. Se poate observa că modelul este pedepsit

cu un grad mult mai mare, ceea ce îl face să aibă aceasta scădere abruptă cât timp este departe de valorile dorite.

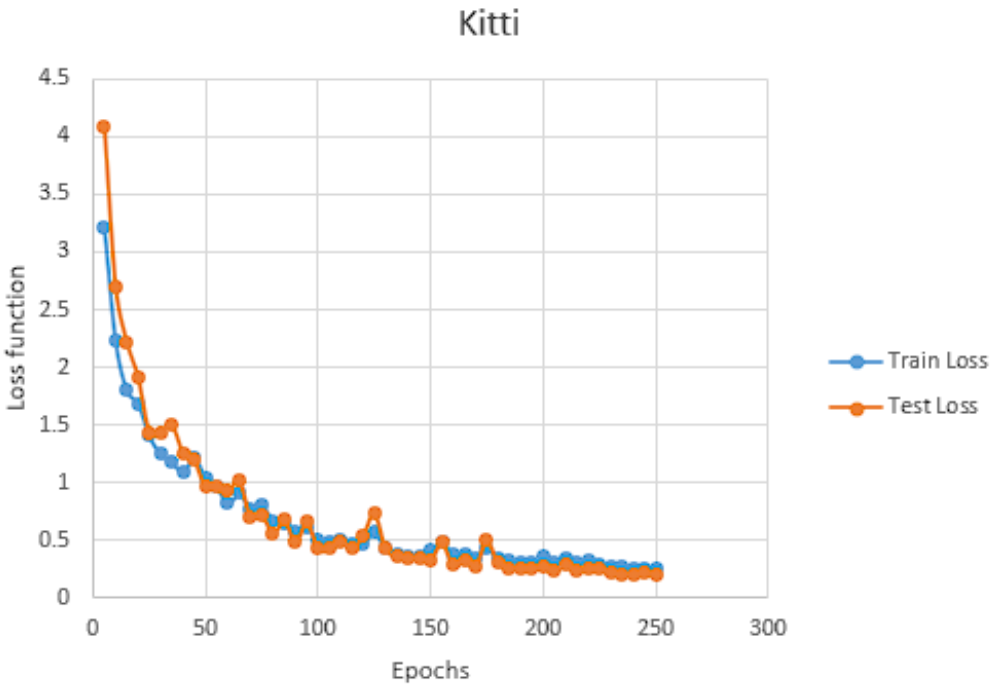

Figura 4.1: Evoluția funcțiilor de pierdere Kitti;

După ce s-a trecut de borna de 100 de epoci, funcția de pierdere nu mai are aceeași pantă de descreștere, dar este observabil cu ochiul liber că valoarea continuă să scadă. Modelul fiind mai aproape de valorile pe care ar trebui să le obțină, funcția de pierdere nu mai pedepsește atât de drastic modelul. Din cauza instabilității modelului, încă se pot vedea anumite porțiuni care nu sunt în concordanta cu scăderea dorită. Aceste anomalii apărute de-a lungul antrenării se datorează încercării modelului să găsească o cale mai bună către funcția dorită, pentru a creste performanța. În momentul în care modelul se îndepărtează de valorile reale, este pedepsit de funcția de pierdere, care îl trage în jos pentru a continua pe traseul optim.

Un alt aspect care trebuie discutat este faptul că funcțiile de pierdere au o alură asemănătoare. Cele 2 au aproximativ aceeași viteză de descreștere pe fiecare secțiune, cu diferite excepții care sunt corectate pe măsură ce modelul avansează. Pe finalul perioadei de antrenare, funcția de pierdere pentru datele de testare reușește chiar, să se afle sub cea pentru antrenare, dar această ordine nu este garantată. Pe viitor, în eventuala continuare a antrenării a modelului, ordinea poate să varieze.

În final, putem observa că cele 250 de epoci sunt suficiente pentru a vedea diferite secvențe în care modelul are un anumit comportament. Rezultatele sunt unele care arată că într-adevăr funcția de pierdere scade, ceea ce înseamnă că modelul este pe drumul cel bun pentru a realiza funcția dorită. Scăderea de la o valoare de aproximativ 4, spre o valoare apropiată de 0.2 arată că modelul și hiperparametrii săi sunt realizați corect și se pot descurca cu o astfel de sarcină. Avem prezentat în tabelul 4.1 și valorile funcției de pierdere pentru fiecare 50 de epoci.

Epocă	5	50	100	150	200	250
Funcția de pierdere antrenare	3,2096	1,0383	0,5029	0,4078	0,3589	0,2528
Funcția de pierdere testare	4,0786	0,9608	0,4287	0,3283	0,2643	0,208

Tabela 4.1: Funcțiile de pierdere Kitti

4.4.2 Rezultate obtinute în urma unui model antrenat cu baza de date NTU Viral

Următorul model antrenat este bazat pe imaginile și datele de la sol oferite de baza de date NTU VIRAL. Dacă în subcapitolul anterior, am folosit inițial o platformă mobilă de sol, în această parte a lucrării, vom arata și vom explica cum a evoluat modelul pentru o platformă mobilă aeriană. Modalitatea de capturare a imaginilor și prelucrarea lor a fost explicată în capitolul 2.6.

Din această bază de date am avut la dispoziție 9 secvențe, din care am ales 3 care provin din același mediu. Aceste 3 secvențe au câte 3986, 3209 și 1813 de imagini cu date de la sol. Aceste 9008 imagini selectate ne oferă o varietate suficient de mare pentru a încerca antrenarea modelului propus în capitolul 3.2. După rezultatele pozitive pe care le-am obținut anterior cu baza de date Kitti, în acest subcapitol încercam să antrenăm modelul de la 0, folosind noile secvențe din baza NTU VIRAL.

Antrenarea acestui model a fost împărțită în 2 etape, fiecare având câte 250 de epoci de antrenare. Scopul acestei împărțiri este de a avea o comparație a primei etape cu modelul antrenat cu baza de date Kitti prezentat anterior, iar cea de a doua etapă pentru a putea realiza o comparație corectă cu modelul care o să fie prezentat în următorul capitol.

S-au păstrat parametrii legați de funcția de pierdere și algoritmul pentru optimizarea modelului. Modificările realizate pentru a potrivi datele în modelul creat, au fost doar în cadrul transformărilor imaginilor, deoarece acestea erau de tip alb-negru. Aceste modificări sunt explicate în capitolul 3.1.2.

În continuare, urmează să prezentăm evoluția modelului pentru primele 250 de epoci. Putem remarca în urma rulării epocilor de antrenare, am avut ca rezultat inițial un timp de rulare mai scurt, de aproximativ 190 de secunde pe epocă. Este de menționat ca echipamentul hardware a rămas identic, adică pachetul de baza oferit de către Google Colaboratory. Această creștere de performanță este datorată tipului de imagini folosite. Fiind imagini alb-negru, chiar dacă aceeași informație a fost copiată pe toate cele 3 canalele de culoare, există mai puține caracteristici pe care le poate identifica modelul, ceea ce înseamnă că este mai ușor pentru model să găsească caracteristici utile. Dacă realizăm înmulțirea acestui timp cu numărul de epoci, obținem timpul de antrenare pentru această etapă, care este de aproximativ 13 ore. Putem vedea deja un avantaj destul de mare, deoarece în timpul în care am antrena modelul folosind baza de date Kitti, am putea antrena aproape un număr dublu de epoci pentru această bază de date. La fel ca în cazul modelului anterior, am salvat într-un document de tip Excel și în acest caz, toate valorile funcțiilor de pierdere pentru a putea vizualiza grafic performanțele modelului. Graficul realizat după antrenarea acestei baze de date este următorul 4.2 și a fost realizat la fel ca în cazul anterior.

Putem vedea deja o asemănare între cele 2 baze de date, legată de alura funcțiilor de pierdere.La fel ca anterior și în acest caz cu baze de date NTU, pornim de la o valoare relativ mare a funcției de pierdere. Valoare este mult mai mică decât cea din primul model antrenat ,dar ținând cont că discutăm despre o altfel de platformă folosită pentru înregistrarea datelor, nu ar trebuie să facem comparații directe între cele două valori numerice. De asemenea , valorile scăzute pot sa fie și din cauza similitudinii imaginilor obținute din platforma vehiculului aerian la momentul ridicării de la sol și al aterizării. Acest lucru nu schimbă totuși validitatea

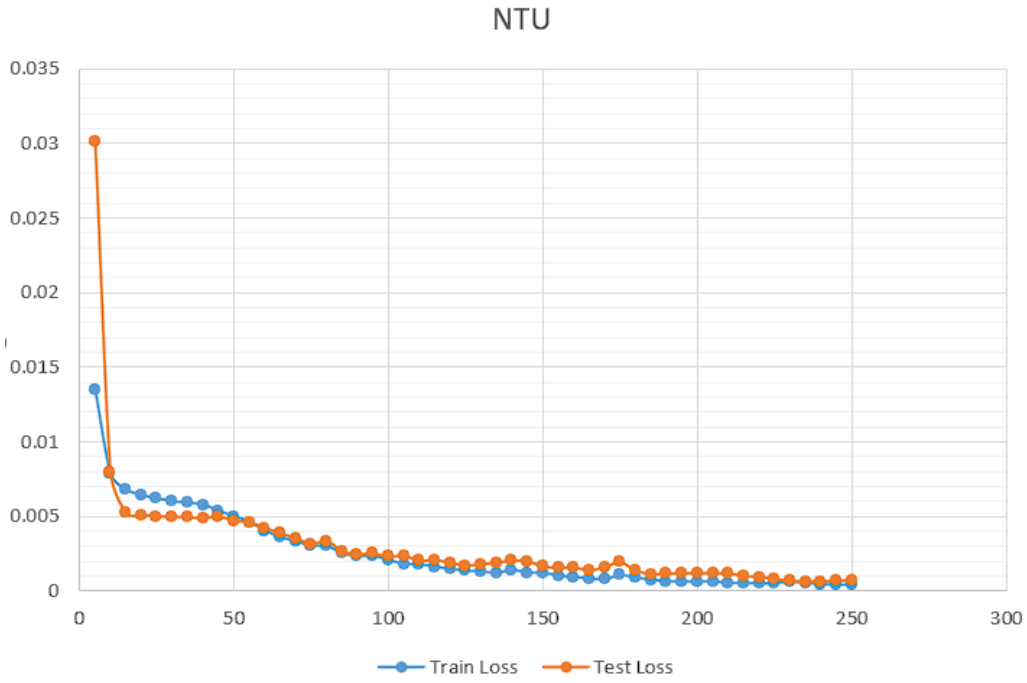

Figura 4.2: Evoluția functiilor de pierdere pentru primele 250 de epoci - NTU;

rezultatelor, deoarece avem un set de date destul de mare, iar rezultatele finale afirmă că modelul a reușit să învețe. Aceste valori inițial mari oferite de model sunt din cauza inițializării modelului, care pleacă de la 0 cu setarea parametrilor interni ai rețelei. Modelul începe să înțeleagă care este evoluția corectă în realizarea funcției, în jurul epocii 50, unde se poate observa că panta graficului se mărește și devine mai abruptă. Înainte de acest punct important, avem o descreștere brusca a valorilor și apoi o panta relativ lină.

De la punctul de 50 de epoci pană în jurul epocii 200 putem aproxima funcțiile de pierdere aproape ca pe niște dreapte cu panta negativă, ceea ce este un semn bun în privința performantelor. La finalul acestei secvente, panta graficului se reduce, dar continuă să scadă. Anomaliile care apăreau în timpul antrenării primul model s-au diminuat. În primul model chiar dacă avea aplicată funcția median, tot puteam observa porțiuni în care apăreau astfel de creșteri anormale. În cazul acestui model, avem o stabilitate mai mare a predicțiilor făcute.

Se poate observa faptul că gama ordinelor de mărime este diferită pentru cele 2 baze de date, dar descreșterea realizată în cadrul a 250 de epoci a fost prezentă în ambele cazuri. Observația care trebuie făcută este că cele 2 au scăzut cu un ordin de mărime relativ la punctul de start, respectiv de la valoarea de 4 la 0.2 pentru Kitti și de la valoarea de 0.03 la 0.007, cu mențiunea că ignorăm în cazul NTU primele valori care sunt foarte depărtate. Putem afirma că arhitectura rețelei neuronale de învățare reușește aproximativ aceeași putere de învățare în ambele cazuri.

Rămâne corectă și afirmația susținută în subcapitolul anterior legată de cele 2 funcții de pierdere. Cele 2 reușesc să scadă cu aproximativ aceeași alură și aceeași pantă pe aceleași porțiuni. Acest lucru indică faptul că modelul nu suferă de supra potrivirea datelor, adică încă reușește să găsească caracteristici utile în perechile de imagini și nu este influențat doar de zgomotul aferent imaginilor. În afară de primul interval de 50 de epoci, unde funcția de pierdere pentru antrenament are valori mai ridicate comparativ cu cea pentru testare, după acel interval, funcția de pierdere pentru antrenament rămâne permanent cu o valoare mai mică decât cea pentru testare. Acest aspect exprimă faptul că datele sunt mai greu de prelucrat și de extras caracteristici utile, din cauza tipului de date provenite din platforma aeriană, dar modelul nu este blocat la o anumită valoare, ceea ce indică o evolutie bună.

Pentru a păstra formatul ales, avem prezentat și sub formă de tabel, evoluția celor 2 funcții

de pierdere din 50 în 50 de epoci. Evoluția celor 2 funcții ne arată că modelul reușește să învețe funcția dorită și să se îmbunătățească treptat.

În continuare, o să fie prezentată evoluția modelului în etapa a două de antrenare, care constă în antrenarea modelului cu încă 250 de epoci. După cum se poate observa și în figura 4.2, funcțiile de pierdere și-au redus considerabil panta de descreștere pe care au avut-o inițial. Această caracteristică o putem observa și în parcursul antrenării din această etapă. Figura 4.3 reprezintă această evoluție pentru cele 250 de epoci.

Un prim aspect ușor de observat este instabilitatea celor 2 funcții de pierdere. Într-o perioadă relativ ridică de epoci, modelul a întâmpinat probleme în a ajusta parametrii interni pentru a reduce cele 2 funcții. Modelul nu mai reușește să reducă cu un ordin de mărime cele 2 valori, dar totuși o scădere a valorii este prezentă. Un alt aspect notabil este că diferența dintre cele 2 funcții de pierdere începe să crească. Dacă în prima etapă, cele 2 erau destul de apropiate, în această etapă, cele 2 au început să se distanțeze, dar totuși evoluția uneia rămâne strâns legată de cealaltă, ceea ce se poate vedea prin faptul că diferitele anomali de creștere bruscă care apar în timpul antrenării, apar simultan în intervale respective de epoci, ceea ce ne poate spune că modelul începe să aibă un comportament subtil de supra adaptare a datelor. Acest comportament a fost prezent și în cazul antrenării primului model, dar nu a fost atât de evident. Timpul de antrenare pentru această etapă a fost similar cu cel al primei etape, adică aproximativ 13 ore, ceea ce rezultă în final un timp de antrenare de 26 de ore pentru antrenarea acestui model.

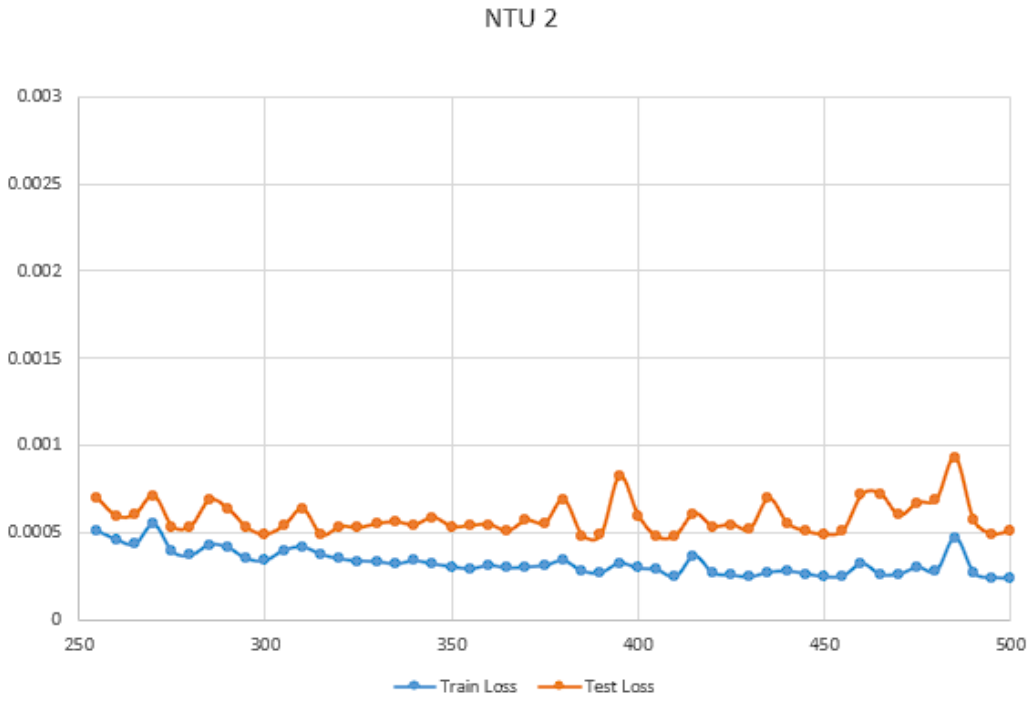

Figura 4.3: Evoluția funcțiilor de pierdere pentru urmatoarele 250 de epoci - NTU;

Având o imagine de ansamblu pentru fiecare 50 de epoci, așa cum este exemplificat în tabelul 4.2, am putea aproxima un număr de epoci în care funcțiile de pierdere au scăzut atât de mult încât eroarea este aproape insesizabilă, dar ar trebui să ținem cont și de faptul că uneori rețelele neuronale pot să fie imprevizibile și să nu se comporte pe anumite perioade de epoci exact cum ne-am aștepta. Ca exemplu putem avea chiar intervalul inițial, în care cele 2 au avut o descreștere rapidă, iar pe ultimele intervale, această scădere s-a redus drastic.

Epocă	5	50	100	150	200	250
Funcția de pierdere antrenare	0.01349	0.00501	0.00207	0.00123	0.00061	0.00042
Funcția de pierdere testare	0.03014	0.00473	0.00232	0.00165	0.00121	0.00074
Epocă	**300**	**350**	**400**	**450**	**500**	
Funcția de pierdere antrenare	0.00033	0.00030	0.00029	0.00024	0.000237	
Funcția de pierdere testare	0.00052	0.00053	0.00059	0.00048	0.00049	

Tabela 4.2: Funcțiile de pierdere NTU

4.4.3 Rezultate obtinute în urma unui model antrenat folosind ambele baze de date

Al treilea model antrenat în cadrul acestei lucrări este o combinație între cele 2 modele realizate anterior. Am dorit să aflăm dacă este necesar și benefic să plecăm de la un model antrenat cu imagini de la sol, iar apoi să continuăm antrenarea cu cele de la platforma mobilă aeriană. Pentru a realiza acest experiment și observațiile aferente, am procedat în felul următor. Avem deja modelul antrenat cu baza de date Kitti,explicate în subcapitolul anterior 4.4.1, care ne oferă practic, o bază de pornire pentru model, iar după această etapă, putem să antrenăm modelul în același stil folosit și explicat în subcapitolul 4.4.2.

O primă observație care trebuie făcută este faptul că modalitatea de comparație realizată, este între ultimele 250 de epoci ale modelului NTU și modelul care urmează să fie analizat în acest subcapitol. Realizăm această comparație, deoarece modelul din acest capitol a avut teoretic 500 de epoci de antrenare, dacă luam în calcul primele 250 de epoci realizate cu baza de date Kitti și încă 250 de epoci cu baza de date NTU. Realizăm această comparație pentru a putea sublinia diferențele dintre cele 2 modele care au fost antrenate folosind baza de date NTU.

Acest model dispune de o gamă variată de imagini pentru antrenare din cele 2 baze de date, dar segmentate, ceea ce înseamnă că nu le-a avut pe toate disponibile în același timp. Odată ce avem deja modelul antrenat cu baza de date Kitti, am putut să încărcam acești parametrii ai modelului folosind următoarele linii de cod, pe care le putem regăsii și în Anexa 1.

```
model_load=CNN_V1()
model_load.load_state_dict(torch.load(f=MODEL_BEST,map_location=torch.device('cpu')))
model_load = model_load.to(device)
for param in model_load.parameters():
    param.requires_grad = True
model_load
```

În continuarea urmează să prezentăm funcționalitatea codului menționat. Alocăm variabilei `model_load`, arhitectura rețelei neuronale, iar apoi din calea setată în variabila `MODEL_BEST`, se încarcă parametrii modelului preînvățat. Parametrii sunt încărcați inițial în memoria unității centrale de procesare, dar ulterior operației de încărcare, parametrii sunt mutați pe dispozitivul setat, adică memoria procesorului grafic. Ultimul pas este de a activa gradienții tuturor parametrilor încărcați, pentru a face posibilă modificarea în timpul pasului de optimizare.

După ce modelul a fost încărcat cu succes, metoda de antrenare nu s-a schimbat. Am păstrat aceleași perechi de imagini pentru a avea o comparație corectă între modelul care a fost antrenat cu ambele baze de date și cel simplu, antrenat doar cu NTU.S-au pastrat aceeași hiperparametrii, iar numărul de epoci a rămas constant la 250 de epoci. Graficul funcțiilor de pierdere pentru acest model este prezentat în figura 4.4.

Putem observa că modelul are o evoluție asemănătoare cu modelul realizat cu baza de date Kitti și prima etapă a modelului antrenat cu NTU. Pornește de la o valoare relativ ridicată pentru cele 2 funcții de pierdere, iar apoi în urma antrenării, valorile funcțiilor de pierdere

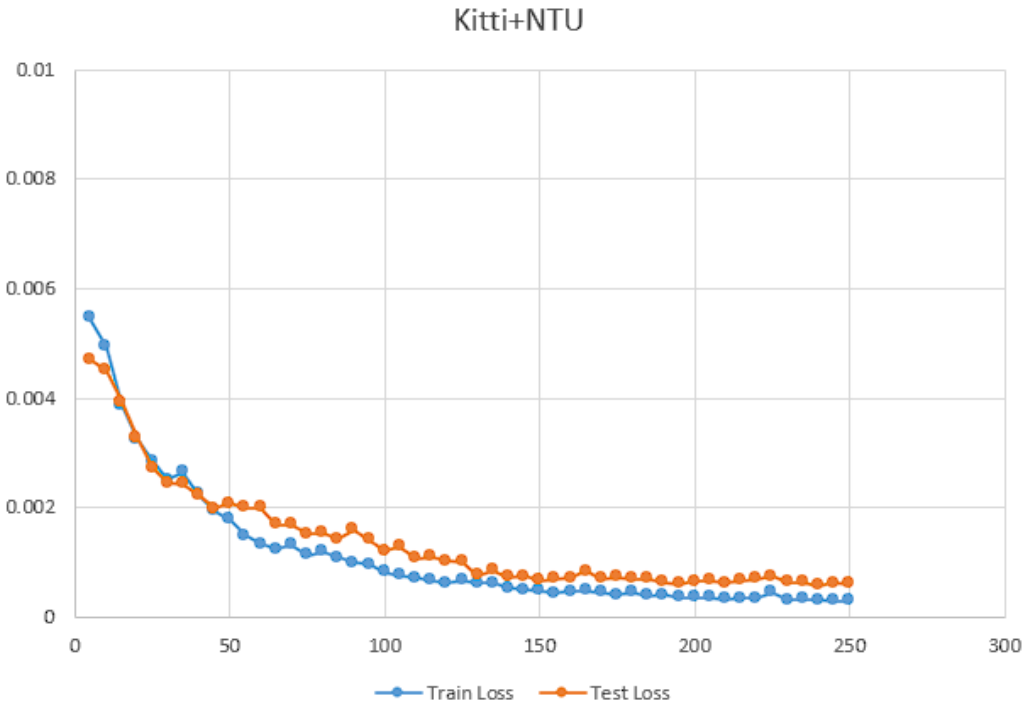

Figura 4.4: Evoluția functiilor de pierdere NTU și Kitti;

sunt reduse treptat. În intervalul de 100 de epoci, avem o scădere relativ ridicată. Cauzele acestei scăderi ridicate, au fost discutate în cele 2 capitole anterioare, fiind strâns legate de tipul funcțiilor de pierdere. În acest interval se observă că funcțiile de pierdere au o alură diferită, pe anumite intervale de epoci, dar după pragul de 100 de epoci, cele 2 încep să aibă un comportament asemănător, ceea ce înseamnă că modelul se stabilizează . După acel prag, avem o scădere relativ mică, dar este important că direcția în care se îndreaptă modelul este una dorită, corectă și nu apar semne de supra adaptare.

Comportamentul instabil remarcat în primele 100 de epoci, poate să fie provocat de parametrii modelului preaînvățat cu baza de date Kitti. Reamintim că modelul realizat cu această baza de date, avea acest comportament instabil, care a fost transmis mai departe noului model. În epocile de final, rețeaua reușește să realizeze o atenuare destul de bună a comportamentului instabil.

Un aspect importat pe care ar trebui să îl remarcăm este faptul că acest model nu reușește să reducă cele 2 funcții de pierdere mai mult decât modelul anterior antrenat. Etapa a doua a antrenării modelului doar cu NTU pleacă cu valorile celor 2 funcții din jurul valorii de 0.0007 și reușește să le reducă la o valoare de sub 0.0005. Modelul prezentat în acest capitol are o valoare de început mult mai mare, aproximativ 0.005 și reușește o reducere considerabilă, ajungând în jurul valorii de 0.0006 pe parcursul celor 250 de epoci, valori pe care le putem remarca și din tabelul 4.3. Este de menționat totuși și faptul că panta prezentată are o descreștere mai mare față de modelul antrenat doar cu NTU, ceea ce este posibil să ajute modelul în continuarea antrenamentului. Modelul anterior a fost predominant instabil, iar panta de descreștere mică, ceea ce înseamnă că antrenarea ulterioară o să fie mai dificilă. Chiar dacă modelul nu a redus cel mai mult cele 2 funcții, diferența nu este una uriașă și având în vedere instabilitatea prezentă în modelul anterior, performanțele acestui model ar putea să fie surprinzătoare.

Timpul de antrenare este un dezavantaj pentru acest model și această abordare aleasă. Pentru antrenarea modelului inițial cu baza de date Kitti, a durat aproximativ 25 de ore, iar pentru aceste 250 de epoci ,13 ore. În total avem aproximativ 38 de ore, ceea ce este considerabil mai mult timp de rulare comparativ cu modelul care a folosit doar baza de date NTU.

În concluzie, acest model a reușit să învețe caracteristici noi din perechile de imagini, a avut o evoluție bună a celor 2 funcții de pierdere și a reușit să reducă instabilitatea moștenită de la modelul antrenat cu baza de date Kitti. În capitolul următor o să se realizeze o comparație mult mai amplă între cele 2 modele pentru a alege care variantă este mai bună și mai eficientă pentru o eventuală continuare a antrenării modelului.

Epocă	**5**	**50**	**100**	**150**	**200**	**250**
Funcția de pierdere antrenare	0.00545	0.00178	0.00083	0.00048	0.0003	0.000297
Funcția de pierdere testare	0.00470	0.00207	0.00121	0.00067	0.00065	0.00061

Tabela 4.3: Funcțiile de pierdere NTU + Kitti

4.5 Performanțele celor 2 modele pentru micro-vehiculele aeriene

În acest subcapitol urmează să realizăm o comparație între ultimele 2 modele antrenate, pentru a putea trage o concluzie între nevoia de a pre antrena un model cu date de la o platformă mobilă de sol sau a începe antrenarea direct cu date provenite de la o platformă aeriană și a realiza mai multe epoci cu această metodă.

După cum a fost prezentat anterior, modelul care a fost antrenat cu ambele baze de date nu a reușit să atingă valori mai mici pentru funcțiile de pierdere, față de modelul care a pornit de la început cu baza de date NTU, dar acest aspect nu poate să ne ofere un răspuns concret pentru această comparație. Astfel pentru a realiza această comparație, avem nevoie să exportăm datele pe care modelul le produce într-un format ușor de citit și prelucrat, mai exact în formatul CSV. Pentru această sarcină am creat codul din Anexa 3 și ne-am folosit de funcția scrisă în Anexa 1, numită ”Preditions”.

Folosind funcția “Prediction”, putem exporta parametrii produși de model și să îi încărcăm în liste de variabile, unde putem să realizăm matricele de rotație și translație. Folosind aceste matrice, putem să cream traiectoria realizată de model sau să convertim matricea de rotație într-un format de unghiuri Euler. Matricea de translație nu necesită nici o transformare, ceea ce înseamnă că o putem folosii direct. Codul din Anexa 3, încarcă într-un document de tip CSV predicțiile pe care le realizează modelul, dar și datele reale pentru a putea calcula ulterior erorile relative. Un aspect important pe care trebuie să îl menționăm este că toți parametrii scriși în document, sunt relativi la momentul anterior, nu la punctul de început ”t=0”.

Pentru a compara cele 2 modele, cum am menționat și anterior, o să ne folosim de 6 parametrii esențiali pentru odometria vizuală. Folosind parametrii de translație și cele 3 unghiuri Euler, am putea reprezenta traiectoria realizată de platforma mobilă aeriană, dar având în vedere numărul redus de epoci de antrenare, modelele nu ating performanțe suficient de bune pentru a trasa traiectorii corecte. Chiar dacă traiectoriile realizate de model nu prezintă o modalitate corectă de evaluare, în stadiul actual de antrenare, putem compara cei 6 parametrii cu datele reale și să realizăm eroarea medie absolută relativă al fiecărui parametru.

Model și mediu testare	**Eroare X**	**Eroare Y**	**Eroare Z**
Model 1 Mediu cunoscut	8.51E+09	5.18E+09	8.17E+09
Model 2 Mediu cunoscut	7.95E+09	8.37E+09	8.81E+09
Model 1 Mediu necunoscut	3.20E+11	1.69E+11	1.58E+11
Model 2 Mediu necunoscut	2.31E+11	1.12E+11	1.90E+11

Tabela 4.4: Erori pentru XYZ

În tabelele 4.4 și 4.5, avem aceste erori calculate pentru 4 cazuri diferite, mai exact cele 2 modele antrenate în 2 medii diferite. Primul mediu ales este mediul din care am extras perechile de antrenare și testare, pentru a vedea cum se descurcă modelul într-un mediu cunoscut. Al doilea mediu ales pentru evaluarea modelelor este unul complet necunoscut modelelor, pentru a evalua și această situație.

De asemenea, pentru a putea face comparații rapide și pentru a ne referi ușor la fiecare model în parte, vom atribui numere modelelor. Astfel, vom considera modelul antrenat doar cu baza de date NTU ca fiind "Modelul 1", iar modelul antrenat cu ambele baze de date ca fiind "Modelul 2". Această abordare ne va permite să identificăm și să discutăm despre fiecare model într-un mod în care să nu existe confuzii.

Pentru început putem analiza tabelul 4.4, care ne prezintă eroarea de translație, care s-a acumulat pe tot parcursul perechilor de imagini din cele 2 secvențe video. Translația platformei mobile este împărțită în 3 parametrii, care reprezintă deplasarea platformei într-un sistem cu 3 axe de deplasare, "X Y Z". Se poate observa o diferență de aproape două ordine de mărime între mediile menționate anterior, mai exact mediul cunoscut și mediul necunoscut. Modelele se descurcă mult mai bine dacă parametrii lor au fost ajustați folosindu-se un mediu cunoscut în antrenare, comparativ cu încercarea de a face predicții pentru un mediu total necunoscut. Se poate observa și care dintre cele 2 modele a reușit să aibă o eroare mai mică pentru cele 3 axe de deplasare. Modelul 1 a reușit să aibă o eroare mai mică pentru 2 dintre cele 3 direcții de deplasare, față de modelul 2. Ultimul model amintit reușește să aibă o eroare mai mică doar pentru direcția X de deplasare, dar la o diferență relativ mică față de primul model.

Dacă analizăm erorile provenite din predicțiile mediului necunoscut, putem observa că situația s-a schimbat, iar la prima vedere putem spune că modelul numărul 2 reușește să aibă 2 direcții de deplasare cu erori mai mici. Dacă ne uităm mai atent, la fel ca în cazul mediului cunoscut, diferența nu este una mare care să ne confirme că modelul într-adevăr se descurcă mai bine comparativ cu modelul numărul 1.

Următorul tabel pe care trebuie să îl evaluam pentru a putea trage o concluzie în legătura cu capabilitățile modelelor este tabelul 4.5, care ne prezintă în aceeași manieră, cele 3 unghiuri Euler pentru ambele modele evaluate în cele 2 medii, unul cunoscut si unul necunoscut. Cele 3 unghiuri se numesc "Yaw", "Pitch" și "Roll" și descriu rotația unui obiect în jurul axei verticale, axei orizontale și axei longitudinale.

Pentru predicția acestor 3 parametrii utilizați pentru orientarea într-un spațiu tridimensional, nu mai avem o diferență majoră între cele 2 medii. Rezultatele au aproximativ același ordin de mărime, ceea ce înseamnă că pentru determinarea unghiurilor, modelul este independent de tipul de mediu folosit.

Model și mediu testare	**Eroare Yaw**	**Eroare Pitch**	**Eroare Roll**
Model 1 Mediu cunoscut	4.32E+08	2.32E+09	5.33E+08
Model 2 Mediu cunoscut	3.35E+08	1.88E+09	1.21E+08
Model 1 Mediu necunoscut	1.53E+08	1.31E+09	1.02E+09
Model 2 Mediu necunoscut	4.57E+08	2.67E+09	8.08E+08

Tabela 4.5: Erori pentru Yaw, Pitch și Roll

Modelul care a reușit să reducă cel mai mult erorile este modelul 1, pe secvențele de imagini provenite de la mediul necunoscut. Acesta realizează cel mai bine predicțiile pentru unghiurile Yaw și Pitch, dar în cazul celui de al 3 lea parametru, eroarea este cu un ordin de mărime mai mare decât toate celelalte combinații de modele si medii prezentate. Nici valorile acestui model pentru mediu cunoscut nu prezintă o variație foarte mare și chiar reușește să reducă eroarea

pentru unghiul Roll. De asemenea nici modelul 2 nu prezintă rezultate foarte depărtate de cele ale primului model, dar este ușor de observat că prezintă predicții bune doar în cazul mediului cunoscut, unde chiar este mai bun decât modelul numărul 1.

După ce am prezentat aceste două tabele și am realizat cateva comentarii legate de performanțele celor 2 modele putem începe o comparație directă. Cea mai bună variantă de a realiza această comparație este prin evaluarea erorii cumulate pentru cei 6 parametrii descriși cele 2 tabele.

Un prim lucru pe care îl putem afirma, este că alegerea unui model ca fiind castigator este dificilă în ceea ce privește performanța totală a modelelor, deoarece unul din modele a avut performanțe mai bune în predicția parametrilor de translație, iar altul în predicția unghiurilor de rotație.

În privința realizării parametrilor de translație, modelul numărul 1 s-a descurcat mult mai bine în mediul cunoscut, având o eroare mai mică pentru translația pe axa y și z, iar pe axa x fiind foarte apropiat de modelul 2. În ceea ce privește mediul necunoscut, modelul numărul 1 a avut valori ale erorii foarte apropiate de modelul numărul 2, ceea ce îl face cea mai bună variantă pentru a determina mișcarea de translație dintre 2 imagini, mai ales dacă modelul a fost testat cu secvențe din același mediu în care a fost și antrenat.

Unghiurile de rotație au evidențiat faptul că modelul nu mai este atât de sensibil la mediu, deoarece pe ambele secvențe de imagini, modelele au avut performanțe apropiate. Acest comportament este unul dorit și indică o evoluție bună a modelelor. Folosind secvența din mediul cunoscut, modelul numărul 2 a reușit să aibă predicții mai corecte și apropiate de cele reale comparativ cu modelul 1, dar diferențele nu sunt atât de mari cât să putem spune că modelul 2 este câștigătorul, mai ales dacă privim ce s-a întâmplat cu cele 2 modele când au primit secvențele din mediul necunoscut. În acest caz, modelul numărul 1, încă odată reușește să aibă performanțe foarte bune pentru 2 din 3 unghiuri, iar al 3 lea nu este foarte departe de valorile celuilalt model.

Un alt lucru pe care trebuie să îl luam în calcul pentru a alege cea mai bună variantă de a realiza sarcina propusă în această lucrare, este timpul de antrenare necesar. Timpii de rulare au fost menționați în capitolul anterior, dar nu a fost realizată o comparație directă între aceștia. Pentru antrenarea modelului numărul 1, au fost necesare 26 de ore de rulare, în care s-au realizat 500 de epoci, iar funcțiile de pierdere au fost reduse cel mai mult de acest model. În cazul modelului numărul 2, timpii de rulare au constat în aproximativ 25 de ore pentru antrenarea a 250 de epoci cu baza de date Kitti, iar apoi alte 13 ore pentru antrenarea cu baza de date NTU. În total avem 38 de ore, deci o diferență de aproximativ 12 ore între timpi de antrenare.

Acestea fiind spuse, trebuie să menționam și dezavantajele acestor modele și a acestei arhitecturi. Folosind o rețea neuronală convoluțională ca arhitectură, întâmpinam una din cele mai mari probleme legate de odometriea vizuală, adică acumularea erorilor. Încercând să realizăm o predicție asupra fiecărei perechi succesive de imagini, este inevitabil să ignoram erorile, chiar dacă acestea sunt foarte mici. Având o secvență de lungime mare, oricât de mici ar fi erorile, spre finalul secvenței, eroarea se acumulează și va provoca discrepanțe între datele reale și cele obținute de rețea. Un alt dezavantaj al acestor modele este numărul redus de epoci, deoarece sarcina este una de complexitate mare și ar avea nevoie de un număr imens de epoci și resurse mult mai multe pentru a realiza într-adevăr un model suficient de bun pentru al putea încarcă într-un vehicul aerian autonom . Din cauza numărului redus de epoci și al erorii care se acumulează constant, traiectoria pentru cele 2 secvențe folosite în timpul evaluării nu reușește să aibă aceeași alura ca cea reală.

Concluzia acestei comparații este una evidentă. Modelul numărul 1, cel care a fost antrenat doar cu baza de date NTU, se evidențiază ca fiind cel mai performant. Acesta a înregistrat aproximativ cele mai mici acumulari de erori în ambele situatii prezentate și prezintă în general

performanțe mai precise în ceea ce privește obiectivul dorit.

Până în acest punct al lucrarii am comparat și analizat performantele celor 2 modele de învățare adâncă, dar am prezentat și dezavantajele acestora. În urma rezultatelor prezentate, am obținut o perspectivă interesantă asupra capabilităților fiecărui model în abordarea sarcinii de odometrie vizuală. Concluziile clare și o perspectivă completă legată de contribuțiile aduse în cadrul realizării acestei lucrări, vor fi prezentate în următorul capitol, intitulat "Concluzii". De asemenea, vom prezenta și viitoarele posibilitati de continuare a acestei lucrari, cu scopul de a ajunge la performanțe suficient de bune pentru a putea echipa un micro-vehicul aerian autonom sau pentru a putea îmbunatății orice sarcină care implică odometria vizuală sau sarcini asemanătoare.

Concluzii

În cadrul acestui capitol, vor fi reluate pe scurt rezultatele prezentate în capitolul anterior 4.5 , legate de cele două modele de învățare adâncă pentru a sublinia concluzia finală în privința celei mai bune variante în procesul de cercetare al acestui domeniu. O să fie prezentată și o perspectivă de ansamblu asupra contribuțiilor aduse de către mine în privința realizării acestei lucrări specializată pe domeniul odometriei vizuale, dar și posibile direcții viitoare și îmbunătățirii în ceea ce privește performanța modelelor.

Pentru realizarea acestei lucrări, a fost nevoie de realizarea unor multiple etape de implementare și adaptări. Pentru început, contribuția cea mai mare a constat în realizarea antrenării rețelei convoluționale. În cadrul acestei etape esențiale, am testat efectul antrenării pe mai multe baze de date, cu scopul de a evidenția performanțele unei rețele neuronale convoluționale în contextul estimării parametrilor de mișcare. Am antrenat 3 modele diferite, unul bazat doar pe baza de date Kitti, cel de al doilea folosind doar baza de date NTU, iar în final s-a încercat extragerea performantelor unui model antrenat cu ambele baze de date disponibile. Modelele de care avem nevoie pentru a estima parametrii de mișcare al unui micro-vehicul aerian sunt ultimele 2 amintite. Acestea au avut la dispoziție un număr total de 500 de epoci de antrenare.

Anterior acestei etape, a fost nevoie de a realiza un script care să prelucreze pozițiile absolute pe care ni le ofereau baze de date, în poziții relative la momentul anterior. Pozițiile relative au fost apoi scrise într-un document de tip Excel alături de perechile de imagini și au fost folosite pentru a încarcă datele reale în rețeaua neuronală. Această transformare este explicată în capitolul 3.1.1 .

O altă etapă în care am contribuit este găsirea și adaptarea unei arhitecturi de rețea neuronală convoluțională. Astfel, modelele create au la baza arhitectura ResNet50, dar a fost nevoie să realizam adaptarea primului strat din rețea pentru a accepta o structura de date cu 6 canale de culori, care reprezintă perechea de imagini suprapuse pe canalul de culoare. Această arhitectură este în general folosită pentru sarcinile de clasificare, așa că a fost nevoie de realizarea unui bloc adițional de straturi în care să realizăm partea de regresie. Această arhitectură este prezentată în amănunt în capitolul 3.2 . Alte contribuții care merită menționate sunt realizarea structurilor de date numite dataloader, care au fost folosite pentru a realiza încărcarea datelor din bazele de date in rețeaua neuronală. Au fost de asemenea, realizate diferite funcții sau scripturi de prelucrare a datelor pentru a putea configura și duce la capăt realizarea acestei sarcini de odometrie vizuală.

Acestea fiind contribuțiile cele mai importante, putem trece și la realizarea concluziei finale legate de cea mai bună modalitate de antrenare a unui model de rețea neuronală cu scopul de a estima parametrii de mișcare. Menționăm faptul că o să păstram denumirile pe care le-am atribuit anterior celor 2 modele. Modelul 1 este modelul antrenat doar cu baza de date NTU, iar modelul 2 este antrenat cu combinația dintre baza de date NTU și Kitti. La fel cum am evidențiat în capitolul anterior, este destul de dificil să alegem modelul cel mai performant, la prima vedere, dar ținând cont de observațiile realizate anterior, răspunsul la această întrebare devine mult mai clar. Modelul 1 se descurcă mult mai bine în estimarea mișcării de translație, dar in cazul estimării mișcării de rotație, cele 2 modele sunt mult mai apropiate. Chiar dacă erorile absolute relative medii sunt apropiate, tot modelul 1 reușește să le reducă mai mult. Timpul de rulare pentru modelul 1 este de asemenea mai redus comparativ cu cel pentru modelul numărul 2, iar funcția de pierdere este redusă mai mult, tot de către modelul 1. După

aceste observații pe realizate pe scurt, dar și cele realizate mai amplu în capitolul 4.5 , putem să tragem concluzia că modelul 1, adică cel antrenat doar cu baza de date NTU reușește să fie mai performant, într-o perioada mai mica de tip, dar trebuie menționat faptul că pe finalul perioadei de antrenare, modelul a avut semne de supra adaptare.

În concluzie, este mult mai optim să antrenam o astfel de rețea neuronală cu date provenite direct dintr-o baza de date special creată pentru micro-vehicule aeriene, deoarece timpul de antrenare este mai redus, erorile cumulate au fost mai mici, iar acest model a reușit să reducă cel mai mult funcțiile de pierdere. Pentru a maximiza potențialul său și a aduce îmbunătățiri ulterioare, putem continua antrenarea acestui model pe parcursul a mai multor epoci și cu o gama mai variată de imagini. Pentru a adresa problema acumulări de erori, o variantă care s-ar putea arata foarte eficient în acest domeniu, este schimbarea arhitecturii CNN cu una recursivă de tipul RNN sau CRNN, în așa fel încât modelul să își ajusteze parametrii, nu doar în funcție de fiecare imagine, ci și de cele anterioare. Aplicând aceste îmbunătățiri am putea obține o rețea capabilă să fie integrată într-un astfel de micro-vehicul aerian sau orice altă sarcină din acest domeniu.

Bibliografie

[1] "Neural network." https://commons.wikimedia.org/wiki/File:Neural_network.svg. Accessed: 2023-15-3.

[2] "Basic convolutional neural network." https://www.oreilly.com/library/view/neural-network-projects/9781789138900/8e87ad66-6de3-4275-81a4-62b54436bf16.xhtml. Accessed: 2023-16-3.

[3] "Remote sensing." https://www.mdpi.com/remotesensing/remotesensing-11-00067/article_deploy/html/images/remotesensing-11-00067-g003.png. Accessed: 2023-20-3.

[4] "Rgb image consisting of three layers.." https://www.researchgate.net/publication/350817724/figure/fig1/AS:1021404615036931@1620533245732/RGB-image-consisting-of-three-layers.png. Accessed: 2023-27-4.

[5] J. D. Kelleher, *Deep Learning*. The MIT Press, 2019.

[6] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458v2*, 2015.

[7] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, p. 1897, Oct 2016.

[8] A. Graves, S. Lim, T. Fagan, and K. P. McFall, "Visual odometry using convolutional neural networks," *The Kennesaw Journal of Undergraduate Research*, vol. 5, no. 3, p. 5, 2017.

[9] "What is python? executive summary." https://www.python.org/doc/essays/blurb/. Accessed: 2023-21-4.

[10] "Python (in machine learning)." https://corporatefinanceinstitute.com/resources/data-science/python-in-machine-learning/. Accessed: 2023-21-4.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[12] T. Carneiro, R. V. Medeiros Da NóBrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, and P. P. R. Filho, "Performance analysis of google colaboratory as a tool for accelerating deep learning applications," *IEEE Access*, vol. 6, pp. 61677–61685, 2018.

[13] J. Ghorpade, J. Parande, M. Kulkarni, and A. Bawaskar, "GPGPU processing in CUDA architecture," *CoRR*, vol. abs/1202.4347, 2012.

[14] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[15] T. Nguyen, S. Yuan, M. Cao, Y. Lyu, T. H. Nguyen, and L. Xie, "NTU VIRAL: A visual-inertial-ranging-lidar dataset, from an aerial vehicle viewpoint," *CoRR*, vol. abs/2202.00379, 2022.

[16] T.-M. Nguyen, S. Yuan, M. Cao, Y. Lyu, T. H. Nguyen, and L. Xie, "Ntu viral: A visual-inertial-ranging-lidar dataset, from an aerial vehicle viewpoint," *The International Journal of Robotics Research*, vol. 41, no. 3, pp. 270–280, 2022.

[17] V. Mohanty, S. Agrawal, S. Datta, A. Ghosh, V. D. Sharma, and D. Chakravarty, "Deepvo: A deep learning approach for monocular visual odometry," *CoRR*, vol. abs/1611.06069, 2016.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[19] "Pytorch building blocks." `https://pytorch.org/docs/stable/nn.html#loss-functions`. Accessed: 2023-5-5.

Anexa 1
Script Principal pentru rețeaua neuronală

```
import random
import torch
import csv
from torch import nn
from pathlib import Path
import os
import pandas as pd
from PIL import Image
from torchvision import datasets, transforms
from torchvision.transforms import ToTensor
from sklearn.model_selection import train_test_split
from torch.utils.data import Subset, DataLoader ,Dataset
import matplotlib.pyplot as plt
import numpy as np
from torchvision.utils import make_grid
import os
from google.colab import drive
import smtplib
from torch.utils.data import ConcatDataset
device="cuda" if torch.cuda.is_available else "cpu"
#device="cpu"
print(device)
drive.mount('/content/drive')
! gdown --id 1Ash8ZjF8tRPqv1cjod-1Xqb30X0IZM2Y
! gdown --id 1gQQFDiDVQqyMyHPW9UN05IYonU5-Pd5s
!unrar x dl.rar
!unrar x bd.rar
data_path = Path('/content/drive/MyDrive/Colab Notebooks/data')
licenta_path=data_path/"licenta"
print(licenta_path)

img_paths_train_00='00/image_2'

img_paths_train_02='02/image_2'


img_paths_test='test'
labels_train_00="dl_00.csv"
labels_train_02="dl_02.csv"
traslation_path=data_path/"licenta/Translation"
transform= transforms.Compose([
    transforms.CenterCrop(376),
    transforms.Resize((128)),
    transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.4),
    transforms.ToTensor(),
    transforms.Normalize(0,1)
])


class CustomDataset(Dataset):
    def __init__(self, directory, csv_file, transform=None):
        self.directory = directory
        self.transform = transform
        self.labels = pd.read_csv(csv_file,header=None)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        img1_name = os.path.join(self.directory, self.labels.iloc[idx, 0])
        img2_name = os.path.join(self.directory, self.labels.iloc[idx, 1])

        label = [self.labels.iloc[idx, i] for i in range(2, 14)]
```

```
        label = torch.tensor(label, dtype=torch.float32)

        img1 = Image.open(img1_name)
        img2 = Image.open(img2_name)

        if self.transform:
            img1 = self.transform(img1)
            img2 = self.transform(img2)
        img = torch.cat([img1, img2], dim=0)

        return img,label

seed = 27
random.seed(seed)
torch.manual_seed(seed)
dataset_00=CustomDataset(img_paths_train_00,labels_train_00,transform)

dataset_02=CustomDataset(img_paths_train_02,labels_train_02,transform)

print(len(dataset_02))
print(len(dataset_00))
indices_0 = range(len(dataset_00))
indices_2 = range(len(dataset_02))

train_indices_0, test_indices_0 = train_test_split(indices_0, test_size=0.2, shuffle=True)
train_indices_2, test_indices_2 = train_test_split(indices_2, test_size=0.2, shuffle=True)
train_subset_0 = Subset(dataset_00, train_indices_0)
test_subset_0 = Subset(dataset_00, test_indices_0)
train_subset_2 = Subset(dataset_02, train_indices_2)
test_subset_2 = Subset(dataset_02, test_indices_2)
dataset_train=ConcatDataset([train_subset_0,train_subset_2])
dataset_test=ConcatDataset([test_subset_0,test_subset_2])

batch_size=64
train_loader = DataLoader(dataset_train, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(dataset_test, batch_size=batch_size, shuffle=False)
test_eval_1 = DataLoader(dataset_00, batch_size=batch_size, shuffle=False)
test_eval_2 = DataLoader(dataset_02, batch_size=batch_size, shuffle=False)
print(len(train_loader))
print(len(test_loader))
print(len(test_eval_1))
print(len(test_eval_2))
import torch.nn as nn
import torchvision.models as models
import torch.nn.functional as F

class CNN_V1(nn.Module):
    def __init__(self):
        super(CNN_V1, self).__init__()

        pretrained_model = models.resnet50(pretrained=True)

        pretrained_model.conv1 = nn.Conv2d(6, 64, kernel_size=7, stride=2, padding=3, bias=False)

        self.regression = nn.Sequential(
            nn.Linear(1000, 2048),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(2048, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),


            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 2048),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(2048, 1024),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(1024, 512),
```

```
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(512, 12),
        )

        self.features = pretrained_model
        for name, param in self.features.named_parameters():
            if name.startswith('layer4'):
                param.requires_grad = True
            elif name.startswith('layer3'):
                param.requires_grad = True
            else :
                param.requires_grad = False

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.regression(x)
        return x

model_1=CNN_V1()
model_1
MODEL_PATH =Path('/content/drive/MyDrive/Colab Notebooks/models')
MODEL_PATH.mkdir(parents=True,
                 exist_ok=True
)

MODEL_NAME = "licenta_model_v1.pth"
MODEL_NAME_LOOP ='best_model_v1.pth'
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME
MODEL_BEST=MODEL_PATH/ MODEL_NAME_LOOP

def predict(model, dataloader, device,path):
    model.eval()
    preds = [[] for i in range(12)]

    with torch.no_grad():
        for inputs, _ in dataloader:
            inputs = inputs.to(device)
            outputs = model(inputs)
            for i in range(12):
                preds[i].extend(outputs[:, i].cpu().tolist())


    with open(path, 'w', newline='') as file:
        writer = csv.writer(file)
        for i in range(len(preds[0])):
            row = [preds[j][i] for j in range(12)]
            writer.writerow(row)

from timeit import default_timer as timer
from tqdm.auto import tqdm
import torch.optim as optim


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model_load
model.to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
num_epochs=50

start_time = timer()
best_loss = 1000
patience = 30
counter = 0


for epoch in tqdm(range(num_epochs)):

    model.train()
    train_loss = 0.0
    for batch, (img, labels) in enumerate(train_loader):
        img = img.to(device)
        labels = labels.to(device)

```

```
        optimizer.zero_grad()
        outputs = model(img)
        outputs=outputs.squeeze(1)

        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    train_loss /= len(train_loader)

    model.eval()
    test_loss = 0.0
    with torch.inference_mode():
        for batch, (img, labels) in enumerate(test_loader):
            img = img.to(device)
            labels = labels.to(device)

            outputs = model(img)
            outputs=outputs.squeeze(1)

            loss = criterion(outputs, labels)

            test_loss += loss.item()

    test_loss /= len(test_loader)
    print('Epoch [{}/{}], Train Loss: {:.4f}, Test Loss: {:.4f}'
          .format(epoch+1, num_epochs, train_loss, test_loss))

    if test_loss < best_loss:
        best_loss = test_loss
        counter = 0

        print('save')
        torch.save(model.state_dict(),f=MODEL_BEST )
    else:
        counter += 1
        if counter >= patience:

            model.load_state_dict(torch.load(f=MODEL_BEST))
            break

end_time=timer()
print(f"Total training time: {end_time-start_time:.3f} seconds")
print(f"Saving model to: {MODEL_SAVE_PATH}")
torch.save(obj=model.state_dict(), f=MODEL_SAVE_PATH)
```

Anexa 2
Script matrice transformare

In această anexă este prezentată metoda de prelucrare a matricelor de transformare

```
import numpy as np
import csv
from google.colab import drive
from pathlib import Path
import os
drive.mount('/content/drive')
data_path = Path('/content/drive/MyDrive/Colab Notebooks/data')
folder_path=data_path/'licenta/Translation'
csv_1=folder_path/'eee_0.csv'
csv_2=folder_path/'eee_0_f.csv'
csv_3=folder_path/'eee_1.csv'
csv_4=folder_path/'eee_1_f.csv'
csv_5=folder_path/'eee_2.csv'
csv_6=folder_path/'eee_2_f.csv'

r00=[]
r01=[]
r02=[]
t0=[]
r10=[]
r11=[]
r12=[]
t1=[]
r20=[]
r21=[]
r22=[]
t2=[]
with open(csv_5,'r',encoding='utf-8-sig') as file:
  reader=csv.reader(file)
  for(row) in reader:
    r00.append(row[0])
    r01.append(row[1])
    r02.append(row[2])
    t0.append(row[3])
    r10.append(row[4])
    r11.append(row[5])
    r12.append(row[6])
    t1.append(row[7])
    r20.append(row[8])
    r21.append(row[9])
    r22.append(row[10])
    t2.append(row[11])
r00 = np.array(r00, dtype=np.float64)
r01 = np.array(r01, dtype=np.float64)
r02 = np.array(r02, dtype=np.float64)
t0 = np.array(t0, dtype=np.float64)
r10 = np.array(r10, dtype=np.float64)
r11 = np.array(r11, dtype=np.float64)
r12 = np.array(r12, dtype=np.float64)
t1 = np.array(t1, dtype=np.float64)
r20 = np.array(r20, dtype=np.float64)
r21 = np.array(r21, dtype=np.float64)
r22 = np.array(r22, dtype=np.float64)
t2 = np.array(t2, dtype=np.float64)
os.remove(csv_4)
rng=len(r00)-1

for i in range(rng):
  mat_N=np.array([[r00[i+1],r01[i+1],r02[i+1],t0[i+1]],[r10[i+1],r11[i+1],r12[i+1],t1[i+1]],[r20[i
    +1],r21[i+1],r22[i+1],t2[i+1]],[0,0,0,1]] ,dtype=np.float64)
  mat_N_1=np.array([[r00[i],r01[i],r02[i],t0[i]],[r10[i],r11[i],r12[i],t1[i]],[r20[i],r21[i],r22[i],
    t2[i]],[0,0,0,1]],dtype=np.float64)
```

```
inv_mat_N_1=np.linalg.inv(mat_N_1)
mat_f=np.matmul(mat_N,inv_mat_N_1)
vec_f=mat_f.reshape(-1)


with open(csv_6, mode='a', newline='') as file:
  writer = csv.writer(file)
  writer.writerow([vec_f[0], vec_f[1], vec_f[2], vec_f[3], vec_f[4], vec_f[5], vec_f[6], vec_f[7],
   vec_f[8], vec_f[9], vec_f[10], vec_f[11]])
```

Anexa 3
Script import și export al parametrilor micro vehiculelor aeriene

```
import numpy as np
import csv
from google.colab import drive
from pathlib import Path
import os
drive.mount('/content/drive')
data_path = Path('/content/drive/MyDrive/Colab Notebooks/data')
folder_path=data_path/'licenta/Translation'
csv_1=folder_path/'eee_0_f.csv'
csv_2=folder_path/'nya_f.csv'

csv_3=folder_path/'Predictions/predict_ntu_1.csv'
csv_4=folder_path/'Predictions/predict_ntu_kitti_1.csv'

csv_5=folder_path/'Predictions/predict_ntu_new_data.csv'
csv_6=folder_path/'Predictions/predict_ntu_kitti_new_data.csv'


csv_tr1=folder_path/'sit1.csv'
csv_tr2=folder_path/'sit2.csv'
csv_tr3=folder_path/'sit3.csv'
csv_tr4=folder_path/'sit4.csv'
r00=[]
r01=[]
r02=[]
t0=[]
r10=[]
r11=[]
r12=[]
t1=[]
r20=[]
r21=[]
r22=[]
t2=[]
with open(csv_6,'r',encoding='utf-8-sig') as file:
  reader=csv.reader(file)
  for(row) in reader:
    r00.append(row[0])
    r01.append(row[1])
    r02.append(row[2])
    t0.append(row[3])
    r10.append(row[4])
    r11.append(row[5])
    r12.append(row[6])
    t1.append(row[7])
    r20.append(row[8])
    r21.append(row[9])
    r22.append(row[10])
    t2.append(row[11])
r00 = np.array(r00, dtype=np.float64)
r01 = np.array(r01, dtype=np.float64)
r02 = np.array(r02, dtype=np.float64)
t0 = np.array(t0, dtype=np.float64)
r10 = np.array(r10, dtype=np.float64)
r11 = np.array(r11, dtype=np.float64)
r12 = np.array(r12, dtype=np.float64)
t1 = np.array(t1, dtype=np.float64)
r20 = np.array(r20, dtype=np.float64)
r21 = np.array(r21, dtype=np.float64)
r22 = np.array(r22, dtype=np.float64)
t2 = np.array(t2, dtype=np.float64)
rot_matrix = np.empty((len(r00), 3, 3))
```

```
rot_matrix[:, 0, 0] = r00
rot_matrix[:, 0, 1] = r01
rot_matrix[:, 0, 2] = r02
rot_matrix[:, 1, 0] = r10
rot_matrix[:, 1, 1] = r11
rot_matrix[:, 1, 2] = r12
rot_matrix[:, 2, 0] = r20
rot_matrix[:, 2, 1] = r21
rot_matrix[:, 2, 2] = r22
trans_matrix=np.empty((len(r00),3))
trans_matrix[:, 0] = t0
trans_matrix[:, 1] = t1
trans_matrix[:, 2] = t2

r00_real=[]
r01_real=[]
r02_real=[]
t0_real=[]
r10_real=[]
r11_real=[]
r12_real=[]
t1_real=[]
r20_real=[]
r21_real=[]
r22_real=[]
t2_real=[]
with open(csv_2,'r',encoding='utf-8-sig') as file:
  reader=csv.reader(file)
  for(row) in reader:
    r00_real.append(row[0])
    r01_real.append(row[1])
    r02_real.append(row[2])
    t0_real.append(row[3])
    r10_real.append(row[4])
    r11_real.append(row[5])
    r12_real.append(row[6])
    t1_real.append(row[7])
    r20_real.append(row[8])
    r21_real.append(row[9])
    r22_real.append(row[10])
    t2_real.append(row[11])
r00_real = np.array(r00_real, dtype=np.float64)
r01_real = np.array(r01_real, dtype=np.float64)
r02_real = np.array(r02_real, dtype=np.float64)
t0_real = np.array(t0_real, dtype=np.float64)
r10_real = np.array(r10_real, dtype=np.float64)
r11_real = np.array(r11_real, dtype=np.float64)
r12_real = np.array(r12_real, dtype=np.float64)
t1_real = np.array(t1_real, dtype=np.float64)
r20_real = np.array(r20_real, dtype=np.float64)
r21_real = np.array(r21_real, dtype=np.float64)
r22_real = np.array(r22_real, dtype=np.float64)
t2_real = np.array(t2_real, dtype=np.float64)
rot_matrix_real = np.empty((len(r00_real), 3, 3))
rot_matrix_real[:, 0, 0] = r00_real
rot_matrix_real[:, 0, 1] = r01_real
rot_matrix_real[:, 0, 2] = r02_real
rot_matrix_real[:, 1, 0] = r10_real
rot_matrix_real[:, 1, 1] = r11_real
rot_matrix_real[:, 1, 2] = r12_real
rot_matrix_real[:, 2, 0] = r20_real
rot_matrix_real[:, 2, 1] = r21_real
rot_matrix_real[:, 2, 2] = r22_real
trans_matrix_real=np.empty((len(r00_real),3))
trans_matrix_real[:, 0] = t0_real
trans_matrix_real[:, 1] = t1_real
trans_matrix_real[:, 2] = t2_real

from scipy.spatial.transform import Rotation

with open(csv_tr4, mode='a', newline='') as file:
      writer = csv.writer(file)
      writer.writerow(["Pair","Translation X","Translation Y","Translation Z",'Yaw','Pitch','Roll','
    ','Translation X Real","Translation Y Real","Translation Z Real",'Yaw Real','Pitch Real','Roll
    Real'])
for i in range(len(rot_matrix)):
```

```
137     r = Rotation.from_matrix(rot_matrix[i])
138     euler_angles = r.as_euler('xyz')
139     r_real = Rotation.from_matrix(rot_matrix_real[i])
140     euler_angles_r = r_real.as_euler('xyz')
141
142     with open(csv_tr4, mode='a', newline='') as file:
143       writer = csv.writer(file)
144       writer.writerow(["",trans_matrix[i, 0],trans_matrix[i, 1],trans_matrix[i, 2],euler_angles[0],
        euler_angles[1],euler_angles[2],"",trans_matrix_real[i, 0],trans_matrix_real[i, 1],
        trans_matrix_real[i, 2],euler_angles_r[0],euler_angles_r[1],euler_angles_r[2]])
```