



Departamentul Automatică și Informatică Industrială
Facultatea de Automatică și Calculatoare
Universitatea POLITEHNICA din București



LUCRARE DE DIPLOMĂ

**Sistem suport dedicat monitorizării
siguranței muncii pe un șantier**

Coordonator științific

Conf. dr. ing. Ștefan MOCANU

Absolvent

Andrei-Cosmin MARIA

2020

Rezumatul lucrării

Sănătatea și siguranța în muncă este domeniul care se ocupă cu reglementarea activităților periculoase ce pot provoca rănirea, îmbolnăvirea sau, chiar decesul angajatului și care pot periclita produsul muncii. Un prim pas important pentru prevenirea sau diminuarea impactului accidentelor la locul de muncă îl reprezintă purtarea echipamentului de protecție personală. În lucrarea de față voi prezenta cum, vederea artificială poate fi folosită pentru soluționarea acestei probleme, prin crearea unui sistem de supraveghere și asistență dedicat monitorizării activităților de pe șantierele de construcții. Principalul obiectiv al proiectului este identificarea în timp real a unor elemente de siguranță obligatorii din vestimentația muncitorilor, precum căștile de protecție sau vestele reflectorizante. Lucrarea îmbină atât elemente de teorie, necesare înțelegerii modului de funcționare al algoritmului de detecție, cât și partea practică, cu explicarea pașilor pe care i-am urmat în procesul de implementare, iar în final, am realizat o analiză comparativă între rezultatele mai multor modele matematice pe care le-am obținut. Pentru antrenarea modelelor am folosit un set de date virtual, generat într-un joc video (GTA V) și algoritmul de identificare YOLO, care face parte din clasa metodelor „state of the art” pentru detecția de obiecte.

CUPRINS

| | |
|--|-------------|
| Glosar | vi |
| Terminologie | vi |
| Listă de abrevieri | vii |
| Listă de figuri | viii |
| Listă de tabele..... | x |
| 1. Introducere | 1 |
| 1.1. Context..... | 1 |
| 1.2. Problema identificată | 1 |
| 1.3. Soluția propusă..... | 2 |
| 1.4. Obiective | 2 |
| 1.5. Structura lucrării | 2 |
| 2. Noțiuni Teoretice..... | 3 |
| 2.1. Prezentare generală | 3 |
| 2.2. Machine Learning | 4 |
| 2.2.1. Intuiție | 4 |
| 2.2.2. Regresie vs Clasificare..... | 6 |
| 2.2.3. Funcția obiectiv..... | 7 |
| 2.2.4. Procesul de antrenare | 8 |
| 2.2.5. Probleme de convergență | 9 |
| 2.3. Deep Learning..... | 10 |
| 2.3.1. Neuronul | 10 |
| 2.3.2. Tipuri de straturi | 11 |
| 2.3.2.1. Rețele neurale dense | 11 |
| 2.3.2.2. Rețele neurale convoluționale | 11 |
| 2.3.2.3. Straturi de pooling | 14 |
| 2.3.3. Funcții de activare..... | 14 |
| 2.3.4. Forward & Backward propagation..... | 16 |
| 2.3.5. Tehnici de creștere a performanței..... | 17 |
| 2.3.6. Overfitting vs Underfitting | 18 |
| 2.3.7. Arhitecturi populare | 20 |
| 2.4. Detecția de obiecte..... | 21 |

| | | |
|-----------|--|-----------|
| 2.4.1. | Metoda ferestrei glisante..... | 22 |
| 2.4.2. | You Only Look Once..... | 22 |
| 2.4.2.1. | Principiul de funcționare | 23 |
| 2.4.2.2. | Arhitectura și evoluția modelului | 25 |
| 2.5. | Recunoașterea facială..... | 26 |
| 2.6. | Urmărirea de obiecte..... | 27 |
| 3. | Cercetare și analiză..... | 28 |
| 3.1. | Setul de date..... | 28 |
| 3.2. | Metrici de performanță | 29 |
| 3.3. | Alegerea metodelor..... | 30 |
| 3.4. | Soluții existente..... | 31 |
| 4. | Resurse utilizate | 33 |
| 4.1. | Resurse software | 33 |
| 4.1.1. | Python | 33 |
| 4.1.2. | Numpy și Matplotlib | 33 |
| 4.1.3. | OpenCV | 34 |
| 4.1.4. | Tensorflow | 34 |
| 4.1.5. | Keras | 35 |
| 4.2. | Resurse hardware | 35 |
| 4.2.1. | Google Colaboratory..... | 35 |
| 4.2.2. | Nvidia Jetson Nano | 36 |
| 4.3. | Proiecte open-source | 37 |
| 5. | Implementare | 38 |
| 5.1. | Analiza datelor | 38 |
| 5.2. | Prelucrarea datelor | 39 |
| 5.2.1. | Reducerea volumul setului de date | 39 |
| 5.2.2. | Problema imaginilor aproape identice | 39 |
| 5.2.3. | Problema obiectelor suprapuse | 40 |
| 5.2.4. | Dimensionarea cadrelor de ancorare..... | 42 |
| 5.3. | Procesul de antrenare | 43 |
| 5.4. | Probleme întâmpinate | 45 |
| 5.4.1. | Probleme de implementare | 45 |
| 5.4.2. | Rezoluția imaginilor la antrenare..... | 46 |

| | | |
|-----------|---|-----------|
| 5.4.3. | Ajustarea cadrelor de ancorare..... | 46 |
| 5.5. | Asamblarea componentelor | 46 |
| 5.5.1. | Workflow | 47 |
| 5.5.2. | Integrare hardware | 47 |
| 5.5.3. | Integrare software | 48 |
| 6. | Rezultate obținute | 49 |
| 7. | Concluzii | 52 |
| | Bibliografie | 54 |
| | Anexe..... | 58 |
| Anexa A. | Fișier de configurare pentru antrenare | 58 |
| Anexa B. | Arhitecturi | 59 |
| Anexa C. | Sumarul parametrilor | 61 |
| Anexa D. | Graficul procesului de antrenare - YOLOv4 | 62 |
| Anexa E. | Rezultatele modelelor antrenate în Tensorflow | 63 |
| Anexa F. | Detecții pe imagini diverse | 64 |
| Anexa G. | Declarație de onestitate academică | 65 |

GLOSAR

Terminologie

| | |
|-------------------|---|
| batch: | procesul de iterare prin K exemple din setul de date (16, 32, 64 etc.) |
| clasificare: | problemele în care etichetele fac parte dintr-o mulțime discretă |
| convergență: | atunci când un model își atinge obiectivul, se îndreaptă spre un minim global |
| dataset: | set de date sau combustibilul algoritmilor de Machine Learning |
| divergență: | atunci când un model se îndepărtează de obiectiv (punctul de minim) |
| epocă: | procesul de iterare prin toate exemplele setului de date |
| face recognition: | detecția (localizarea) și identificarea facială a unei persoane |
| feature: | trăsătură / caracteristică, o proprietate măsurabilă a datelor |
| fitting: | procesul de învățare / antrenare, prin care se determină modelul matematic |
| hiper-parametru: | un parametru ajustabil care influențează procesul de antrenare |
| inferență: | realizarea de predicții pe date necunoscute |
| label: | etichetă, rezultatul pe care dorim să îl determinăm / prezicem |
| metrică: | metodă de evaluare a performanței unui algoritm |
| model: | relația matematică dintre trăsături (features) și etichete (labels) |
| object detection: | clasificarea și localizarea tuturor obiectelor de interes din imagini statice |
| object tracking: | urmărirea obiectelor pe o secvență video |
| overfitting: | atunci când modelul se specializează pe datele cu care a fost antrenat și nu mai generalizează pe date necunoscute |
| parametru: | sau în engleză „weights”, o valoare numerică ce caracterizează un model; parametrii sunt învățați în timpul procesului de antrenare |
| regresie: | problemele în care etichetele au o valoare continuă |
| fine-tuning: | post-antrenarea unui model cu un alt set de date față de cel inițial |
| underfitting: | atunci când modelul nu are rezultate bune pe datele de la antrenament |

Listă de abrevieri

| | |
|----------|---|
| Adam: | Adaptive moment estimation - algoritm de optimizare |
| AI: | Artificial Intelligence - inteligență artificială |
| API: | Application Programming Interface - interfață prin care se pot accesa funcționalitățile unor componente hardware sau aplicații software |
| ASIC: | Application Specific Integrated Circuit - circuit integrat specializat |
| CNN: | Convolutional Neural Network - rețea neurală convoluțională |
| CUDA: | Compute Unified Device Architecture - API pentru accesarea și controlul GPU-urilor NVIDIA |
| CV: | Computer Vision - vedere artificială |
| CVPR: | Computer Vision and Pattern Recognition - conferință internațională |
| DL: | Deep Learning - învățare automată utilizând rețele neurale adânci |
| FPS: | Frames Per Second - cadre video pe secundă |
| GPU: | Graphical Processing Unit - placă video / grafică |
| HSB/L/V: | Hue Saturation Brightness / Lightness / Value - spațiu de culori |
| IaaS: | Infrastructure as a Service - platformă cloud ce oferă acces la resurse hardware |
| IoT: | Internet of Things - totalitatea sistemelor de calcul ce comunică prin internet |
| IoU: | Intersection over Union - aria intersecției supra aria reuniunii |
| L1: | Norma L1 - distanța Manhattan sau regresie Lasso |
| L2: | Norma L2 - distanța Euclidiană sau regresie Ridge |
| LSTM: | Long Short Term Memory - celulă / neuron folosit în rețelele neurale recurente |
| MAE: | Mean Absolute Error - eroare medie absolută, funcție de cost |
| mAP: | mean Average Precision - metrică folosită pentru măsurare preciziei detectoarelor |
| ML: | Machine Learning - învățare automată |
| MSE: | Mean Squared Error - eroare medie pătratică, funcție de cost |
| NMS: | Non Max Suppression - algoritm de supresie maximală pentru detecția de obiecte |
| PPE: | Personal Protection Equipment - echipament de protecție personală |
| ReLU: | Rectified Linear Unit - funcție de activare |
| RGB: | Red Green Blue - spațiu de culori |
| RMSprop: | Root Mean Square propagation - algoritm de optimizare |
| SGD: | Stochastic Gradient Descent - algoritm de optimizare |
| TPU: | Tensor Processing Unit - ASIC pentru Deep Learning |
| YOLO: | You Only Look Once - algoritm de detecție |

LISTĂ DE FIGURI

| | |
|---|----|
| Figura 2.1 Programare tradițională | 5 |
| Figura 2.2 Programare cu Machine Learning | 5 |
| Figura 2.3 Regresie vs Clasificare | 6 |
| Figura 2.4 Procesul de antrenare..... | 8 |
| Figura 2.5 Convergență vs Divergență | 9 |
| Figura 2.6 Minime locale și puncte șa | 9 |
| Figura 2.7 Neuronul artificial | 10 |
| Figura 2.8 Modele matematice complexe | 11 |
| Figura 2.9 Rețea neurală | 11 |
| Figura 2.10 Operația de convoluție..... | 12 |
| Figura 2.11 Convoluții pe volume | 12 |
| Figura 2.12 Rețea convoluțională | 12 |
| Figura 2.13 Detectarea liniilor verticale și orizontale..... | 13 |
| Figura 2.14 Identificarea trăsăturilor în imagini | 13 |
| Figura 2.15 Operația de pooling | 14 |
| Figura 2.16 Rețea neurală fără activare | 14 |
| Figura 2.17 Funcții de activare pentru ultimul strat..... | 15 |
| Figura 2.18 Funcții de activare pentru straturile ascunse | 15 |
| Figura 2.19 Forward & Backward propagation | 16 |
| Figura 2.20 Normalizare + Optimizare | 17 |
| Figura 2.21 Overfitting vs Underfitting | 18 |
| Figura 2.22 Arhitectura VGG-16..... | 20 |
| Figura 2.23 Bloc funcțional ResNet..... | 20 |
| Figura 2.24 Bloc funcțional Inception | 21 |
| Figura 2.25 Detecția de obiecte | 21 |
| Figura 2.26 Metoda ferestrei glisante | 22 |
| Figura 2.27 YOLO - Principiul de funcționare..... | 23 |
| Figura 2.28 IoU | 24 |
| Figura 2.29 Algoritmul de supresie maximală..... | 24 |
| Figura 2.30 Recunoașterea facială | 26 |
| Figura 4.1 Graf computațional..... | 34 |

| | |
|---|----|
| Figura 5.1 Setul de date virtual | 38 |
| Figura 5.2 Setul de date real | 39 |
| Figura 5.3 Imagini aproape identice | 40 |
| Figura 5.4 Indice de suprapunere..... | 41 |
| Figura 5.5 Eliminarea obiectelor suprapuse..... | 42 |
| Figura 5.6 Algoritmul K-means..... | 42 |
| Figura 5.7 Graficul procesului de antrenare | 45 |
| Figura 5.8 Integrarea hardware | 47 |
| Figura 5.9 Arhitectura software a sistemului..... | 48 |

LISTĂ DE TABELE

| | |
|--|----|
| Tabel 3.1 Performanțele algoritmilor de detecție | 30 |
| Tabel 4.1 Caracteristici plăci video | 36 |
| Tabel 6.1 Rezultate validare pe seturile de date virtuale V350 (S) și V3500 (M)..... | 49 |
| Tabel 6.2 Rezultate validare pe setul de date virtual V350 (S) | 50 |
| Tabel 6.3 Rezultate testare pe setul de date real (R)..... | 50 |
| Tabel 6.4 Rezultate testare pe setul de date real (R) după post-antrenare | 51 |

1. INTRODUCERE

1.1. Context

La nivel mondial [1], anual au loc peste 340 milioane de accidente la locul de muncă și există 160 milioane de victime ale bolilor profesionale. Zilnic se produc, în medie, aproximativ 6300 de accidente mortale, însumând un total de 2.3 milioane de accidente fatale pe an. De asemenea, conform unei statistici ce vizează doar Marea Britanie [2], orice accident de muncă vine și cu un cost, iar acesta poate fi cuantificat în pierderi de până la 15 miliarde de lire sterline pe an și 31.2 milioane de zile lucrătoare pierdute (la un număr de 31.3 milioane de angajați). Costurile se datorează, în mare parte, cheltuielilor medicale și asigurărilor de sănătate, întârzierilor de producție și celor cu refacerea capacității forței de muncă, amenzilor, dar și problemelor juridice derivate.

La nivel european [3], România este „lider” în topul țărilor cu cele mai multe accidente de muncă. O caracteristică a ocupării acestei poziții nu o reprezintă numai resursele insuficiente alocate domeniului de sănătate și siguranță în muncă, cât mai ales formalismului și birocrăției care caracterizează această activitate. În anul 2019 [4], la noi în țară s-au înregistrat 5750 de accidente și 170 de decese. Situația este mult mai îngrijorătoare, întrucât nu toate accidentele care au loc sunt și raportate, în realitate cifrele putând fi chiar și de până la 9 ori mai mari.

Cele mai expuse domenii [5] la accidente de muncă sunt: cel al construcțiilor, cu o pondere de 21%, sectorul industrial cu 17%, urmat de transport și depozitare cu 16.5% și agricultură, silvicultură și pescuit cu 13.2%.

1.2. Problema identificată

Problema adevărată este că, o mare parte dintre aceste accidente ar putea fi evitate, dacă oamenii ar fi mai atenți la respectarea regulilor de sănătate și siguranță în muncă sau, dacă autoritățile ar fi preocupate de evaluarea măsurilor practice luate de către companii.

Un prim pas important în prevenirea sau reducerea impactului accidentelor de muncă este purtarea echipamentului de protecție personală. Acest lucru se aplică atât domeniului construcțiilor, dar și celui industrial. În cazul șantierelor de construcții, echipamentul poate fi constituit din: cască de protecție, vestă reflectorizantă, bocanci cu talpă anti-perforație și bombeu metalic, salopetă, mască de sudură, căști de protecție împotriva zgomotelor, mănuși de protecție și izolatoare pentru curentul electric, ham de siguranță pentru prevenirea căderilor și așa mai departe. Nepurtarea echipamentului de protecție personală expune muncitorii la riscuri și pericole de natură fizică, electrică, chimică sau chiar radiații.

În lucrarea de față o să mă rezum doar la problema identificării a două elemente de siguranță obligatorii din echipamentul de protecție al muncitorilor (cască de protecție și vestă reflectorizantă) și două elemente cu caracter situațional (mască de sudură și căști anti-zgomot), cu mențiunea că identificarea se poate face pentru orice clasă de obiecte, în funcție de cerințele proiectului și de setul de date disponibil.

1.3. Soluția propusă

Soluția pe care o propun pentru rezolvarea acestei probleme este crearea unui sistem de supraveghere și asistență bazat pe vederea artificială, destinat urmăririi respectării normelor de sănătate și siguranță în muncă pe șantierele de construcții. Sistemul este o aplicație software ce poate fi instalată pe o infrastructură deja existentă de camere de supraveghere. Funcționalitățile principale pe care le va îndeplini sistemul sunt identificarea echipamentului de protecție personală al muncitorilor, urmărirea activităților desfășurate de către aceștia și emiterea unor alarme în caz că regulile sunt încălcate sau dacă are loc vreun accident.

1.4. Obiective

Din punct de vedere tehnic, principalele obiective pe care intenționez să le ating, atât pentru lucrarea de diplomă, dar și după aceasta, în planurile de viitor, sunt următoarele:

- identificarea a 7 tipuri de clase de obiecte împărțite în 3 categorii:
 - zona capului: cap fără protecție (Head), cască de protecție (Helmet), mască de sudură (Mask), căști de protecție împotriva zgomotelor (Headset);
 - zona pieptului: piept fără protecție (Chest), vestă reflectorizantă (Vest);
 - persoane (tot corpul acestora - Person);
- caz particular, identificarea culorii căștilor de protecție (roșu, galben, albastru, alb);
- urmărirea eficientă (prin algoritmi de object tracking) a fluxului video;
- recunoașterea și identificarea facială a muncitorilor;
- integrare hardware și crearea unui prototip;
- integrare software într-o aplicație web de monitorizare și platformă de video streaming.

În lucrarea curentă mă voi axa mai mult pe detalierea primului obiectiv, fiind și cel mai important pentru realizarea proiectului și care necesită cel mai mult timp ca volum de muncă.

1.5. Structura lucrării

Închei capitolul de introducere în care am prezentat o viziune de ansamblu asupra lucrării. Următorul capitolul vizează noțiuni teoretice unde explic, într-un mod cât mai intuitiv, fără să pun foarte mult accent pe formulele matematice din spatele algoritmilor, ce este inteligența artificială și conceptele esențiale necesare înțelegerii rezolvării problemei abordate. În capitolul 3 realizez o analiză mai tehnică asupra cerințelor aplicației, soluțiilor deja existente și justific alegerile metodelor folosite. În capitolul 4 avem o prezentare asupra resurselor hardware, software și open-source folosite. Capitolul 5 prezintă procesul de implementare, cu tot cu problemele pe care le-am întâmpinat, rezolvarea acestora și diversele arhitecturi de modele prin care am iterat, dar și arhitectura software a întregului sistem. Ultimele două capitole marchează finalul lucrării printr-o analiză a rezultatelor diverselor modele realizate, evaluarea fiabilității acestora și concluzionează dacă proiectul și-a atins obiectivele și cum se compară față de soluțiile deja existente.

2. NOȚIUNI TEORETICE

2.1. Prezentare generală

„AI is the new electricity.” – Andrew Ng.

Precum curentul electric a revoluționat lumea acum mai bine de 100 de ani în toate domeniile societății, de la iluminat, transport, industrie și comunicații, în prezent, inteligența artificială este o componentă software nelipsită din majoritatea aplicațiilor / tehnologiilor cu care interacționăm. Este folosită în motoarele de căutare, asistenți vocali, recunoaștere facială, prognoză meteo, sisteme de recomandare pentru diverse platforme, reclame personalizate, mașini autonome, instrumente de securitate cibernetică, filtre anti-spam pentru e-mail, jocuri, automatizări, recunoașterea caracterelor, inginerie medicală și este integrată inclusiv în electronicele și electrocasnicele pe care le avem la îndemână. De cele mai multe ori folosim sau beneficiem de aceste componente de inteligență artificială fără să fim măcar conștienți, dar fără de care viața noastră nu ar mai fi la fel de ușoară.

Definiție

Inteligența artificială [6] reprezintă capacitatea unui sistem informatic de a rezolva probleme prin simularea gândirii și comportamentului uman și rațional.

Conform lui Alan Turing [7], un sistem inteligent este caracterizat de faptul că, un om nu poate distinge în timpul unei conversații scrise cu acesta dacă poartă o discuție cu un alt om sau cu un calculator. Definiția se poate extinde la testul complet al lui Turing, în care interogatorul uman poate să testeze și capacitățile perceptuale ale interlocutorului. Astfel, putem clasifica inteligența artificială în 6 mari subdomenii:

- reprezentarea cunoștințelor (cum sunt stocate informațiile)
- deducția automată (a unor reguli și fapte pe baza unor cunoștințe predefinite)
- învățarea automată (recunoașterea unor tipare)
- procesarea limbajului natural (cum este realizată comunicarea)
- vederea artificială (cum sunt percepute imaginile)
- robotică (cum sunt manipulate obiectele)

Momente importante

Deși bazele și fundamentele matematice ale inteligenței artificiale pe care o folosim astăzi au fost puse încă de la mijlocul secolului 20 [8] (1943 - W. S. McCulloch, W. Pitts, primul model matematic al neuronului; 1950 - testul Turing; 1950 - C. Shannon, primul program care joacă șah; 1952 - A. Samuel, primul program care învață singur să joace dame; 1957 - F. Rosenblatt, modelul perceptronului și rețelele neurale), acestea nu au beneficiat de o soluție viabilă de implementare din cauza lipsei puterii de calcul de la vremea respectivă.

A urmat o perioadă de ascensiune a sistemelor expert [8] (anii `70-`80) și a limbajelor de programare logică precum Prolog și LISP. Sistemele expert [6] simulează gândirea rațională, prin deducerea unor reguli de inferență, pe baza unui set de cunoștințe inițiale, definite de către un expert uman. Metoda este utilă atunci când se lucrează cu sisteme exacte și care nu sunt afectate de incertitudini. În cazul unui program care joacă șah, această abordare poate fi folosită întrucât regulile jocului nu se modifică. Problema sistemelor expert este lipsa capacității de adaptare la evenimente neprevăzute, dar și dificultatea dezvoltării de programe complexe. În cazul problemei identificării obiectelor, ar fi imposibil pentru un om să creeze un set de reguli bazate pe logică, astfel încât algoritmul să generalizeze cu acuratețe bună pe imagini diferite ale aceluiași obiect. Aceste limitări au dus la declinul sistemelor expert [8] (anii `90) și apariția domeniului de învățare automată, prin crearea algoritmului de propagare înapoi (D. Rumelhart, G. Hinton, R. Williams), inferențelor probabilistice (J. Pearl) și începutul erei Big Data¹.

Un alt moment important în istoria inteligenței artificiale [8], și mai ales al vederii artificiale, este anul 2012, care marchează începutul folosirii modelului de învățare de tip Deep Learning pentru antrenarea rețelelor neurale adânci. Utilizarea de hardware specializat pentru rezolvarea problemelor de calcul algebric, dezvoltarea tehnicilor de procesare paralelă și a algoritmilor de optimizare au redus considerabil timpul necesar pentru crearea modelelor matematice și au permis dezvoltarea foarte rapidă a acestui domeniu în ultimii 8 ani.

Încă suntem departe de momentul în care vom putea dezvolta un sistem complet [6], de tipul „Strong AI”, care să poată fi comparat cu inteligența umană. Succesul acestui domeniu se datorează capacității sistemelor de a rezolva probleme specializate, „Weak AI”, cu o acuratețe foarte bună și la o viteză mult mai mare decât cea necesară oamenilor.

2.2. Machine Learning

„Domeniul care studiază modul în care calculatoarele pot fi înzestrate cu abilitatea de a învăța, fără ca acestea să fie programate în mod explicit.” – Arthur Samuel.

Învățarea automată [6] este utilă atunci când ne este „imposibil” să definim un set de reguli care caracterizează un sistem, dar dispunem de suficiente date necesare pentru determinarea unui tipar.

2.2.1. Intuiție

Abordare intuitivă

Din punct de vedere intuitiv, învățarea automată nu este foarte diferită de modul în care oamenii învață să perceapă lucrurile în primii ani de viață. Pentru un bebeluș este foarte important ca părinții să comunice cu acesta, astfel el reușește să învețe. Asemănător și în cazul

¹ Big Data: termen folosit pentru referirea unui volum foarte mare de date. Din același context mai avem și „Data Mining” care se referă la tehnicile eficiente de prelucrare a datelor.

sistemelor de calcul, dacă îi arătăm unui calculator 10000 de poze cu câini, 10000 de poze cu pisici și îi spunem ce reprezintă fiecare imagine, când o „să vadă” o imagine pe care nu a mai întâlnit-o până atunci, acesta va fi capabil să distingă o serie de trăsături specifice obiectului respectiv și să decidă cu o acuratețe cât mai mare dacă în imagine se află un câine sau o pisică.

Abordare matematică

Din punct de vedere matematic, putem să comparăm acest proces de predicție cu o funcție matematică $f(x) = y$, unde x este argumentul de intrare, care în cazul nostru pot să fie valorile pixelilor dintr-o imagine, iar y este rezultatul sau parametrul de ieșire, care poate să fie 1 dacă în imagine se află o pisică, 2 dacă este un câine și 0 altfel. Obiectivul domeniului de învățare automată este determinarea acestei funcții f , care asociază cât mai bine o relație matematică între x și y , în funcție de setul de date utilizat.

Abordare programatică

În programarea clasică, în funcție de datele de intrare și a unui algoritm scris de către programator, calculatorul procesează informația și produce un rezultat (Figura 2.1).

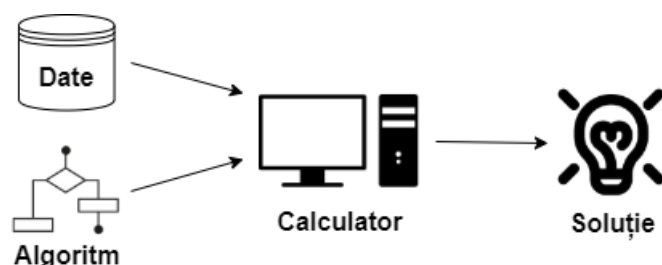


Figura 2.1 Programare tradițională

În Machine Learning, rolul programatorului se schimbă. Acesta nu mai scrie în mod explicit programul care rezolvă problema, ci se folosește de niște algoritmi de învățare care, pe baza unui set de date de intrare și a rezultatelor la care trebuie să se ajungă, produc un model matematic (sau algoritmul) pentru cerința respectivă (Figura 2.2).

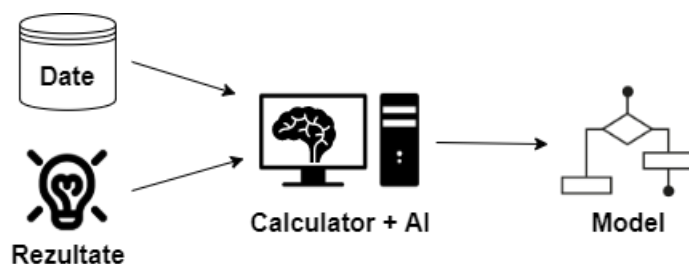


Figura 2.2 Programare cu Machine Learning

Paradigme de învățare [6]

- Învățare Supervizată (Supervised Learning), atunci când avem toate datele etichetate. Este cea mai comună ramură și cea care oferă cele mai bune rezultate. Poate să fie de două tipuri: regresie, atunci când eticheta are valoare continuă sau, clasificare, atunci când etichetele fac parte dintr-o mulțime discretă.
- Învățare Nesupervizată (Unsupervised Learning), atunci când nu avem datele etichetate. Este folosită pentru determinarea de structuri în setul de date și gruparea acestora (clustering), identificarea de tipare, detecția de anomalii.
- Învățare Semi-Supervizată, este folosită atunci când avem un număr mic de date etichetate și un număr mare de date neetichetate, cele din urmă putând fi folosite pentru îmbunătățirea acurateții modelelor antrenate în manieră supervizată.
- Învățare pe bază de recompense (Reinforcement Learning), folosește un set de recompense și penalități prin care algoritmi învață să ia decizii.
- Învățare prin transfer de cunoștințe (Transfer Learning), pornind de la un model antrenat pentru rezolvarea unei anumite probleme, exemplu localizarea fețelor în imagini, folosim o parte din cunoștințele acestuia pentru crearea unui nou model care rezolvă o problemă diferită, exemplu identificare și recunoașterea facială a persoanei. Această metodă o să vedem că este foarte utilizată în vederea artificială, deoarece modelele dezvoltate, indiferent de scopul pentru care au fost antrenate, au în comun o serie de proprietăți ce pot fi împărtășite.

2.2.2. Regresie vs Clasificare

În Figura 2.3 avem reprezentarea grafică a celor două tipuri de probleme. În cazul regresiei liniare, linia roșie reprezintă dreapta care aproximează cel mai bine valorile din setul de date. Pentru o valoare necunoscută x , punctul care corespunde dreptei $y = f(x)$ reprezintă cea mai „apropiată” valoare de datele problemei. Pentru problema clasificării, linia roșie reprezintă doar o graniță de separare între punctele albastre și cele portocalii. Dacă un punct necunoscut se află sub dreaptă, atunci este foarte probabil ca acesta să fie portocaliu și invers.

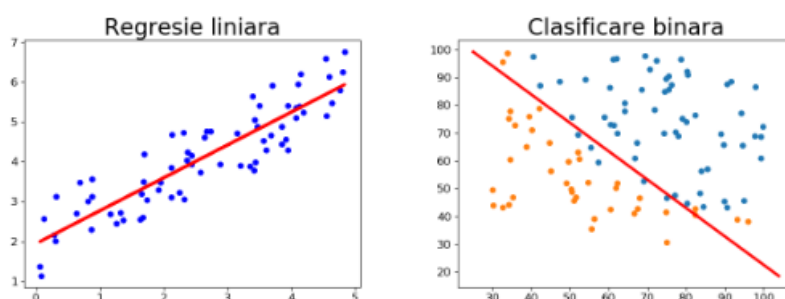


Figura 2.3 Regresie vs Clasificare

În prima imagine din figură avem reprezentarea grafică a unui set de date cu o singură caracteristică (feature), prin urmare ecuația dreptei o să fie de forma $y = \theta_0 + \theta_1 \cdot x$, unde θ_0 și θ_1 sunt parametrii ce definesc sistemul și sunt determinați în cadrul procesului de antrenare.

În a doua imagine avem un set de date cu două caracteristici, prin urmare graficul este doar o reprezentare bidimensională a proprietăților x_1 și x_2 ale sistemului. Pentru problema clasificării nu este suficient să avem doar o reprezentare liniară ca în (1), ci avem nevoie și de o „activare” care să atribuie etichetele 0 (albastru) și 1 (portocaliu), în funcție de valoarea părții liniare, z . Funcția folosită în (2) se mai numește și funcție de activare logistică [9] (Sigmoid).

$$z = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 \quad (1)$$

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (2)$$

De remarcat faptul că, atunci când valoarea lui z este negativă, y tinde spre 0, iar când valoarea lui z este pozitivă, y tinde spre 1. Când z este 0, valoarea lui y este exact 0.5 și poate fi rotunjită fie la 1, fie la 0. La fel ca în cazul regresiei, θ_i reprezintă parametrii ce definesc modelul și sunt învățați în cadrul procesului de antrenare.

2.2.3. Funcția obiectiv

Problemele de inteligență artificială se rezumă, în majoritatea cazurilor, la niște probleme de minimizare a unor expresii matematice numite funcții obiectiv. Vom nota cu N numărul de elemente din setul de date, cu $\hat{y} = h_\theta(x)$ predicția sau ipoteza (echivalentul lui $y = f(x)$ din sub-capitolul anterior), cu L funcția de pierdere dintre etichetele și predicțiile unui sistem, iar cu J funcția de cost (sau obiectiv), care este media între sumele pierderilor din setul de date. În (3) și (4) avem funcția de eroare medie pătratică [9] (MSE – Mean Squared Error), specifică problemelor de regresie.

$$L(\hat{y}, y) = \frac{1}{2} \cdot (\hat{y} - y)^2 = \frac{1}{2} \cdot (h_\theta(x) - y)^2 \quad (3)$$

$$J(\theta) = \frac{1}{N} \cdot \sum_{i=1}^N L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2N} \cdot \sum_{i=1}^N (h_\theta(x^{(i)}) - y^{(i)})^2 \quad (4)$$

În problemele de clasificare nu se folosește MSE, deoarece introduce probleme de convergență de tipul minime locale, motiv pentru care este preferată o funcție de cost bazată pe cross-entropie [9], reprezentată în ecuațiile (5) și (6).

$$L(\hat{y}, y) = -\left(y \cdot \log(h_\theta(x)) + (1 - y) \cdot \log(1 - h_\theta(x))\right) \quad (5)$$

$$J(\theta) = -\frac{1}{N} \cdot \sum_{i=1}^N \left(y^{(i)} \cdot \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))\right) \quad (6)$$

De observat că, atunci când valoarea etichetei y este 0, primul termen din ecuații se anulează, iar când eticheta are valoarea 1, al doilea termen se anulează. Semnul „-” este folosit la însumare, deoarece funcția h_θ produce un rezultat între 0 și 1, iar valoarea logaritmului pe acel interval este negativă, prin urmare semnele se anulează și costul final va fi pozitiv.

2.2.4. Procesul de antrenare

Antrenarea este un proces iterativ, în care se încearcă, la fiecare pas, obținerea unui rezultat mai bun decât cel precedent, după cum se observă și în Figura 2.4.

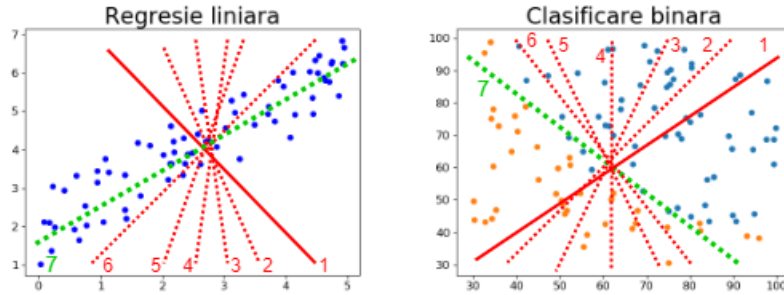


Figura 2.4 Procesul de antrenare

Algoritmul utilizat pentru minimizarea funcțiilor obiectiv (atât pentru regresie, cât și pentru clasificare) se numește Gradient Descent [9] și este definit în felul următor:

- 1) Se inițializează toți parametrii θ_i ai modelului cu 0 sau valori aleatoare;
- 2) Pentru $k = 1$ până la numărul total de iterații:
 - a. Se calculează funcția de cost J a sistemului;
 - b. Se actualizează toți parametrii sistemului conform formulei (7);

$$\theta_j = \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta) = \theta_j - \frac{\alpha}{N} \cdot \sum_{i=1}^N (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot (h_{\theta}(x^{(i)}))' \quad (7)$$

Pentru algoritmul definit mai sus putem face următoarele observații:

- Se preferă inițializarea cu numere aleatoare pentru termenii ce însoțesc caracteristicile datelor x_i , deoarece inițializarea cu 0 produce simetrie și face ca toți parametrii să aibă valori identice în timpul antrenării. Pentru termenul liber θ_0 , putem inițializa cu 0;
- Numărul de iterații poate fi exprimat în număr de epoci, dacă calculăm funcția de cost J pe întreg setul de date, caz în care algoritmul se numește Batch Gradient Descent sau numărul de iterații poate fi exprimat în număr de batch-uri, dacă calculăm funcția de cost pe fragmente din setul de date, caz în care algoritmul se numește mini-Batch Gradient Descent. Dacă mini-batch-ul este format dintr-un singur exemplu, atunci algoritmul se numește Stochastic Gradient Descent (SGD);
- Pentru regresia liniară și clasificarea binară, funcțiile de cost sunt (4) și, respectiv (6), dar putem avea și funcții mai complexe, în funcție de modelele pe care le folosim;
- În formula (7), α este un hiper-parametru ajustabil ce reprezintă pasul de învățare (learning rate), derivatele costului $J(\theta)$ sunt identice pentru cele două tipuri de probleme (regresie și clasificare), iar parametrii sistemului trebuie actualizați toți în același timp.

2.2.5. Probleme de convergență

A converge, în contextul inteligenței artificiale, este proprietatea unui model matematic de a-și minimiza funcția obiectiv (de cost) până la un optim global. În timpul procesului de antrenare pot să apară mai multe tipuri de probleme legate de ritmul cu care converge sistemul.

În Figura 2.5 avem reprezentarea costului J în funcție de un singur parametru θ . Pentru funcții cu peste 2 parametri, graficul ar fi imposibil de reprezentat, deoarece ar fi de tipul $(N + 1)$ -dimensional, unde N este numărul de parametri învățabili din sistem, iar $(N + 1)$ este numărul de dimensiuni. Din punct de vedere geometric, derivata unei funcții reprezintă panta graficului acesteia, adică cât de înclinat / abrupt este graficul într-un anumit punct. În ecuația (7), dacă pasul de învățare este prea mare, atunci există riscul ca modelul să sară peste minimul funcției sau chiar riscul ca modelul să fie divergent, cum se întâmplă în al doilea grafic din figură. Pe de altă parte, dacă pasul de învățare este prea mic, atunci ritmul de convergență o să fie unul scăzut, iar procesul de antrenare ar putea să dureze foarte mult [9].

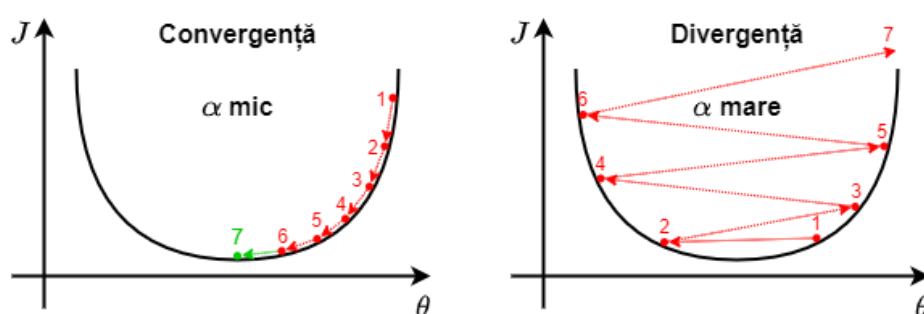


Figura 2.5 Convergență vs Divergență

Dacă folosim funcții obiectiv care nu sunt convexe (nu au formă geometrică de castron care ține apa - Figura 2.5), există riscul ca sistemul să nu mai atingă minimul global, ci să convergă către un punct de minim local (Figura 2.6). Aceasta nu este o problemă foarte comună pentru sistemele cu mulți parametri (de exemplu rețelele neurale), întrucât ar trebui ca toate derivatele parțiale să fi ajuns în mod simultan la un punct de minim. O problemă mai comună este ca algoritmul de învățare să rămână pentru mult timp blocat într-un „punct șa”, unde derivata este aproape 0, deci pasul cu care se face actualizarea parametrilor este mic [9].

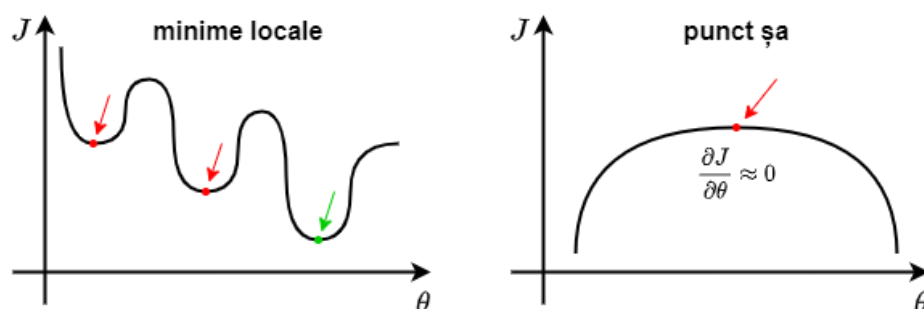


Figura 2.6 Minime locale și puncte șa

2.3. Deep Learning

Deep Learning este o paradigmă de învățare, extensie a Machine Learning-ului, ce folosește ca instrument de lucru rețelele neurale pentru crearea de modele matematice complexe. Rețelele neurale se aseamănă și sunt inspirate din modul de funcționare al creierului uman. Acestea urmează paradigma conexiionistă [6], în care cunoștințele reprezintă niște stări ce sunt activate de neuroni, atunci când sunt îndeplinite anumite condiții. Cuvântul „deep” din titlu se referă la faptul că aceste rețele sunt formate din mai multe straturi suprapuse, numărul straturilor, în general, crește acuratețea, dar și costul computațional. Deep Learning-ul reprezintă „trend”-ul sau tehnologia de ultimă generație a ultimilor 8 ani în domeniul inteligenței artificiale. Algoritmii și ideile anterioare din Machine Learning fie nu mai sunt folosite, fie au fost adaptate și folosesc acum rețele neurale pentru implementare.

2.3.1. Neuronul

Neuronul [10] este unitatea atomică fundamentală a rețelilor neurale artificiale. Din punct de vedere funcțional, neuronul se aseamănă foarte mult cu problema clasificării din capitolul anterior. Acesta este compus dintr-o ecuație de activare liniară z , asemănătoare cu o problemă de regresie și o funcție de activare f_a , care în cazul clasificării binare era funcția Sigmoid (σ). În Figura 2.7, x_1 și x_2 sunt proprietățile sau caracteristicile setului de date (features), w_1 și w_2 se mai numesc și ponderi (weights), iar împreună cu termenul liber b reprezintă parametrii învățabili ai sistemului.

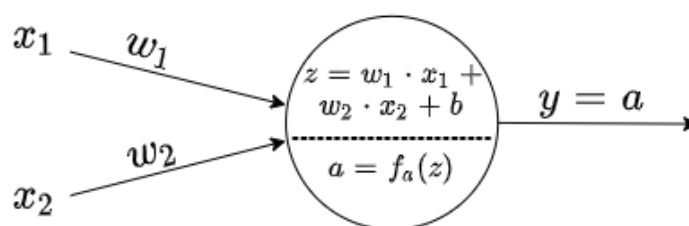


Figura 2.7 Neuronul artificial

Ideea aceasta de „activare” a unui neuron poate fi comparată cu modul în care creierul uman reacționează la anumiți stimuli din exterior sau interior. De exemplu, atunci când ne uităm la o serie de fotografii, informația vizuală din acestea este procesată și transferată prin rețeaua de neuroni a creierului nostru. În momentul în care recunoaștem o persoană sau un anumit peisaj din fotografii, cu alte cuvinte identificăm un tipar (pattern), anumiți neuroni din cortex se activează și ne semnalează faptul că am mai întâlnit acel context. Dacă este o amintire foarte importantă pentru noi, atunci acei neuroni vor fi puternic activați. La fel și în cazul neuronilor artificiali, aceștia sunt activați în funcție de valorile parametrilor w_i , cu cât caracteristicile x sunt mai proeminente sau neuronii au mai multă experiență, cu atât rezultatele activărilor lor vor fi mai puternice.

2.3.2. Tipuri de straturi

2.3.2.1. Rețele neurale dense

În capitolul de Machine Learning am folosit numai ecuații ale unor linii drepte pentru a separa clasele din setul de date. De cele mai multe ori, acestea nu sunt suficiente pentru a obține o acuratețe bună pentru setul nostru de date. Rețelele neurale ne ajută să creăm modele matematice complexe, precum cele din Figura 2.8.

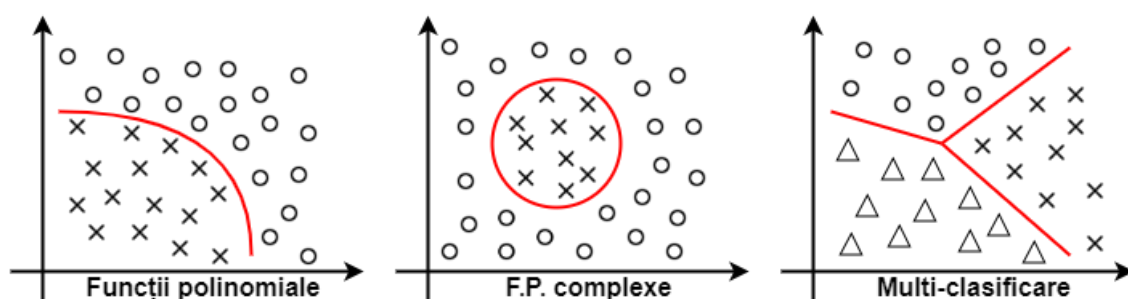


Figura 2.8 Modele matematice complexe

Rețelele neurale [10] sunt formate din mai multe unități funcționale - neuroni, dispuși sub orice configurație și în orice număr. Se numesc straturi dense deoarece fiecare neuron din fiecare strat este conectat cu toți neuronii din stratul precedent și următor. În Figura 2.9 avem o rețea neurală cu 4 intrări, 5 straturi (dintre care 3 sunt straturi ascunse) și o ieșire. Numărul arhitecturilor posibile de rețele neurale este limitat doar de imaginația noastră.

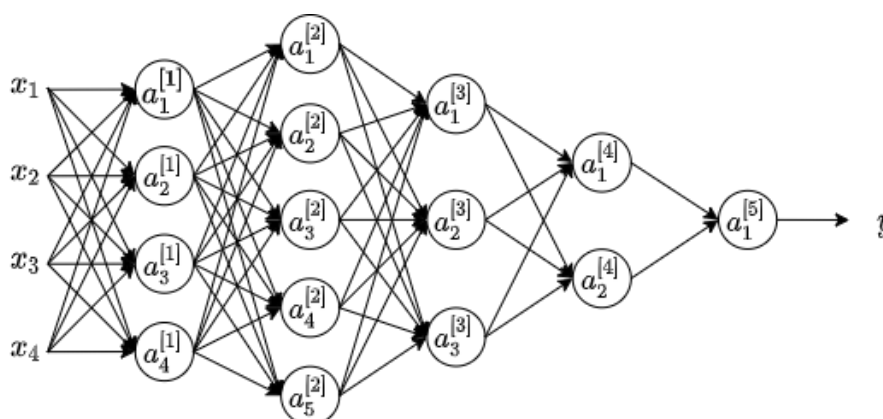


Figura 2.9 Rețea neurală

2.3.2.2. Rețele neurale convoluționale

Din punct de vedere matematic, convoluția reprezintă o serie de înmulțiri și adunări între elementele a doi vectori N-dimensional. În Figura 2.10, prima matrice reprezintă datele de intrare, iar a doua matrice se numește filtru. De asemenea, convoluția se poate realiza și pe volume și cu orice număr de filtre diferite ca în Figura 2.11.

| | | | | | |
|---|---|---|---|---|---|
| 3 | 2 | 4 | 1 | 0 | 7 |
| 6 | 3 | 6 | 2 | 3 | 2 |
| 1 | 2 | 2 | 4 | 1 | 7 |
| 4 | 2 | 0 | 5 | 7 | 1 |
| 1 | 0 | 7 | 2 | 3 | 2 |
| 3 | 0 | 2 | 4 | 0 | 7 |

 6×6

| | | |
|---|---|----|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

 3×3
 $*$

| | | | |
|----|----|----|----|
| -2 | 0 | 8 | -9 |
| 3 | -4 | -3 | 1 |
| -3 | -7 | -2 | 1 |
| -1 | -9 | -1 | 1 |

 4×4

Figura 2.10 Operația de convoluție

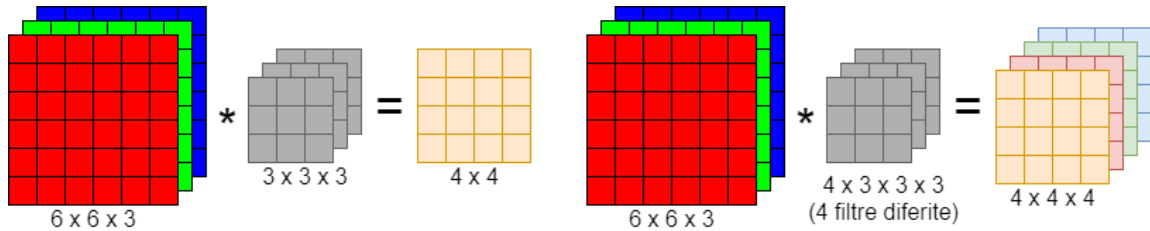


Figura 2.11 Convoluții pe volume

Un strat convoluțional [10] este definit de următoarele proprietăți:

- dimensiunea filtrului sau kernel size (k);
- pasul de deplasare sau strides (s);
- numărul de filtre (f), cu câte filtre diferite este convoluționat volumul într-un singur strat;
- padding (p) pentru umplerea spațiilor laterale din jurul datelor de intrare. Dacă (p) este „same”, atunci adăugăm 0-uri, astfel încât volumul rezultat să aibă aceeași dimensiune ca cel inițial. Dacă avem padding „valid”, atunci adăugăm cât padding este necesar ca să obținem o convoluție validă pentru pasul (s);
- la fel ca în cazul rețelelor dense, straturile convoluționale pot fi activate.

În figurile de mai sus avem exemple de convoluții cu dimensiunea filtrului $k=(3 \times 3)$, pasul de deplasare $s=(1 \times 1)$, padding valid și un număr de $f=1$, respectiv $f=4$ filtre pentru ultima imagine.

O rețea convoluțională (CNN) este o funcție matematică ce preia ca argument de intrare un volum, care în majoritatea cazurilor este o imagine în format RGB și, printr-o serie de convoluții, volumul este redus, rezultând în final predicția noastră. În Figura 2.12 avem un exemplu de rețea convoluțională, în care sunt ilustrate variații ale proprietăților k, s, f și p.

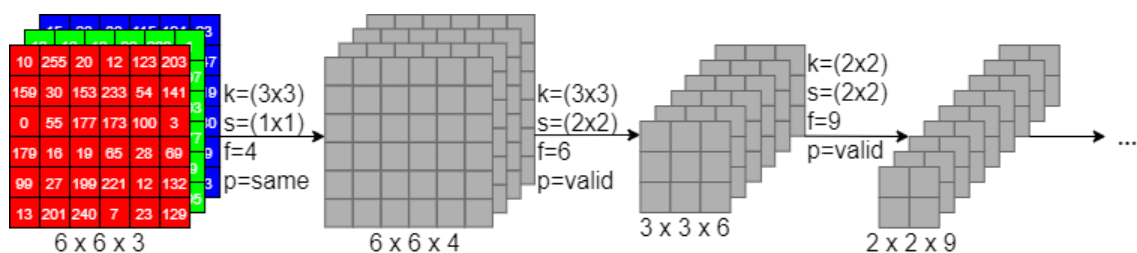


Figura 2.12 Rețea convoluțională

Detectarea de trăsături cu ajutorul filtrelor

O proprietate foarte importantă a filtrelor [10] este capacitatea acestora de a detecta trăsături (sau pattern-uri) precum linii, curbe, texturi sau forme. De asemenea, un filtru specializat pe o anumită sarcină va fi aplicat pe toate regiunile dintr-o imagine și va detecta toate acele zone care respectă acea proprietate. În Figura 2.13 avem un exemplu cu 2 filtre specializate pentru detectarea liniilor verticale și orizontale. Un alt operator interesant este filtrul Sobel (folosește secvențele $[1\ 2\ 1]$, $[0\ 0\ 0]$, $[-1\ -2\ -1]$, dispuse orizontal sau vertical), care are proprietatea de a detecta muchiile indiferent de orientarea acestora.

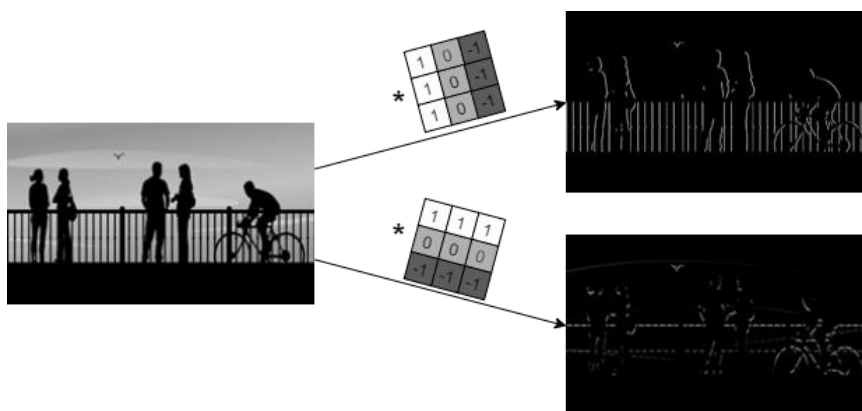


Figura 2.13 Detectarea liniilor verticale și orizontale²

În rețelele convoluționale adânci [10], primele straturi sunt specializate pe detectarea trăsăturilor simple precum linii, curbe și muchii, iar pe măsură ce avansăm mai adânc în rețea, straturile pot să detecteze și trăsături mai complexe de tipul texturi, forme, diverse părți ale corpului și chiar obiecte întregi, ca în Figura 2.14.



Figura 2.14 Identificarea trăsăturilor în imagini²

În realitate, filtrele nu sunt proiectate de către oameni. Valorile acestora sunt înlocuite cu parametri învățați w_i , ce sunt calculați în timpul procesului de antrenare și sunt specifice în funcție de setul de date. Cu toate acestea, trăsăturile fundamentale detectate în primele straturi ale rețelelor convoluționale sunt comune, indiferent de tipul aplicației și pot fi partajate.

² Sursă imagine: <https://www.coursera.org/lecture/convolutional-neural-networks/edge-detection-example-4TroD>

Convoluțiile reprezintă stratul fundamental în toate aplicațiile vederii artificiale. Avantajele acestora față de straturile dense de neuroni sunt următoarele [10]:

- numărul mai mic de parametri ce trebuie învățați pentru fiecare strat. În cazul rețelelor dense, pentru o imagine de dimensiune $H \times W \times C$ (înălțime \times lățime \times adâncime), ar trebui să avem un număr de parametri egal cu $H \cdot W \cdot C \cdot K$, unde K este numărul de neuroni din următorul strat. În cazul convoluțiilor, se lucrează cu filtre de dimensiuni mici (3×3 , 5×5), prin urmare numărul de parametri este redus la $h \times w \times C \times F$, unde F este numărul de filtre specifice fiecărui strat, iar h și w au valori mici (3 sau 5).
- aceleași filtre sunt aplicate pe regiuni diferite din imagine; acest lucru va fi util mai departe în capitolul în care prezint detecția de obiecte.

2.3.2.3. Straturi de pooling

Straturile de pooling [10] sunt folosite pentru a selecta acele trăsături (valori ale parametrilor) care sunt cele mai proeminente pentru anumite regiuni dintr-o imagine. La fel ca în cazul convoluțiilor, acestea împart proprietățile de dimensiune a kernel-ului, pasul de deplasare și pot fi aplicate pe volume. În Figura 2.15 avem un exemplu cu două operații de pooling pentru selectarea maximului și calcularea mediei dintr-o regiune.

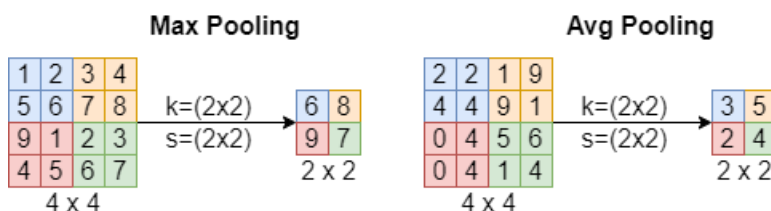


Figura 2.15 Operația de pooling

2.3.3. Funcții de activare

Funcțiile de activare [10] sunt acele componente care introduc neliniaritate în sistem. Dacă nu ar exista o componentă neliniară, atunci toate rețelele neurale, indiferent de numărul de straturi și neuroni, ar putea fi reduse la un sistem cu un singur strat și K neuroni, în funcție de dimensiunea vectorului de ieșire (asemănător se întâmplă și cu sistemele criptografice dacă nu au componente neliniare). Pentru a înțelege de ce se întâmplă acest lucru, putem să urmărim Figura 2.16, în care este reprezentată o rețea neurală fără funcții de activare.

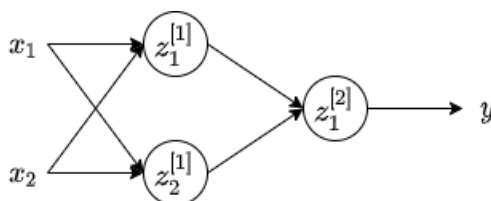


Figura 2.16 Rețea neurală fără activare

Formulele matematice pentru cele 3 valori ale lui z sunt următoarele:

$$z_1^{[1]} = w_{11}^{[1]} \cdot x_1 + w_{12}^{[1]} \cdot x_2 + b_1^{[1]} \quad (8)$$

$$z_2^{[1]} = w_{21}^{[1]} \cdot x_1 + w_{22}^{[1]} \cdot x_2 + b_2^{[1]} \quad (9)$$

$$z_1^{[2]} = w_{11}^{[2]} \cdot z_1^{[1]} + w_{12}^{[2]} \cdot z_2^{[1]} + b_1^{[2]} \quad (10)$$

Înlocuind în relația (10) cu expresiile din (8) și (9) obținem:

$$\begin{aligned} z_1^{[2]} &= w_{11}^{[2]} \cdot (w_{11}^{[1]} \cdot x_1 + w_{12}^{[1]} \cdot x_2 + b_1^{[1]}) + w_{12}^{[2]} \cdot (w_{21}^{[1]} \cdot x_1 + w_{22}^{[1]} \cdot x_2 + b_2^{[1]}) + b_1^{[2]} \\ &= (w_{11}^{[1]} \cdot w_{11}^{[2]} + w_{21}^{[1]} \cdot w_{12}^{[2]}) \cdot x_1 + (w_{12}^{[1]} \cdot w_{11}^{[2]} + w_{22}^{[1]} \cdot w_{12}^{[2]}) \cdot x_2 + w_{11}^{[2]} \cdot b_1^{[1]} + w_{12}^{[2]} \cdot b_2^{[1]} + b_1^{[2]} \end{aligned} \quad (11)$$

Prin urmare, relația (11) poate fi echivalată cu (12), care este formula pentru o rețea cu un singur neuron, identică cu cea din Figura 2.7.

$$y = w'_1 \cdot x_1 + w'_2 \cdot x_2 + b' \quad (12)$$

În Figura 2.17 avem câteva exemple de funcții de activare folosite în general pentru ultimul strat al rețelelor neurale, iar în Figura 2.18 avem exemple de funcții de activare folosite pentru straturile intermediare sau ascunse.

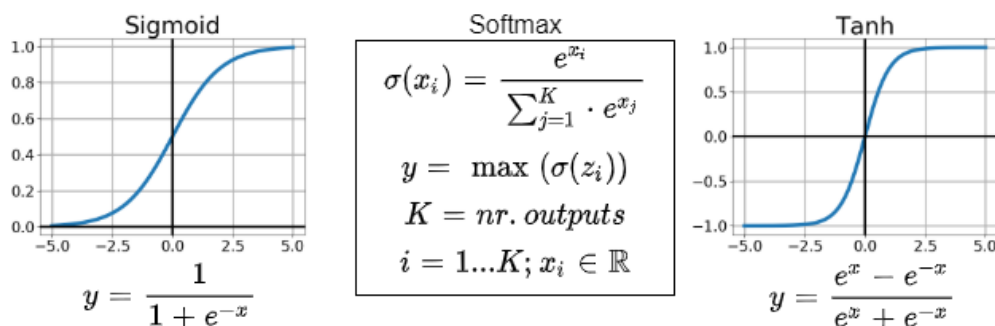


Figura 2.17 Funcții de activare pentru ultimul strat

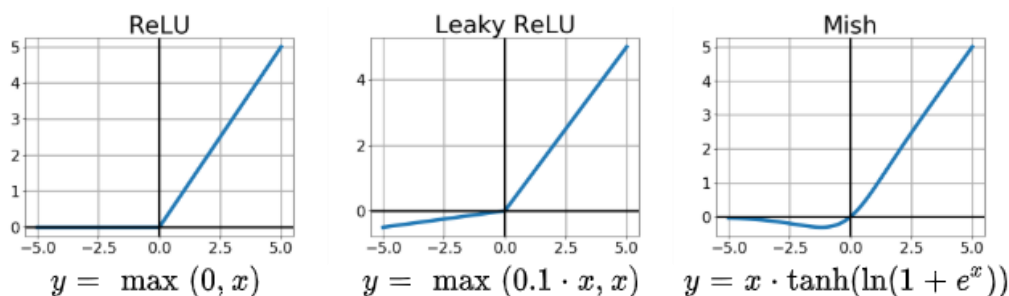


Figura 2.18 Funcții de activare pentru straturile ascunse

Sigmoid este folosită în general pentru clasificare binară. Softmax este o funcție folosită pentru clasificarea între mai multe clase de obiecte, operatorul σ calculează probabilitatea ca obiectul să reprezintă clasa i și normalizează acea valoare în intervalul $0 - 1$, astfel suma tuturor

probabilităților pentru fiecare clasă o să fie 1, iar funcția $\max()$ selectează apoi clasa cu probabilitatea cea mai mare. Motivul pentru care aceste funcții nu se (mai) folosesc pentru toate straturile rețelelor neurale este că au derivata foarte apropiată de 0 pentru valori îndepărtate de origine, ceea ce îngreunează procesul de antrenare și convergența algoritmului.

Cea mai folosită funcție de activare pentru straturile intermediare este ReLU. Aceasta are derivata 1 pentru valori pozitive și, deși are derivata 0 pentru valori negative, acest lucru nu afectează procesul de antrenare, deoarece chiar dacă un neuron nu este activat într-un anumit strat, având un număr foarte mare de neuroni, cel puțin jumătate dintre aceștia vor fi activați și se vor putea calcula gradientii sistemului [10].

2.3.4. Forward & Backward propagation

Forward și Backward propagation [10] sunt sinonimele pentru termenii de inferență, respectiv antrenare pentru Deep Learning. În Figura 2.19 avem o reprezentare sumară a unei rețele neurale cu 3 straturi. Fiecare termen din figură este un vector sau o matrice. Colorat cu albastru avem procesul de propagare înainte, unde în fiecare strat se calculează componenta liniară Z și partea de activare A , cu funcția de activare specifică. În final, rezultă predicția noastră \hat{Y} pentru datele de intrare X . Dacă ne oprim în acest punct, avem echivalentul inferenței, adică predicția pe date necunoscute. Pentru a antrena rețeaua și a actualiza parametrii sistemului, se calculează mai departe funcția de eroare între predicția curentă și etichetele Y , iar apoi, pentru fiecare strat din rețea se calculează, în sens invers, gradientii sistemului. Obiectivul procesului de propagare înapoi este determinarea derivatelor parțiale dW și db (în Figura 2.19 avem derivate totale deoarece lucrăm cu vectori și matrici), care vor fi introduse apoi într-un algoritm de optimizare precum SGD pentru actualizarea parametrilor. De observat în figură că, valorile intermediare ale lui Z și A trebuie păstrate în memorie (cached), pentru a fi folosite mai departe. dZ este derivata în funcție de Z a activării stratului curent pentru fiecare termen din reprezentarea vectorială a acestuia. Valoarea aceasta intermediară este folosită pentru calcularea lui dW și db , care reprezintă obiectivul procesului de antrenare și calcularea lui dA , ce poate fi folosită mai departe ca funcție de eroare intermediară în calcularea gradientilor.

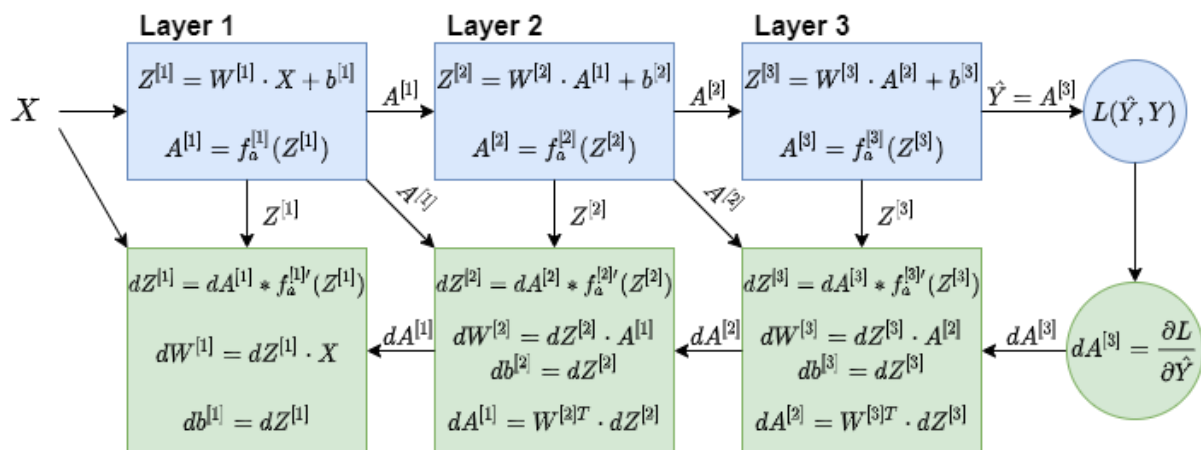


Figura 2.19 Forward & Backward propagation

2.3.5. Tehnici de creștere a performanței

În secțiunea 2.2.5 am vorbit despre convergență și câteva tipuri de probleme care pot să apară în timpul procesului de antrenare. În această secțiune voi prezenta câteva soluții menite să crească viteza și modul în care converg modelele spre minimul global.

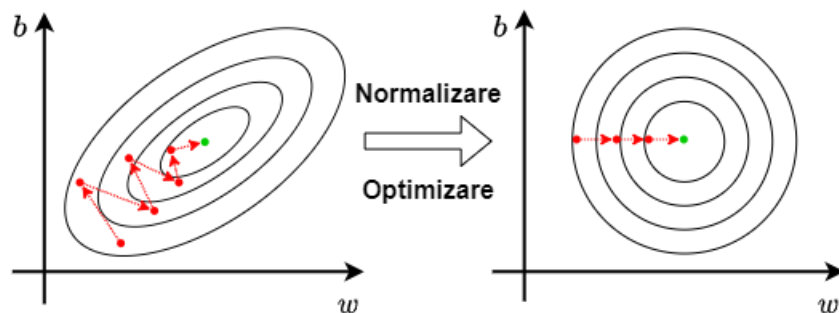


Figura 2.20 Normalizare + Optimizare

Normalizarea datelor [10] este o tehnică folosită în creșterea performanței, ce poate să presupună fie scalarea valorilor în intervalul 0-1, fie valorile sunt ajustate să fie centrate pe 0 după o distribuție gaussiană. Această procedură este utilă deoarece într-un set de date este posibil să avem diferențe de câteva ordine de mărime (10^2 , 10^3 , etc.) între valorile proprietăților x_i , diferențe ce se vor propaga și asupra parametrilor modelului.

În Figura 2.20 avem o reprezentare pe nivele³ [9] a costului J în funcție de parametrii w și b ai sistemului. Dacă avem valori nebalansate ale parametrilor, atunci funcția obiectiv va arăta asemănător cu primul grafic, unde se poate observa că distanța de la minimul acesteia la punctele de pe contur este disproporționată. Aceasta este o problemă, deoarece algoritmul de învățare va fi nevoit să facă mai mulți pași în timpul antrenării până să atingă punctul de minim. În contextul vederii artificiale, cea mai simplă formă de normalizare a datelor este împărțirea tuturor valorilor pixelilor la 255, valorile fiind din intervalul 0-255, nu este nevoie să facem o scalare de tipul Min-Max (13). În cadrul rețelelor neurale adânci [10], cu cât avansăm mai mult în rețea, cu atât valorile parametrilor s-ar putea să scadă sau să crească exponențial din cauza multitudinii de operații algebrice ce au loc. Prin urmare, se poate implementa normalizarea datelor și la nivel de straturi (Batch Normalization), pentru a preveni această problemă. În (14) avem formula utilizată în cadrul straturilor pentru normalizarea trăsăturilor, unde μ este media și σ este deviația standard.

$$\bar{x} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (13)$$

$$\bar{x} = \frac{x - \mu}{\sigma}; \quad \mu = \frac{1}{N} \cdot \sum_{i=1}^N x_i; \quad \sigma = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^N (x_i - \mu)^2} \quad (14)$$

³ reprezentare pe nivele (contour plot): un grafic 3D al unei funcții reprezentat în două dimensiuni

Algoritmii de optimizare nu sunt altceva decât niște îmbunătățiri aduse algoritmului Gradient Descent, menite să rezolve o parte din problemele care pot să apară în timpul antrenării, precum cea a divergenței, atunci când valoarea derivatei este foarte mare și există riscul ca modelul să sară peste punctul de minim sau dacă modelul este blocat într-un punct de sa și valorile derivatelor sunt foarte apropiate de 0.

Algoritmul pe care l-am folosit în această lucrare se numește Adam [11] și este unul dintre cei mai performanți și populari algoritmi de optimizare din prezent. Adam este o metodă ce combină tehnicile folosite în algoritmii SGD+momentum și RMSprop. Nu voi mai prezenta formula deoarece implică noțiuni avansate de matematică. O explicație detaliată poate fi găsită în lucrarea referențiată anterior sau în acest articol [12]. De reținut este că partea de momentum ajută SGD să accelereze în direcția potrivită către minimul funcției și reduce oscilațiile care apar ca în primul grafic din Figura 2.20, asemănător cu „o bilă care coboară un deal”. Partea inspirată din RMSprop ajută această bilă să frâneze atunci când panta se termină și începe iar urcușul. Adam este un algoritm recurent ce depinde de stările din trecut ale gradientilor pentru a-și da seama când trebuie să accelereze și să decelereze. Pentru a ține istoricul gradientilor, algoritmul înmulțește la fiecare iterație valoarea precedentă cu un coeficient de amortizare și le adună la valoarea curentă, astfel valorile din trecutul îndepărtat nu vor mai fi luate în considerare, iar cele recente vor avea impact mare asupra rezultatului.

2.3.6. Overfitting vs Underfitting

Termenii din titlu reprezintă două fenomene cu care orice practicant al inteligenței artificiale are de-a face, indiferent de tipul problemei pe care încearcă să o rezolve.

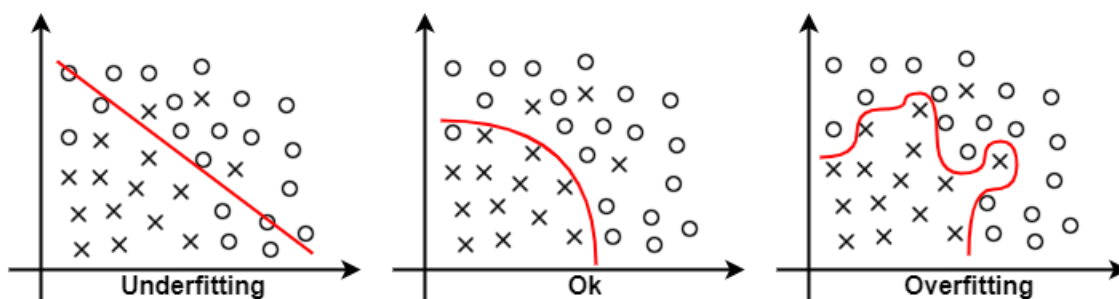


Figura 2.21 Overfitting vs Underfitting

Underfitting [10] înseamnă că modelul are acuratețe slabă pe datele folosite la antrenare. Această problemă poate să apară fie că avem erori în setul de date, fie arhitectura pe care o folosim nu este bună, fie că avem probleme de convergență din cauză că nu am antrenat suficient de mult modelul, caz în care putem încerca alt algoritm de optimizare sau să schimbăm pasul de învățare. În Figura 2.21 încercăm să clasificăm punctele X și O folosind ca ipoteză ecuația unei drepte, deși datele sunt dispuse după o funcție polinomială. O soluție ar fi să încercăm o ipoteză de forma (15) sau o rețea neurală cu 2 straturi și 3 neuroni.

$$h_{\theta} = \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_1^2 + \theta_4 \cdot x_2^2 + \theta_5 \cdot x_1 \cdot x_2 + \theta_6 \quad (15)$$

Overfitting-ul [10] are loc atunci când avem acuratețe foarte bună pe datele folosite la antrenament, însă modelul nu reușește să generalizeze și pe datele folosite pentru validare sau testare. Această problemă apare în general din cauză că nu avem suficiente date pentru antrenament sau datele pe care le folosim sunt foarte asemănătoare (nu avem diversitate). Pentru a soluționa această problemă există tehnici ce se pot aplica la nivel de model, numite tehnici de regularizare, sau tehnici ce se aplică la nivel de date, numite tehnici de augmentare.

O primă metodă de regularizare [10] presupune adăugarea de informație suplimentară sau zgomot în momentul în care calculăm funcția de pierdere (loss). Regularizarea poate să fie tip L1 (16), ce folosește distanța Manhattan, sau L2 (17), ce folosește distanța Euclidiană. E este funcția de eroare specifică tipului de problemă (regresie sau clasificare), iar λ este un hiperparametru pe care îl putem ajusta și care controlează cât de multă „greutate” să cadă pe parametrii modelului.

$$L(\hat{y}, y) = E(\hat{y}, y) + \lambda \cdot \sum_{i=1}^N |w_i| \quad (16)$$

$$L(\hat{y}, y) = E(\hat{y}, y) + \lambda \cdot \sum_{i=1}^N w_i^2 \quad (17)$$

O altă metodă de regularizare folosită în rețelele neurale este Dropout [10], ce presupune eliminarea aleatorie a unor neuroni din rețea în timpul antrenării, astfel încât aceștia să nu se specializeze pe detectarea unei singure trăsături, ci sarcinile acestora să fie împărțite în mod egal în cadrul rețelei. Când un neuron este eliminat în timpul antrenamentului, un neuron vecin o să fie nevoit să învețe trăsăturile celui dintâi. Dropout-ul se aplică procentual, fie la nivel de strat, % dintre neuroni sunt eliminați aleator în iterația curentă, fie la nivel de rețea, % dintre straturi sunt eliminate aleator în iterația curentă. În timpul inferenței, Dropout-ul este oprit, informația trecând prin toți neuronii din rețea.

În vederea artificială, indiferent de dimensiunea setului de date, augmentarea datelor [10] (adăugarea de informație suplimentară prin distorsionare la nivel de pixel sau geometrie a imaginii) este folosită de fiecare dată, deoarece imaginile conțin foarte multă informație ce trebuie învățată. Pentru o singură imagine la rezoluția 512×512 avem 786432 pixeli, numărul acestora depășind de multe ori numărul de poze din setul de date. Un alt rol important al augmentării datelor este creșterea robusteții rețelei, atunci când întâlnește imagini din medii diferite față de cele pe care a fost antrenată. Dacă am antrena modelele doar pe imagini HD, atunci acestea nu ar putea să generalizeze când le vom testa pe imagini de o calitate redusă.

Augmentarea poate să aibă loc la nivel de pixel, caz în care informația din imagine rămâne în același loc, dar se modifică ușor doar valoarea acestora prin ajustarea proprietăților hue, saturație, expunere (în spațiul HSB) / luminozitate (în spațiul HSL), contrast sau se poate adăuga chiar zgomot gaussian sau efect de blur.

Mai există augmentarea la nivel geometric, în care se modifică poziția și orientarea obiectelor. Se poate realiza prin rotații, punerea în oglindă, redimensionări, decupări (crop) aleatorii de porțiuni sau chiar metode mult mai complexe, în care se îmbină mai multe imagini într-una singură.

2.3.7. Arhitecturi populare

Ca în orice alt domeniu al ingineriei, nu trebuie să reinventăm roata, ci putem să ne uităm pe ce au făcut alții înaintea noastră și să învățăm din greșelile și reușitele lor.

În Figura 2.22, avem una din cele mai simple, populare și eficiente arhitecturi pentru clasificarea imaginilor, numită VGG-16 [13]. Putem să observăm câteva tipare:

- La un număr de 2-3 straturi de convoluții, urmează un strat de Max Pooling. Putem să intuim că operațiile de convoluție sunt cele care calculează trăsăturile imaginii, iar operațiile de Max Pooling le aleg pe acelea care sunt cele mai semnificative;
- Cu cât avansăm mai adânc în rețea, cu atât înălțimea și lățimea imaginii este redusă, dar crește în adâncime (numărul de canale);
- Ultimele straturi sunt formate din neuroni (Fully Connected / denși), trecerea de la volumul 3D la un vector uni-dimensional se face prin operatorul Flatten.

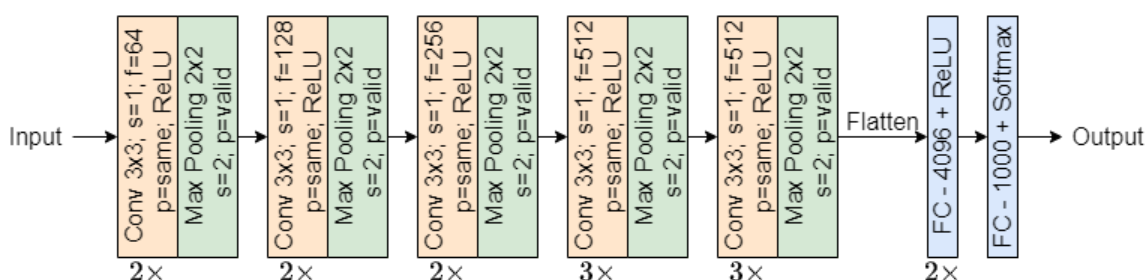


Figura 2.22 Arhitectura VGG-16

În teorie, cu cât o rețea convoluțională este mai adâncă, cu atât este mai performantă și poate să calculeze trăsături mai complexe. În practică, aceasta este o problemă pentru VGG-16 pentru că, după un anumit număr de straturi, rețeaua începe să își piardă din puterea de generalizare. Problema pe care o rezolvă ResNet [14] este cea a gradientilor care se diminuează. Cu cât avansăm mai mult în rețea, dat fiind faptul că lucrăm cu valori normalizate din intervalul 0-1, acestea vor fi din ce în ce mai mici pe măsură ce aplicăm operații matematice asupra lor. În Figura 2.23 avem ilustrat blocul fundamental din arhitectură. Intuiția este că oricât de mare ar fi rețeaua, dacă adăugăm un bloc de tip Skip-connection (Shortcut-connection) care ia input-ul și îl adaugă direct la output, fără să aplice nicio operație matematică asupra lui, atunci performanța nu este afectată deoarece nu s-au modificat datele. Dacă în schimb, cealaltă bucată din bloc reușește să învețe trăsături importante, atunci ele se vor regăsi la ieșire. Cu alte cuvinte, rețeaua este capabilă să învețe funcția identitate atunci când are nevoie. Această proprietate permite crearea de rețele neurale foarte adânci, chiar și de peste 100 de straturi.

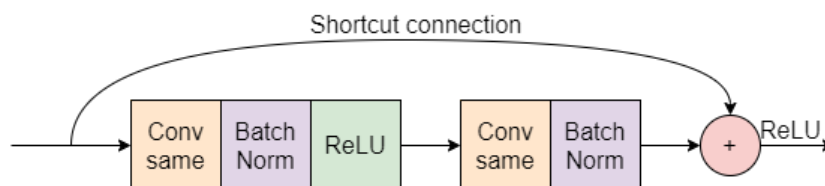


Figura 2.23 Bloc funcțional ResNet

Ultimul tip de arhitectură pe care îl vom analiza se numește Inception [15] și este reprezentat în Figura 2.24. Ideea de bază este că, dacă vrem să creăm o rețea convoluțională și nu știm care este cel mai bun tip de strat pe care să îl folosim, atunci lășăm modelul să decidă singur în timpul procesului de antrenare ce parametri să învețe. La fel ca la ResNet, Inception folosește padding „same” pentru a putea face concatenarea în adâncime a volumelor și, de asemenea, arhitectura permite crearea de rețele neurale foarte adânci.

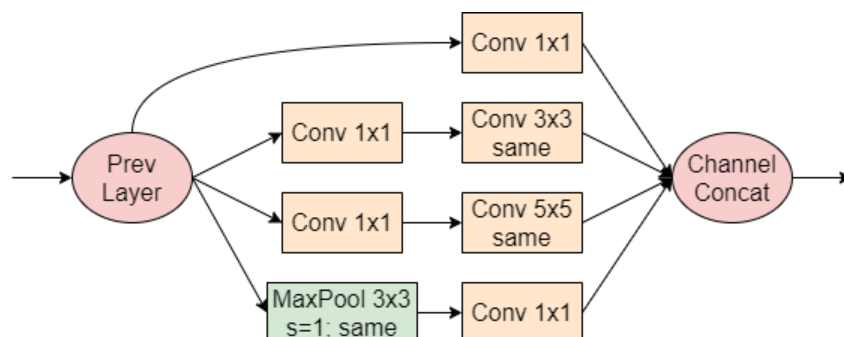


Figura 2.24 Bloc funcțional Inception

2.4. Detecția de obiecte

Detecția de obiecte [10] presupune rezolvarea simultană a două probleme de vedere artificială: clasificarea (ce obiect este într-o imagine) și localizarea (unde se află obiectul în imagine). În plus, clasificarea și localizarea trebuie să aibă loc pentru toate obiectele de interes.

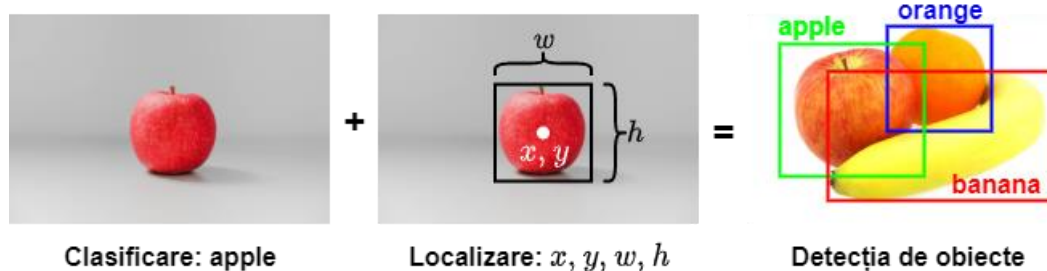


Figura 2.25 Detecția de obiecte

Coordonatele obiectelor pot să fie sub formă centroidă (coordoanatele centrului obiectului x, y , cu înălțimea h și lățimea w a acestuia), cum este reprezentat și în Figura 2.25, sau coordonate carteziene pentru colțurile cutiilor de încadrare: $x_{min}, y_{min}, x_{max}, y_{max}$. Valorile coordonatelor pot să fie absolute (în număr de pixeli), sau normalizate (din intervalul 0.0 - 1.0), avantajul celei de-a doua reprezentări este că valorile normalizate nu depind de dimensiunea imaginii. Există mai multe metode de identificare ce pot fi clasificate în:

- metode de identificare în doi pași: oferă o acuratețe mai mare pentru predicții, cu impedimentul unei viteze de lucru mai scăzute;
- metode de identificare într-un singur pas: nu au o acuratețe la fel de mare, dar sunt mult mai rapide, putând să facă detecții în timp real (peste 30 de cadre pe secundă).

2.4.1. Metoda ferestrei glisante

Reprezintă o abordare naivă, dar intuitivă. Folosește ca instrument de detecție un clasificator antrenat pentru recunoașterea obiectelor, ce este aplicat pe diverse regiuni dintr-o imagine. Regiunile sunt alese secvențial, procedura de identificare începe cu o fereastră de dimensiune inițială ce este mutată cu un pas la fiecare iterație a algoritmului pentru selectarea unei noi porțiuni ce urmează a fi analizată. Clasificatorul poate fi aplicat pe mai multe dimensiuni ale ferestrei și poate fi glisat în orice direcție și cu orice lungime a pasului, condiția fiind să acopere cât mai bine toate zonele de interes dintr-o imagine [10].

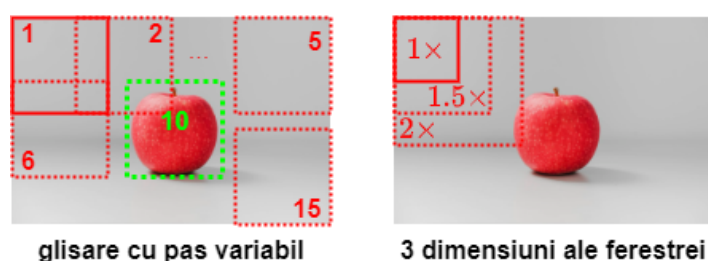


Figura 2.26 Metoda ferestrei glisante

Cu cât pasul de deplasare este mai mic, cu atât numărul zonelor scanate este mai mare, prin urmare acuratețea detectorului crește. Marele dezavantaj al acestei metode este timpul foarte mare de inferență, din cauza numărului mare de clasificări ce au loc.

Pornind de la această idee, s-au dezvoltat diverse îmbunătățiri ale algoritmului pentru creșterea vitezei de execuție. Aceste tehnici poartă denumirea de „metode pentru propunerea de regiuni” [10] și fac parte din clasa metodelor de detecție în doi pași. R-CNN, Fast R-CNN și Faster R-CNN sunt câteva exemple de metode ce folosesc în prima etapă un algoritm pentru selectarea regiunilor care au probabilitatea cea mai mare să reprezinte un obiect și apoi, un clasificator pentru analizarea acestora. În funcție de tipul metodei folosite, timpul de predicție poate să fie de 50, 2.5 sau 0.2 secunde în cazul ultimului algoritm.

2.4.2. You Only Look Once

YOLO [16] este o metodă de identificare într-un singur pas, ce folosește o rețea neurală convoluțională, pentru detecția tuturor obiectelor de interes dintr-o imagine. Marele avantaj al acestei metode față de cele de identificare în doi pași și, inclusiv față de restul algoritmilor într-un singur pas (SSD, RetinaNet), este viteza foarte mare de predicție (posibil cea mai mare) la o acuratețe foarte apropiată de cea a metodelor clasice de detecție. YOLO este un model robust, ce face parte din categoria metodelor „state of the art” pentru detecția de obiecte (fiind algoritmul implicit din biblioteca OpenCV) și este în continuare actualizat și îmbunătățit în conformitate cu cele mai noi standarde și dezvoltări din domeniul inteligenței artificiale. Modelul este în prezent la a 4-a versiune oficială, ultimul articol fiind publicat recent față de momentul redactării acestei lucrări (23 aprilie 2020).

2.4.2.1. Principiul de funcționare

Modelul YOLO [17] operează pe imagini de dimensiuni multiplu de 32 (416×416 , 512×512 , 608×608). Se alege o rezoluție pentru care se dorește antrenarea modelului și se redimensionează toate pozele din setul de date. Cu cât rezoluția este mai mare, cu atât acuratețea modelului crește și poate să identifice obiecte de dimensiune mai mică, dar în același timp crește timpul necesar antrenării și scade viteza de predicție. Se aplică un grilaj (grid) peste fotografia redimensionată și se împarte imaginea în blocuri de dimensiune 32×32 (pixeli). Pentru fiecare bloc creat, algoritmul realizează o predicție și încearcă să atribuie centrul obiectului cel mai reprezentativ din căsuța respectivă. Dacă un obiect ocupă mai multe blocuri, doar centrul acestuia este atribuit o singură dată căsuței corespunzătoare. În prima imagine din Figura 2.27 se poate observa cum centrul fiecărui fruct este atribuit în blocul potrivit.

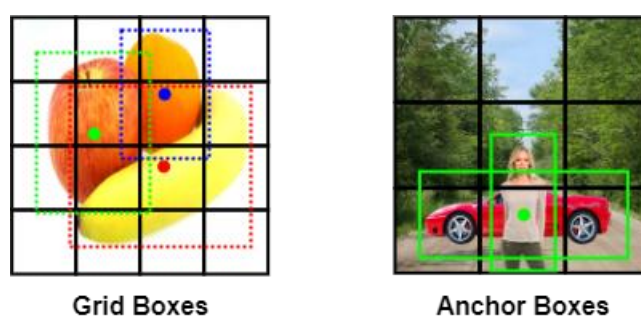


Figura 2.27 YOLO - Principiul de funcționare

Pentru obiectele suprapuse, care au centrul în același bloc din grid, YOLO [18] folosește o tehnică bazată pe „cadre de ancorare” pentru identificarea corectă a obiectelor în funcție de forma și orientarea acestora. În a doua imagine din Figura 2.27 se observă cum obiectele sunt identificate corect folosind un dreptunghi mai înalt pentru persoană și unul mai lat pentru mașină. Cadrele de ancorare pot să fie alese manual sau cu ajutorul unui algoritm de clustering, condiția fiind să încadreze cât mai bine geometria obiectelor din setul de date.

În primele două versiuni ale algoritmului, modelul avea dificultate în a identifica obiectele de dimensiune mică. Pentru a rezolva această problemă, autorii au adus în versiunea 3 a modelului YOLO [19] o arhitectură ce permite scanarea imaginilor la 3 scări de măsură diferite, pentru a detecta obiectele de dimensiune mică, medie și mare, asemănător cu cele 3 dimensiuni ale ferestrei din Figura 2.26. Practic, algoritmul o să împartă imaginea în blocuri de dimensiune 32×32 și apoi, în blocuri de două și de 4 ori mai mici (16×16 și 8×8).

În final, algoritmul o să realizeze predicții de dimensiunea $H \times W \times K \times (1 + 4 + C)$, unde H și W sunt numărul de blocuri în înălțime și lățime, K este numărul cadrelor de ancorare, iar a 4-a dimensiune este un vector de forma $\hat{y} = [p, x_c, y_c, w, h, c_1, c_2, c_3 \dots]$, unde p reprezintă probabilitatea ca în căsuța curentă să existe un obiect, urmat de coordonatele cutiei de încadrare x_c, y_c, h, w , iar c_i reprezintă probabilitățile (disjuncte) de apartenență ale obiectului la clasele respective. Dacă p nu depășește un anumit prag, restul valorilor nu sunt luate în considerare.

În cazul modelelor YOLOv3 și YOLOv4, dat fiind faptul că detecția se face la 3 scări diferite de măsură și folosim 9 cadre de ancorare, pentru o imagine la rezoluția 416×416 vom avea dimensiunile 13×13 , 26×26 , 52×52 pentru cele 3 ferestre ($\frac{416}{32} = 13$), deci un total de $3 \cdot 13 \cdot 13 + 3 \cdot 26 \cdot 26 + 3 \cdot 52 \cdot 52 = 10647$ de detecții. Rezultatul este apoi procesat printr-un algoritm de supresie maximală (NMS) și redus la numărul real de obiecte din imagine. Pentru a reduce numărul predicțiilor și a păstra doar datele relevante, YOLO folosește un coeficient numit IoU [10] (Intersection over Union), pentru măsurarea gradului de suprapunere dintre două cutii de încadrare și este definit astfel (18):

$$IoU = \frac{Aria(\cap)}{Aria(\cup)} \quad (18)$$

După cum se poate observa și în Figura 2.28, în a doua imagine $IoU = \frac{20}{30+40-20} = 0.4$. Dacă obiectele nu sunt în contact, IoU este 0, iar dacă se suprapun complet, atunci IoU va fi 1.

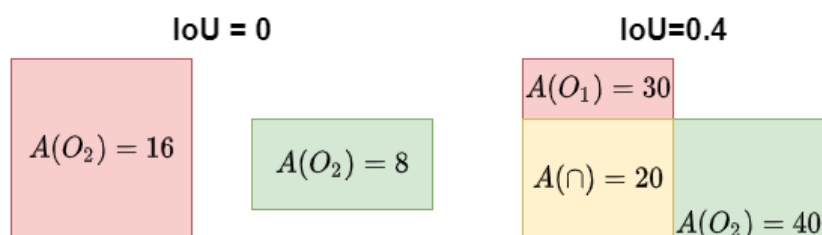


Figura 2.28 IoU

Cu ajutorul acestui coeficient putem defini astfel algoritmul NMS [10]:

- 1) Se elimină toate cutiile de încadrare care au scorul de încredere mai mic decât un prag p_1 . Cu cât pragul e mai mare, cu atât crește șansa ca detecțiile făcute să fie corecte.
- 2) Cât timp mai există cutii de eliminat:
 - a. Se alege cutia de încadrare cu scorul cel mai mare de încredere;
 - b. Se elimină toate cutiile de încadrare care au un IoU mai mare decât un prag p_2 față de cutia anterioară aleasă.
- 3) Se afișează cutiile de încadrare selectate ca predicții ale modelului.

În Figura 2.29 putem să observăm o reprezentare grafică pentru modul de funcționare al algoritmului NMS. Valurile uzuale pentru cele două tipuri de praguri sunt 0.5.

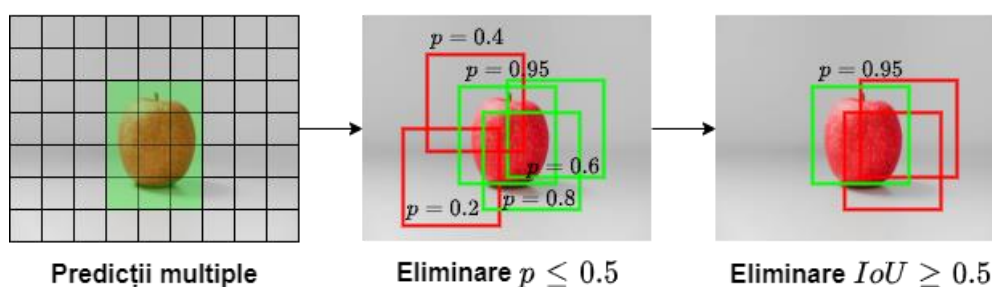


Figura 2.29 Algoritm de supresie maximală

2.4.2.2. Arhitectura și evoluția modelului

La modul general [20], un algoritm pentru detecția de obiecte este format dintr-un corp (body), care este un CNN antrenat pentru a clasifica ce obiect este într-o imagine și are rolul de a extrage trăsăturile importante din acesta și un cap (head), care descrie modul în care se face detecția. În cazul YOLO, capul reprezintă implementarea pașilor descriși în secțiunea anterioară (Grid Boxes + Anchor Boxes + NMS etc.).

În lucrarea versiunii 4 a algoritmului [20] mai este menționată o componentă suplimentară numită gât (neck), formată dintr-un set de elemente auxiliare adăugate între head și body pentru creșterea performanței. Tot aici sunt prezentate diverse metode ce pot fi folosite în timpul antrenării pentru creșterea acurateții modelelor. Acestea poartă denumirea de „Bag of freebies” (BoF) sau metode ce schimbă strategia de antrenare fără să crească timpul de inferență (augmentarea de tip mozaic, DropBlock, CIoU) și „Bag of specials”, care cresc foarte puțin timpul de inferență, dar duc și la o creștere a acurateții (Mish, SPP, DIoU-NMS).

Pentru partea de body s-au încercat mai multe tipuri de arhitecturi (variații de ResNet și EfficientNet) care sunt foarte bune pe problema clasificării, dar nu se descurcă la fel de bine pentru detecție [20]. Autorii au creat propria arhitectură specializată pentru detecția de obiecte numită Darknet (în versiunea 3 și 4 se folosesc variații diferite ale arhitecturii Darknet53 care are 53 de straturi convoluționale) și care are următoarele trăsături distinctive:

- Blocurile convoluționale fundamentale folosesc toate un kernel de 3×3 (care au un câmp receptiv foarte mare [20]), pas de deplasare $\text{strides}=1$, padding valid și sunt folosite în combinație cu straturi de Batch Normalization.
- Pentru a permite crearea unei rețele cât mai adânci, autorii folosesc blocuri reziduale de tip Skip-connection, asemănătoare cu cele din ResNet.
- În ultimele două versiuni ale modelului nu se mai folosesc combinațiile clasice din CNN-uri de tipul Conv + MaxPooling (vezi VGG-16), deoarece autorii au considerat ca operațiile de pooling duc la pierderea de informație importantă [19].
- Primele 3 versiuni ale algoritmului YOLO foloseau activări de tip Leaky ReLU atât pentru partea de body și head, iar în ultima versiune modelul implementează funcția de activare de tip Mish pentru partea de body și Leaky ReLU în rest.
- În ultimele două versiuni, în locul straturilor dense de neuroni (Fully-Connected) din CNN se folosesc convoluții cu kernel 1×1 , asemănătoare cu cele din Inception, ce permit antrenarea și testarea aceluiași model pe dimensiuni diferite ale imaginii. Practic, dacă mărim rezoluția imaginii de la 416×416 la 608×608 , se vor crea mai multe grid box-uri (de la 13×13 la 19×19). Din punct de vedere funcțional, acestea sunt echivalente cu un strat dens de neuroni aplicat în adâncimea fiecărei căsuțe din grid.

Funcția obiectiv [17] a sistemului este definită ca o combinație între o problemă de regresie și una de clasificare. Pentru coordonatele cutiei de încadrare se folosește funcția de cost MSE între predicțiile pentru fiecare celulă din grid și etichetele corespunzătoare acesteia (dacă există), iar pentru partea de clasificare se folosește o funcție de cost bazată pe cross-entropie binară pentru probabilitatea de existență a unui obiect în celulă și probabilitățile de apartenență a obiectului la o clasă. De asemenea, se mai folosesc și niște termeni de corecție [19] pentru a da o „greutate” mai mare obiectelor de dimensiune mică (care nu vor avea un

impact așa de mare asupra costului, deoarece distanțele dintre etichete și predicții sunt mai mici) și o metodă numită „pierdere focală” (focal loss) pentru a rezolva problema seturilor de date nebalansate (atunci când există o discrepanță mare între numărul de obiecte etichetate pentru fiecare clasă). În final, toți acești termeni se însumează și formează costul final al modelului. Deoarece MSE tratează coordonatele cutiilor de încadrare ca variabile independente și nu ține cont de integritatea obiectelor, în versiunea 4 [20] se folosește o funcție de pierdere bazată pe IoU, numită CIOU, ce tratează implicit obiectele în mod egal, indiferent de dimensiune și, în plus, ține cont și de aria de suprapunere, distanța dintre centrele obiectelor și proporția lor.

Alte aspecte interesante de menționat în legătură cu evoluția algoritmului ar fi că, în a doua lucrare științifică [18], este prezentat un mod prin care se poate antrena modelul atât cu date etichetate pentru detecție în mod normal, dar și cu imagini etichetate pentru clasificare, prin înghețarea componentelor ce țin de localizare. Astfel, se poate profita de numărul foarte mare de date etichetate pentru problemele de clasificare. În plus, în versiunea 4 a modelului [20], alegerea unora dintre hiper-parametrii modelului s-a făcut cu algoritmi genetici.

La finalul lucrării se pot vedea, în Anexa B, arhitecturile pentru două modele de tip YOLOv3 și YOLOv3-tiny, iar în Anexa C se poate vedea un sumar al numărului de parametri învățabili pentru acestea.

2.5. Recunoașterea facială

Recunoașterea facială [10] este un proces complex format din mai multe etape. Într-o primă etapă avem partea de detectare a feței într-o imagine. Pentru această parte putem folosi orice tip de detector antrenat special pentru această sarcină. Se decupează fața identificată și este trimisă mai departe către un CNN, care extrage un număr de 127 de trăsături distinctive ale chipului numite puncte „landmarks”. După cum se observă și în Figura 2.30, aceste puncte reprezintă poziția, forma și orientarea anumitor elemente ce țin de fizionomie și sunt unice pentru fiecare persoană. În final, se compară trăsăturile extrase cu cele ale fețelor dintr-o bază de date și se încearcă identificarea persoanei.

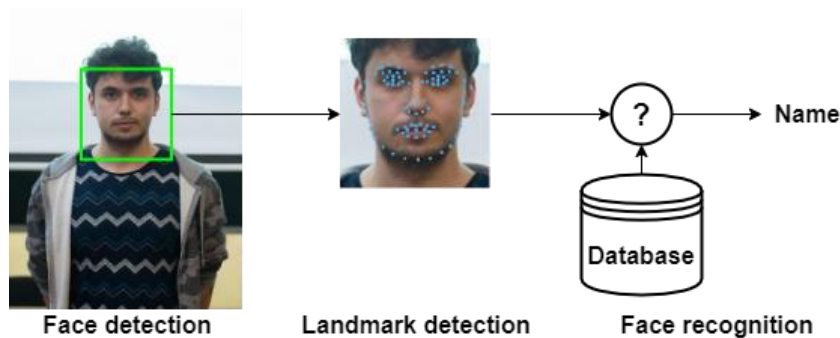


Figura 2.30 Recunoașterea facială

Din punct de vedere arhitectural, rețeaua folosită pentru identificarea landmark-urilor este o rețea convoluțională simplă, asemănătoare cu VGG-16, antrenată pentru a clasifica binar

dacă în imagine există o față sau nu. Se elimină ultimul strat din CNN iar neuronii din penultimul strat vor reprezenta punctele de reper pentru trăsăturile feței. Pentru a identifica cine este într-o imagine, se antrenează un model bazat pe o „rețea siameză” [10], în care sunt folosite două CNN-uri pentru identificarea landmark-urilor puse în paralel. Funcția obiectiv a sistemului încearcă să minimizeze costul dacă cele două imagini de la intrare sunt ale aceleiași persoane și să îl maximizeze dacă sunt persoane diferite. O astfel de rețea este antrenată pe un set de date ce conține imagini cu persoane diferite ce formează o etichetă negativă și fiecare persoană are mai multe poze cu propria față ce vor forma o etichetă pozitivă.

În cadrul proiectului am folosit o rețea pre-antrenată pentru identificarea și recunoașterea facială a persoanelor din imagini, pe care doar am conectat-o la arhitectura completă a sistemului ca o componentă software.

2.6. Urmărirea de obiecte

Detecția de obiecte este o aplicație a vederii artificiale, ce presupune localizarea și clasificarea obiectelor în imaginile statice. Pentru a face identificarea pe o secvență video, putem să aplicăm un detector pe fiecare cadru din aceasta. Dacă folosim un algoritm de detecție precum YOLO, care este foarte rapid (are un timp de inferență mic), atunci vom obține rezultatele predicțiilor în timp real.

Urmărirea de obiecte (object tracking) [21] este un domeniu al vederii artificiale axat special pe detectarea și urmărirea obiectelor în secvențele video. Din punct de vedere funcțional, algoritmi de object tracking sunt formați dintr-un detector antrenat pentru identificarea claselor de interes și un tracker. Tracker-ul folosește trăsăturile obținute de la detector și analizează informațiile spațio-temporale pentru a prezice direcția / traiectoria pe care o urmează obiectul. Pentru fiecare obiect detectat se atribuie un ID și este urmărit până când acesta dispăre din cadru.

Din punct de vedere arhitectural, algoritmi de Deep Learning pentru urmărirea de obiecte pot fi împărțiți în două categorii. Prima categorie este formată din algoritmi ce folosesc rețele neurale recurente pentru obținerea trăsăturilor spațio-temporale. Un exemplu de astfel de algoritm este ROLO [22] (Recurrent YOLO) care folosește ca detector algoritmul YOLO, un CNN pentru extragerea trăsăturilor spațiale și este urmat de o celulă LSTM pentru extragerea trăsăturilor temporale din detecțiile precedente ale modelului. A doua categorie este formată din metode ce folosesc filtre Kalman. Pe baza unei detecții din trecut, detecției curente și a informațiilor temporale pe care le obținem din durata dintre cele două cadre, filtrele Kalman estimează viteza pe care o are obiectul și care va fi poziția lui viitoare.

DeepSort [23] este modelul pe care am ales să îl folosesc în această lucrare. Este un model robust la cele mai comune tipuri de probleme pentru urmărirea de obiecte: urmărirea de obiecte multiple, obiecte suprapuse, urmărirea pe mai multe camere, camere nestăționare, schimbări de fundal la variația luminii. Nu voi prezenta principiul de funcționare deoarece în cadrul proiectului am folosit un model pre-antrenat, pe care l-am integrat ca pe o componentă software instalabilă (plugin) în arhitectura completă a sistemului.

3. CERCETARE ȘI ANALIZĂ

Datele și algoritmii reprezintă cele două componente esențiale în dezvoltarea soluțiilor de inteligență artificială. Având fixate deja noțiunile teoretice fundamentale pentru a înțelege modul de funcționare al algoritmilor și care sunt aspectele importante ce trebuie luate în calcul, în acest capitol o să prezint o analiză tehnică asupra resurselor disponibile, soluțiilor deja existente și o să justific alegerea metodelor folosite.

3.1. Setul de date

Datele reprezintă combustibilul algoritmilor de inteligență artificială și motivul pentru care acest domeniu s-a dezvoltat atât de mult în ultimii ani. Cele mai bune modele nu sunt create neapărat de cei care au cel mai bun algoritm, ci de cei care dețin cele mai multe date [24]. Pentru algoritmii de vedere artificială este cu atât mai important să avem un volum cât mai bogat și divers de date, întrucât imaginile conțin foarte multe trăsături ce trebuie învățate și astfel reușim să prevenim fenomenul de overfitting.

În inteligența artificială, atunci când vrem să rezolvăm o problemă și să creăm o soluție, primul lucru pe care trebuie să îl avem în vedere este găsirea unui set de date corespunzător. În mediul online există mai multe seturi de date foarte populare și standardizate: MNIST [25] și ImageNet [26] pentru clasificare, Pascal VOC [27] și MS COCO [28] pentru detecția de obiecte sau chiar platforme de ML cu foarte multe seturi de date precum Kaggle [29]. Problema pe care am abordat-o, identificarea echipamentului personal de protecție (PPE), nu este una foarte comună, fapt ce se observă prin lipsa unor seturi de date consistente și accesibile pe internet. Pentru detecția de obiecte este cu atât mai dificilă găsirea unui set de date întrucât nu este suficientă doar colectarea imaginilor, ce se poate face prin niște script-uri de Web Scraping⁴, ci trebuie, în plus, să le etichetăm și să încadrăm fiecare obiect din acestea.

Din fericire, un grup de cercetători s-a gândit deja la acest lucru, lipsa unui set de date pentru problemele de detecție și, demonstrează în articolul [30], cum jocurile video pot fi folosite pentru generarea unor seturi de date virtuale, care reprezintă o soluție viabilă pentru antrenarea modelelor de Deep Learning. Motivul pentru care această metodă funcționează este că, în prezent, jocurile video au o grafică foarte performantă, aproape imperceptibilă de realitate. Autorii au folosit jocul GTA V și game engine-ul numit RAGE pentru a genera automat oameni cu sau fără echipament de protecție în 30 de locații diferite ale hărții și care să eticheteze automat imaginile capturate. În teorie, această metodă ar putea să automatizeze complet procesul de etichetare și ar putea reprezenta, în viitor, o sursă infinită de date.

Prin urmare, punctul de plecare în cadrul proiectului dezvoltat este setul de date virtual prezentat anterior, la care se mai adaugă un set de date real etichetat de către aceiași autori, mai multe seturi de date ce au etichete doar pentru căștile de protecție și un set de date cu imagini de pe un șantier din România pe care le-am etichetat personal.

⁴ Web Scraping: crearea unor script-uri de parsare a paginilor Web pentru colectarea de informații.

3.2. Metrice de performanță

Atunci când dezvoltăm un produs este important să avem definit un indice numeric [9] care să ne spună cât de performant este modelul nostru și cum se comportă în comparație cu alte soluții deja existente. În capitolul de teorie am observat că obiectivul algoritmilor de inteligență artificială este minimizarea unei funcții obiectiv. Costul funcției este o metrică utilă care ne ajută să ne dăm seama dacă modelul nostru converge, însă nu ne spune foarte multe informații deoarece costul depinde exclusiv de setul de date pe care antrenăm modelul și de algoritmul folosit. În problemele de clasificare un bun indice de performanță este acuratețea. Aceasta măsoară numărul de predicții corecte realizate din numărul total de imagini din setul de date. Problema acurateții este aceea că nu oferă foarte multe informații asupra cauzelor erorilor și nu este foarte relevantă dacă avem un set de date dezechilibrat (dacă 99% din imagini sunt cu pisici și 1% sunt cu câini, iar clasificatorul nostru identifică corect toate imaginile din prima clasă, atunci acuratețea este de 99%). Alte metrice mai specifice pentru măsurarea performanței sunt precizia (19) și recall-ul (20), unde TP (True Positives), FP (False Positives), FN (False Negatives). Pozitiv și negativ se referă la existența sau absența etichetei. TP înseamnă că rezultatul predicției ar trebui să fie pozitiv și este pozitiv, FP înseamnă că rezultatul predicției ar trebui să fie negativ (nu este etichetat), dar este pozitiv, iar FN înseamnă că rezultatul ar trebui să fie pozitiv (este etichetat), dar este negativ. În termeni mai generali [31], precizia măsoară cât de corecte sunt predicțiile noastre, iar recall-ul măsoară cât de bine detectăm obiectele etichetate. Din necesitatea de a avea un singur indice numeric pentru evaluarea performanței modelelor, s-a introdus scorul F1, care este media armonică dintre precizie și recall și este definit în formula (21).

$$Precision = \frac{TP}{TP + FP} \quad (19)$$

$$Recall = \frac{TP}{TP + FN} \quad (20)$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (21)$$

Detecția de obiecte este o combinație între o problemă de regresie și una de clasificare, prin urmare metricile definite anterior nu pot fi aplicate direct deoarece nu țin cont și de procentul de suprapunere al detecțiilor cu obiectele etichetate. Precizia medie (mean Average Precision - mAP) [31] este metrica specială folosită în cazul detecției de obiecte, pentru a măsura „acuratețea” detectoarelor și este o combinație între elementele definite anterior. Din punct de vedere matematic, AP este aria de sub graficul format din reprezentarea preciziei detecțiilor în funcție de recall și pentru o anumită valoare a indicelui IoU. AP_{50} este AP calculat pentru un $IoU=0.5$, AP_{75} pentru $IoU=0.75$ și așa mai departe. mAP este suma AP-urilor de la $IoU=0.5$ până la $IoU=0.95$ cu o subdiviziune de 0.05. Mai există anumite variații ale preciziei medii definite în funcție de dimensiunea obiectelor (small, medium, large), iar în anumite situații, mAP este media AP-urilor pentru un anumit IoU între toate clasele obiectelor din setul de date. Cu toate acestea, mAP este metrica standardizată pentru măsurarea performanței detectoarelor și este folosită de toți cercetătorii și în toate lucrările științifice pentru compararea rezultatelor modelelor dezvoltate.

3.3. Alegerea metodelor

Am văzut deja că metodele folosite pentru detecția de obiecte pot fi clasificate în două categorii (în 2 pași sau într-un singur pas) și care este compromisul dintre viteza și acuratețea acestora. Domeniul inteligenței artificiale, fiind unul foarte dinamic, nu are un câștigător definitiv legat de care este cel mai bun algoritm de detecție, clasamentele schimbându-se de la an la an și sunt echilibrate în funcție de mai mulți factori.

A fost o perioadă în care detectoarele în doi pași, bazate pe propunerea de regiuni, erau foarte populare (R-CNN 2013, Fast R-CNN 2015, Faster R-CNN 2015), a urmat apoi o perioadă în care detectoarele într-un singur pas au avut o ascensiune (SSD 2015, YOLOv1-3 2015-2018, RetinaNet – 2017), iar în prezent există o serie de soluții hibride ce folosesc detecția într-un singur pas fără cadre de ancorare [20] (CenterNet, CornerNet, FCOS) sau chiar arhitecturi ce duc detecția de obiecte la următorul nivel și realizează inclusiv segmentarea obiectelor din imagini (Mask R-CNN). O să observăm că în funcție de bibliotecile software pe care le folosim, vom avea modele diferite ce implementează detecția de obiecte (Faster R-CNN în Tensorflow, YOLOv3 în OpenCV). De cele mai multe ori nu există o soluție universal valabilă, performanța metodelor utilizate depinzând atât de setul de date folosit, cât și de problema pe care o rezolvă.

Cu toate acestea, cercetătorii încearcă să își compare rezultatele testând performanța modelelor pe niște seturi de date standard. În majoritatea lucrărilor științifice, testul standard pentru calcularea performanței unui detector este antrenarea modelelor pe setul de date MS COCO și validarea acestora cu indicele mAP. Conform articolului scris de către autorul principal al arhitecturii YOLOv4 [32], modelul lor este cel mai performant algoritm de detecție în timp real pe testul standard, depășind toate celelalte implementări (Google EfficientDet, Facebook Detectron, RetinaNet, SSD, Mask R-CNN, Faster R-CNN, YOLOv3). În Tabel 3.1 putem să observăm o comparație între performanțele mai multor detectoare în funcție de backbone (body) și dimensiunea imaginii la intrarea în rețea (size), precum și valorile indicilor de performanță (AP pentru precizie și FPS pentru viteză). Viteza depinde de tipul plăcii grafice pe care rulează modelul, prin urmare (M) Maxwell, (P) Pascal, (V) Volta sunt arhitecturile de GPU pe care au fost testate, ordonate crescător în ordinea performanței.

Tabel 3.1 Performanțele algoritmilor de detecție⁵

| Algoritm | Backbone | Size | AP(%) | AP ₅₀ (%) | AP ₇₅ (%) | FPS |
|---------------|---------------|------|-------|----------------------|----------------------|-------------------|
| YOLOv4 | CSPDarknet-53 | 416 | 41.2 | 62.8 | 44.3 | 38(M) 54(P) 96(V) |
| YOLOv4 | CSPDarknet-53 | 512 | 43.0 | 64.9 | 46.5 | 31(M) 43(P) 83(V) |
| YOLOv4 | CSPDarknet-53 | 608 | 43.5 | 65.7 | 47.3 | 23(M) 33(P) 62(V) |
| YOLOv3 | Darknet-53 | 416 | 31.0 | 55.3 | 32.3 | 35(M) |
| YOLOv3 | Darknet-53 | 608 | 33.0 | 57.9 | 34.4 | 20(M) |

⁵ Rezultatele din tabel au fost preluate din lucrarea modelului YOLOv4 [20].

| | | | | | | |
|------------------------|-----------------|-----|------|------|------|---------------|
| SSD | VGG-16 | 300 | 25.1 | 43.1 | 25.8 | 43(M) |
| SSD | VGG-16 | 512 | 28.8 | 48.5 | 30.3 | 22(M) |
| RetinaNet | ResNet-50 | 640 | 37.1 | 56.9 | 40.0 | 10.8(P) 37(V) |
| RetinaNet | ResNet-101 | 640 | 37.9 | 57.5 | 40.8 | 29.4(V) |
| EfficientDet-D0 | EfficientNet-B0 | 512 | 33.8 | 52.2 | 35.8 | 62.5(V) |
| EfficientDet-D1 | EfficientNet-B1 | 640 | 39.6 | 58.6 | 42.3 | 50(V) |
| Faster R-CNN | ResNet-50 | - | 39.8 | 59.2 | 43.5 | 9.4(P) |

Am ales să folosesc modelul YOLO deoarece am considerat că în problema pe care încerc să o rezolv, identificarea echipamentului de protecție personală, este mai importantă viteza cu care au loc detecțiile decât acuratețea acestora. Clasele pe care vreau să le identific (cască, vestă reflectorizantă etc.) sunt oricum foarte ușor de recunoscut datorită formei, dar mai ales culorii (în general galben pentru căști și verde pentru veste), prin urmare nu ar trebui să fie o dificultate pentru algoritm să le învețe. Metoda YOLO identifică foarte bine obiectele de dimensiune mică datorită arhitecturii rețelei ce scanează imaginea la 3 scări de mărime, fiind un avantaj în cadrul șantierelor de construcții, acolo unde camerele de supraveghere vor fi cel mai probabil amplasate în puncte îndepărtate față de zona de lucru. De asemenea, a fost demonstrat că algoritmul este robust și generalizează foarte bine pe imagini care sunt diferite față de cele pe care a fost antrenat modelul, fiind invariant la schimbări de cadru și peisaj. Nu în ultimul rând, există un balans între acuratețe și viteză, dat de dimensiunea imaginii la intrarea în rețea, deci oricând se pot ajusta acești hiper-parametri fără a fi nevoiți să re-antrenăm modelul.

3.4. Soluții existente

Înainte de rezolvarea efectivă a unei probleme este important să începem cu o etapă de cercetare și să ne documentăm dacă soluția la care ne-am gândit este unică / inovatoare sau dacă au mai încercat și alte persoane înaintea noastră. Astfel putem să observăm ce rezultate s-au mai obținut, care sunt obstacolele pe care le-au întâmpinat, care sunt minusurile acelor aplicații și ce putem noi să aducem în plus și, nu în ultimul rând, proiectele precedente pot să constituie un punct de referință față de care să ne comparăm.

Problema identificării echipamentului de protecție este una destul de nouă, fapt ce se observă prin lipsa unui set de date real și consistent pe internet. La momentul în care m-am hotărât ce anume vreau să fac pentru proiectul de diplomă (octombrie 2019), singura resursă pe care am găsit-o la momentul respectiv a fost setul de date virtual [33] împreună cu lucrarea științifică a autorilor [30], care tocmai fusese publicată în aceeași lună. Între timp am reușit să mai colectez câteva seturi de date de pe internet, dar care conțin doar etichete pentru cască de protecție. Recent am descoperit că există și o companie care vinde sisteme de supraveghere pentru detectarea echipamentului personal de protecție. Însă nu am reușit să identific un proiect open-source sau modele antrenate pentru detecția echipamentului personal de protecție.

Prin urmare, lucrarea la care mă voi raporta în etapa de evaluare a modelelor este cea a autorilor setului virtual de date. În aceasta există o secțiune în care sunt trecute în niște tabele scorurile mAP obținute pentru modelul lor la diferite iterații (număr de batch-uri), rezultatele obținute pe setul de validare cu date virtuale și rezultatele obținute pentru testarea pe date reale. De asemenea, aceștia prezintă și strategia de antrenare pe care au folosit-o. Obiectivul meu principal în acest proiect este să creez un model pentru detectarea echipamentului personal de protecție care să fie viabil și să depășească rezultatele obținute de autori.

4. RESURSE UTILIZATE

4.1. Resurse software

4.1.1. Python

Python [34] este un limbaj de programare de nivel înalt, interpretat, multi-paradigmatic, cu tipuri de date dinamice și cu o sintaxă simplă, care ajută programatorii să scrie cod într-un mod cât mai organizat. Este unul dintre cele mai populare limbaje de programare datorită numărului foarte mare de pachete software dezvoltate, fiind folosit în majoritatea domeniilor din zona IT: Web, Data Science, Machine Learning, Scripting sau chiar Embedded.

Python este un bun limbaj de prototipare, deoarece abstractizează acele concepte de programare care sunt consumatoare de timp, precum declararea tipurilor de date, management-ul memoriei, scrierea de cod redundant, programatorul putând astfel să se axeze pe rezolvarea efectivă a problemei. În același timp, fiind un limbaj interpretat, este lent ca timp de execuție (de aproximativ 72 de ori mai încet ca limbajul C [35]). Pentru a rezolva problema vitezei, s-au dezvoltat diverse tehnici de accelerare bazate pe modul în care este interpretat codul (PyPy, Cython etc.). Numba [36] este un compilator în timp real (JiT-compiler), care transformă funcțiile anotate corespunzător în cod mașină prima dată când sunt invocate, iar apoi folosește varianta pre-compilată a acestora pentru viitoarele execuții. Aceste soluții de creștere a vitezei sunt utile doar pentru anumite cazuri de utilizare și nu oferă performanța unui limbaj de nivel scăzut. Cu toate acestea, Python este cel mai popular limbaj de programare folosit în domeniul inteligenței artificiale unde viteza de execuție este foarte importantă. Motivul este că Python are integrare foarte bună cu limbajele de nivel scăzut, mai exact poate fi folosit ca un API de nivel înalt peste implementările eficiente ale algoritmilor scrise în C/C++.

Am ales să folosesc Python ca limbaj principal de programare în dezvoltarea proiectului de diplomă datorită numărului foarte mare de pachete software disponibile pentru domeniul inteligenței artificiale și implementările eficiente ale acestora, extensibilitatea limbajului cu alte platforme (Web, IoT) și vitezei de prototipare. Am folosit distribuția de Python3 oferită de Anaconda [37], care asigură stabilitate pe sistemul de operare Windows și un număr mare de module software pre-instalate pentru Data Science și Machine Learning.

4.1.2. Numpy și Matplotlib

NumPy [38] este o bibliotecă software pentru limbajul Python optimizată pentru calculul vectorial și matriceal. Aceasta stă la baza majorității framework-urilor și bibliotecilor de inteligență artificială (Tensorflow, PyTorch etc.). De asemenea, conține o colecție mare de funcții matematice predefinite, asemănător cu limbajul Matlab. Matplotlib [39] este o bibliotecă software folosită pentru crearea de grafice și vizualizarea datelor.

Am folosit cele două pachete software pentru prelucrarea setului de date, crearea de diverse scripturi pentru automatizarea unor sarcini, vizualizarea datelor și, în mod indirect, folosind framework-urile de inteligență artificială ce urmează a fi prezentate.

4.1.3. OpenCV

Open Computer Vision [40] este o bibliotecă software, open-source, destinată prelucrării imaginilor și vederii artificiale. A fost dezvoltată inițial în limbajul C++ și integrată apoi în Python și în alte limbaje de programare sub formă de API.

Din seria funcțiilor și metodelor pe care le-am folosit în cadrul proiectului se numără: funcții pentru conversia imaginilor între diferite formate (PNG, JPG), redimensionare, rotații, modificarea parametrilor hue, saturation, brightness, aplicarea de blur și zgomot gaussian, funcții pentru desenarea de figuri și scrierea de text pe imagini, accesul la camera web, realizarea de fotografii și videoclipuri. Biblioteca conține inclusiv modele pre-antrenate pentru detecția de obiecte și recunoaștere facială ce pot fi folosite printr-un simplu apel de funcție.

4.1.4. Tensorflow

Tensorflow [41] este framework-ul de Deep Learning dezvoltat de Google și principalul instrument software pe care l-am folosit în lucrare. Ca mod de funcționare, acesta folosește o reprezentare bazată pe grafuri computaționale [10] pentru evaluarea expresiilor matematice. În Figura 4.1 avem o astfel de reprezentare a funcției obiectiv $J(x, y, z) = 2 \cdot (x^2 + yz)$. Colorat în albastru avem echivalentul procesului de inferență, pornind de la valorile parametrilor $x = 2, y = 3, z = 5$ și parcurgând graful de la stânga la dreapta ajungem la valoarea 38, care reprezintă costul final al funcției, iar cu verde avem algoritmul de propagare înapoi pentru calcularea gradientelor în timpul procesului de antrenare, scopul final fiind determinarea valorilor dx (22), dy (23), dz (24), ce vor fi folosite pentru actualizarea parametrilor x, y, z cu algoritmul de optimizare ales. Ca utilizatori ai unui astfel de framework, tot ce trebuie să facem este să ne definim din punct de vedere matematic funcția obiectiv pe care dorim să o minimizăm iar, pe baza grafului generat automat, framework-ul o să calculeze valorile gradientelor și o să actualizeze parametrii modelului.

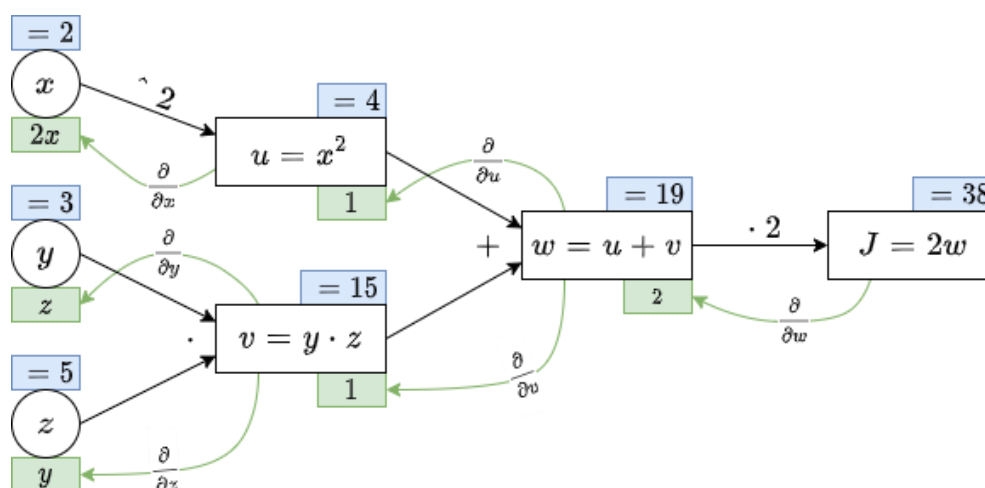


Figura 4.1 Graf computațional

$$dx = \frac{\partial J}{\partial x} = \frac{\partial u}{\partial x} \cdot \frac{\partial w}{\partial u} \cdot \frac{\partial J}{\partial w} = 2x \cdot 1 \cdot 2 = 4x = 4 \cdot 2 = 8 \quad (22)$$

$$dy = \frac{\partial J}{\partial y} = \frac{\partial v}{\partial y} \cdot \frac{\partial w}{\partial v} \cdot \frac{\partial J}{\partial w} = z \cdot 1 \cdot 2 = 2z = 2 \cdot 3 = 6 \quad (23)$$

$$dz = \frac{\partial J}{\partial z} = \frac{\partial v}{\partial z} \cdot \frac{\partial w}{\partial v} \cdot \frac{\partial J}{\partial w} = y \cdot 1 \cdot 2 = 2y = 2 \cdot 5 = 10 \quad (24)$$

4.1.5. Keras

Keras [42] este API-ul implicit care rulează peste Tensorflow (dar este compatibil și cu alte tipuri de backend-uri, CNTK sau Theano) și oferă un strat în plus de abstractizare. Dacă în Tensorflow trebuia să ne definim singuri, din punct de vedere matematic, funcția obiectiv, în Keras avem deja implementate cele mai utilizate tipuri de layere (Dense, Convolutional, Pooling), funcții de cost (MAE, MSE, Crossentropy), algoritmi de optimizare (SGD, RMSprop, Adam), funcții de activare (ReLU, Sigmoid, Softmax), elemente pe care le putem folosi prin doar câteva linii de cod. De asemenea, putem să ne creăm și propriile module prin extinderea unor clase abstracte și să le integrăm apoi în framework. Există două metode prin care se pot crea rețelele neuronale, în mod secvențial, prin instanțierea unei liste speciale în care adăugăm în ordine tipurile de straturi care alcătuiesc rețeaua și în mod funcțional, prin crearea componentelor și interconectarea lor în mod asemănător cu nodurile dintr-un graf. A doua variantă este cea pe care am folosit-o deoarece permite crearea de modele cu mai multe intrări, ieșiri și implementarea blocurilor de tip Skip-Connection asemănătoare cu cele din ResNet.

4.2. Resurse hardware

4.2.1. Google Colaboratory

Colab [43] este o platformă de tip IaaS (Infrastructure as a Service) ce oferă acces gratuit utilizatorilor la hardware specializat pentru antrenarea modelelor de Deep Learning. În momentul în care ne conectăm, platforma ne creează o mașină virtuală asociată contului de Gmail cu care suntem autentificați și timp de 12 ore, durata unei sesiuni, avem acces la resursele hardware puse la dispoziție. Mașinile virtuale pot avea 12 sau 25 GB de RAM, în funcție de cât de multe resurse are nevoie aplicația care rulează, CPU cu posibilitatea utilizării de hardware specializat (GPU sau TPU) și 108 (CPU, TPU) sau 68 (GPU) GB spațiu de stocare. Deși sistemul de operare ocupă 31 GB și reduce din spațiul disponibil, avem opțiunea să ne conectăm la Google Drive și să-l montăm ca partiție separată. Interfața platformei este intuitivă, de tipul Jupyter Notebook [44], în care putem să rulăm cod de Python în mod implicit sau putem apela funcții ale sistemului de operare (Linux) prin folosirea caracterului „!” înaintea comenzii.

În Tabel 4.1 am făcut o comparație asupra performanțelor celor 4 tipuri de GPU oferite de platformă (P4, K80, T4, P100), placa video a laptop-ului de pe care lucrez (GTX 1050Ti) și una dintre cele mai puternice unități de procesare grafică dezvoltate de Nvidia (V100). Rezultatele din tabel au fost determinate manual rulând comanda „nvidia-smi”.

Tabel 4.1 Caracteristici plăci video

| Nume produs | Compute Capability | Memorie (MiB) | Putere (W) | Performanță ⁶ FP32 (tFLOPS) | Preț ⁷ (\$) |
|---|--------------------|---------------|------------|--|------------------------|
| Nvidia GeForce GTX 1050Ti - Notebook | 6.1 | 4096 | 75 | - | 202 |
| Nvidia Tesla P4 | 6.1 | 7611 | 75 | 5.5 | 3342 |
| Nvidia Tesla K80 | 3.7 | 11441 | 149 | - | - |
| Nvidia Tesla T4 | 7.5 | 15079 | 70 | 8.1 | 4583 |
| Nvidia Tesla P100 (PCIe) | 6.0 | 16280 | 250 | 9.3 | 6570 |
| Nvidia Tesla V100 (PCIe) | 7.0 | 32480 | 250 | 14 | 9526 |

Compute Capability nu reprezintă performanța adevărată, ci este un scor atribuit plăcilor video CUDA în funcție de numărul de caracteristici hardware și software, adică cât de „moderne” sunt ca tehnologii folosite. Se poate observa în tabel că T4 are cel mai mare scor și este și cea mai eficientă energetic raportat la performanță. Deși GPU-urile sunt atribuite aleator, am încercat să folosesc numai P100 deoarece este cel mai performant accelerator hardware oferit de Colab, exceptând probabil TPU-ul care este încă în varianta Beta și greu de accesat. Diferența de preț și performanță între P100 și V100 este dată de numărul de nuclee de procesare CUDA (3584 vs 5120) și celor 640 de nuclee speciale Tensor Core ale lui V100.

Google Colab este destinat în mod special studenților și cercetătorilor care nu dispun de hardware suficient de puternic pentru a antrena local modele de DL sau care nu își permit să plătească pentru alte platforme mai performante (Google-GCE, Amazon-EC2).

4.2.2. Nvidia Jetson Nano

Nvidia Jetson Nano [45] este o plăcuță de dezvoltare asemănătoare cu Raspberry Pi-ul, dar care are integrat un procesor grafic suficient de puternic să ruleze în paralel și în timp rezonabil mai multe modele antrenate de DL, cu un consum energetic de doar 5W. Ca specificații tehnice, plăcuța are 128 de nuclee CUDA, 4GB de RAM, 4 porturi USB, port HDMI, pini GPIO ce pot fi folosiți în proiecte de IoT pentru controlul dispozitivelor electronice și port serial pentru conectarea unei camere de fotografiat/filmat.

În dezvoltările ulterioare ale proiectului o să folosesc această plăcuță într-un sistem de tip Master-Slave, unde nodurile (Slaves) o să ruleze algoritmul de identificare a echipamentului de protecție pe secvențele video înregistrate de cameră și, în funcție de rezultatele obținute la nivel local, o să transmită mai departe către un server central (Master) informații despre tipul problemei, unde a avut loc, cine a fost implicat și înregistrarea video a secvenței respective.

⁶ FP32: Single Precision Performance; <https://developer.nvidia.com/cuda-gpus>

⁷ https://www.insight.com/en_US/shop.html

4.3. Proiecte open-source

Dacă pentru problema clasificării obiectelor se pot folosi rețele neurale simple, cu un număr mic de straturi, ce pot fi implementate în câteva linii de cod în framework-urile moderne de inteligență artificială, identificarea obiectelor este o problemă foarte complexă din punct de vedere matematic. Au fost nevoie de cel puțin 5 ani de cercetări și experimente pentru a aduce modelul YOLO la performanțele de acum. Prin urmare, a implementa de la 0 propriul model pentru detecția obiectelor ar necesita cunoștințe foarte avansate în domeniu și foarte mult timp.

Unul din motivele pentru care domeniul inteligenței artificiale s-a dezvoltat atât de mult în ultimii ani este comunitatea foarte deschisă, atât pe plan academic, toate cercetările și descoperirile realizate sunt publicate în articole științifice la conferințe accesibile tuturor, cât și pe plan industrial, prin numărul foarte mare de proiecte open-source disponibile. La fel și în cazul modelului YOLO, autorii au făcut public codul sursă încă din prima zi.

Problemele cu implementarea originală a modelului YOLO [46] sunt următoarele: nu este documentată, este scrisă în limbajul C, un limbaj de programare „low-level”, prin urmare codul este foarte greu de urmărit, neavând o abstractizare care să ofere o imagine de ansamblu asupra structurii proiectului și, nu în ultimul rând, implementarea originală nu mai este actualizată de 2 ani, autorul principal renunțând la cercetarea în domeniul vederii artificiale.

Din fericire proiectul a fost preluat, noul repository [47] fiind foarte bine documentat, actualizat în mod constant și servește ultima versiune a modelului YOLO (versiunea 4). Darknet este numele framework-ului dezvoltat în C pe care l-am folosit în ultima parte a proiectului pentru a antrena 4 modele YOLOv3 și 4 modele de tip YOLOv4, pe care le-am transformat apoi în modele compatibile cu Tensorflow folosind un script de conversie. De asemenea, această implementare este optimizată să folosească atât memoria video și memoria RAM, iar modelele sunt făcute pentru a rula pe GPU-uri comerciale.

Pentru partea de proiecte open-source dezvoltate în framework-urile de Python nu am optat să le folosesc pe cele mai populare de pe GitHub, deoarece nu mai erau actualizate de foarte mult timp. Am pornit în schimb de la un proiect [48] scris în ultima versiune de Tensorflow (versiunea 2.0+) și care implementează „cele mai bune practici” în scrierea de cod. Problema cu acest proiect este că nu are maturitatea celorlalte și conține doar o implementare minimalistă a algoritmului. Pentru partea de antrenare, nu conține niciun mecanism de augmentare a datelor, nici cea mai optimă funcție de cost și nici multitudinea de metode BoF și BoS descrise în lucrarea modelului YOLOv4 și care cresc acuratețea modelului. Am încercat să modific proiectul și să îl actualizez în conformitate cu lucrările științifice și implementările originale și chiar am reușit să obțin niște îmbunătățiri destul de mari în rezultate, dar nu suficiente pentru a face modelele create viabile. Principala problemă este că în toate încercările de antrenare modele ajungeau în overfitting înainte de a atinge punctul de convergență, indiferent de metodele de augmentare și regularizare pe care le foloseam. Prin urmare, am folosit această implementare doar ca pe un schelet de cod care să mă ajute să îmi construiesc aplicația pentru partea de testare și integrare.

Pentru partea de urmărire a obiectelor [49] și recunoaștere facială [50] am folosit două modele pre-antrenate foarte populare și care se găsesc pe GitHub.

5. IMPLEMENTARE

5.1. Analiza datelor

După cum am prezentat și în capitolul 3.1, setul de date principal pe care l-am folosit în implementarea proiectului este cel cu imaginile generate virtual [33], la care se mai adaugă încă 3 seturi de date mai mici folosite pentru post-antrenare și testare. În cele ce urmează o să prezint o serie de informații statistice pentru seturile de date folosite, acestea fiind formate din:

- 132 GB de poze în format PNG la rezoluția 1088×612 , generate virtual în jocul GTA V. Pozele sunt împărțite în 126900 pentru antrenament și 13500 pentru validare și post-antrenare (fine-tuning). Datele pentru antrenament conțin 2778615 de obiecte pentru cele 7 tipuri de clase, distribuite astfel: 8.7% pentru clasa cap, 7.7% pentru clasa cască de protecție, 8.1% pentru clasa mască de sudură, 8.1% pentru clasa căști de protecție anti-zgomot, 16.8% pentru clasa piept fără protecție, 16.5% pentru clasa vestă reflectorizantă și 34% pentru clasa persoană. Datele pentru antrenament sunt generate în 27 de locații diferite din harta jocului și la momente diferite din zi și din noapte. Cele 13500 provin dintr-o distribuție statistică diferită, adică sunt generate în alte 3 locații diferite ale hărții și le-am împărțit în 3500 pentru validare și 10000 pentru fine-tuning.
- 220 de poze reale ce conțin aceleași 7 clase și pe care autorii setului de date virtual le-au folosit pentru fine-tuning și creșterea acurateții modelului. Aceștia demonstrează în lucrarea lor [30] că antrenarea pe un set de date virtual și post-antrenarea pe date reale duce la rezultate foarte bune.
- 1850 de poze de pe un șantier din România pe care le-am etichetat personal și pe care le-am folosit atât pentru post-antrenare, cât și pentru testarea modelelor.
- 17800 de poze colectate din mai multe seturi de date de pe internet, ce conțin etichete doar pentru cap și cască de protecție. Dintre acestea, 3174 de imagini conțin etichete distinctive pentru culorile căștilor de protecție. Aceste date le-am folosit pentru antrenarea unui model mai mic (YOLOv3-tiny) care are o acuratețe mai scăzută, dar o viteză de predicție mult mai mare (20-25 milisecunde pe un GPU Nvidia GTX 1050Ti).

În Figura 5.1 avem câteva imagini cu setul de date virtual, iar în Figura 5.2, prima poza este din setul de date real, etichetată de către autorii setului virtual, iar a doua este o poză de pe șantierul din România. Putem de asemenea să observăm că obiectele din setul de date virtual sunt mai mici comparativ cu datele reale și raportate la dimensiunea imaginii.

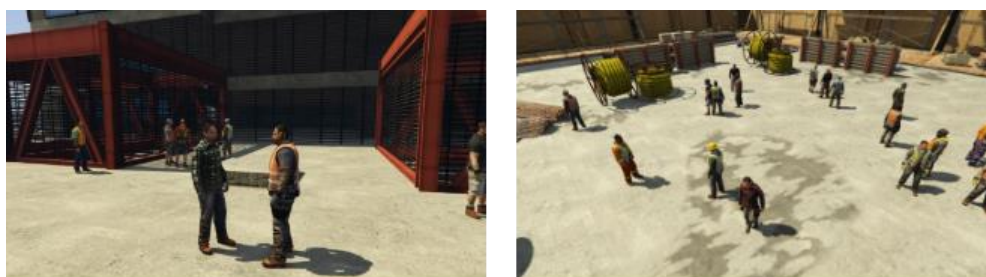


Figura 5.1 Setul de date virtual



Figura 5.2 Setul de date real

5.2. Prelucrarea datelor

5.2.1. Reducerea volumul setului de date

Primele operații de prelucrare pe care le-am folosit au fost pentru reducerea volumului datelor și a le face mai ușor de manipulat. Într-o primă etapă, am redimensionat toate pozele de la dimensiunea 1088×612 la 640×416 și le-am serializat împreună cu fișierele de adnotări în formatul Tensorflow record, format special folosit de Tensorflow pentru citirea rapidă a datelor în timpul procesului de antrenare. Am ales formatul 640×416 , deoarece este compatibil cu cerințele de funcționare ale arhitecturii YOLO (dimensiunea imaginilor să fie multiplu de 32) și păstrează o proporție relativ acceptabilă pentru dimensiunea obiectelor: $\frac{1088}{612} = \frac{16}{9} = 1.77$, $\frac{640}{416} = \frac{20}{13} = 1.53$. Dacă aș fi făcut imaginile pătrate, atunci toate obiectele ar fi fost disproporționate și foarte înguste. Astfel, din 120 de GB de poze doar pentru antrenament am redus volumul lor la 50 de GB (2.4 rata de reducere). Cu cât dimensiunea imaginilor este mai mică crește și viteza de antrenare pentru că numărul de parametri ce trebuie calculați este mai mic. Dacă aș fi lucrat cu imaginile la dimensiunea lor inițială, atunci timpul de antrenare ar fi crescut de cel puțin $\frac{1088 \cdot 612}{640 \cdot 416} = \frac{665856}{266240} = 2.5$ ori, deci pentru 7 zile de antrenare la rezoluție mică, ar fi trebuit cel puțin 17 zile de antrenare la rezoluția originală.

A doua metodă pe care am folosit-o pentru reducerea volumului datelor a fost schimbarea metodei de compresie de la format PNG (lossless), la format JPG (lossy). Reducerea calității pozelor nu constituie o problemă în procesul de antrenare, ci chiar un artificiu ce crește acuratețea modelelor. Dacă am face antrenarea doar pe imagini la rezoluție ridicată și de o calitate foarte bună, atunci când am testa în producție pe imagini care s-ar putea să fie de o calitate scăzută, algoritmul nu ar reuși să facă detecții. Prin această metodă de conversie am redus dimensiunea întregului set de date virtuale de la 132 de GB la doar 30.6 GB (4.3 rata de reducere), fără a fi nevoie să redimensionez imaginile.

5.2.2. Problema imaginilor aproape identice

Deși pozele sunt generate și etichetate automat de către un script, deci nu ar trebui să existe erori în ceea ce privește corectitudinea acestora, am identificat două tipuri de probleme în setul de date care pot afecta acuratețea modelelor.

Prima este aceea că foarte multe imagini sunt făcute din același unghi al camerei și sunt aproape identice. Acest lucru constituie o problemă, întrucât poate să ducă la fenomenul de overfitting, adică modelul se specializează foarte bine pe setul de antrenament și nu mai generalizează pe datele de validare și testare. După cum se observă și în Figura 5.3, cele două imagini sunt într-adevăr diferite, dar omul a rămas în aceeași poziție.



Figura 5.3 Imagini aproape identice

Problema apare din cauza faptului că personajele din jocurile video, cele care nu sunt controlate de către alți jucători, sunt destul de statice și au un set limitat de acțiuni pe care le pot îndeplini. În cazul acestui set de date, personajele au fost programate să se deplaseze doar într-un anumit perimetru. Acțiunea de mișcare este una liniară și este precedată și de o stare de repaos. În cazul personajului de mai sus, cel mai probabil acesta a rămas blocat în hartă din diferite motive și obstrucționează camera de luat vederi. Problema este că din cele 900 de poze realizate în această regiune a hărții, toate imaginile sunt identice cu cele din figură.

Pentru rezolvarea acestei probleme am folosit tehnici clasice folosite în augmentarea setului de date precum modificarea aleatorie a hue-ului (până la 10%), saturației (maxim 50%), expunere sau luminozitate (maxim 50%) și rotirea în plan orizontal a imaginilor cu modificarea în prealabil a etichetelor obiectelor. În ultimele modele dezvoltate am folosit și operația de blur și adăugare de zgomot gaussian, iar modelul YOLOv4 folosește inclusiv o tehnică geometrică de augmentare a datelor numită mozaic, care constă în decuparea aleatoare și combinarea a 4 imagini într-una singură.

5.2.3. Problema obiectelor suprapuse

A doua problemă pe care am identificat-o este legată de modul în care sunt etichetate obiectele. Script-ul folosit pentru generarea persoanelor etichetează toate obiectele dintr-un cadru, inclusiv dacă acestea nu sunt vizibile pe cameră. Acest lucru este o problemă deoarece se introduc erori de tipul false-pozitive (obiecte ce nu ar trebui etichetate) care pot afecta acuratețea modelelor. Analizând setul de date am observat că în 35% din imaginile folosite pentru antrenament există cel puțin două obiecte care se suprapun în proporție mai mare de 75% și 5.9% din numărul total de obiecte sunt acoperite cu același raport.

Pentru a detecta dacă două obiecte sunt suprapuse mi-am definit propria metrică pe care am numit-o indice de suprapunere (IS) și care are următoare formulă (25):

$$IS_{obj_1} = \frac{Aria(obj_1 \cap obj_2)}{Aria(obj_1)} \quad (25)$$

Cu cât valoarea indicelui de suprapunere este mai mare, cu atât obiectul este mai acoperit de obiectul cu care se suprapune, după cum se poate observa în Figura 5.4

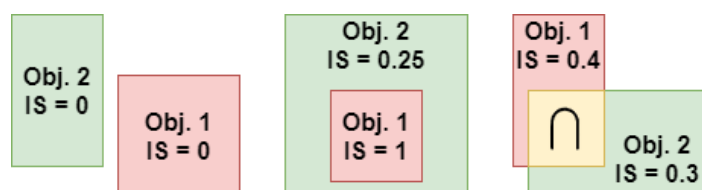


Figura 5.4 Indice de suprapunere

Pentru eliminarea etichetelor care introduc probleme, am pornit de la premiza că obiectele din prim plan au aria mai mare decât cele din fundal și toate clasele de obiecte din setul de date se suprapun cu clasa persoană. Dacă am etichetat un obiect din categoria cap sau un obiect din categoria corp, atunci sigur o să am o etichetă și pentru clasa persoană. Astfel, am definit următorul algoritm:

- 1) Pentru fiecare persoană din imagine calculez indicele de suprapunere cu toate celelalte persoane și, salvez într-o listă separată pe toți aceia care au un IS mai mare decât un prag de suprapunere (ps) $ps_{persoane}$ cu oricare altă persoană din imagine;
- 2) Aleg din listă persoana cu cel mai mare IS pentru a-l elimina;
- 3) Pentru fiecare obiect din celelalte două categorii de clase (cap și corp), calculez IS dintre obiectul curent și persoana selectată la punctul 2). Salvez în două liste separate pentru cap și corp obiectele care au un IS mai mare decât un prag ps_{cap_corp} ;
- 4) Pentru fiecare din cele două liste create anterior, aleg obiectele cu aria cea mai mică pentru a fi eliminate, fiindcă acestea au probabilitatea cea mai mare să se afle în spatele persoanei cu care se suprapun;
- 5) Elimin persoana, capul și corpul din lista de obiecte ale imaginii și reiau procesul de la capăt până când nu mai există obiecte de eliminat.

În practică, am observat că această regulă nu este general valabilă și depinde de mai multe aspecte precum orientarea camerei sau poziția oamenilor din imagine. Am ales valorile $ps_{persoane} = 0.9$ și $ps_{cap_corp} = 0.8$ pentru a evita riscul de a introduce erori de tipul false-negative, adică obiecte neetichetate ce ar trebui etichetate. De asemenea, pentru setul de validare am folosit valorile $ps_{persoane} = 0.75$ și $ps_{cap_corp} = 0.75$ pentru a elimina complet imaginile care pot genera erori și astfel să asigur că am o măsură cât mai corectă asupra acurateții modelului pe setul de validare. În Figura 5.5 se pot observa rezultatele înainte și după rularea algoritmului.



Figura 5.5 Eliminarea obiectelor suprapuse

5.2.4. Dimensionarea cadrelor de ancorare

După cum am prezentat și în capitolul de teorie, algoritmul YOLO folosește o metodă bazată pe cadre de ancorare pentru identificarea corectă a obiectelor suprapuse. Aceste cutii sunt predefinite și trebuie să încadreze cât mai corect geometria setului de date. Cadrele de ancorare folosite pentru modelul YOLOv3 și chiar YOLOv4 au fost concepute pentru setul de date MS COCO, care conține 80 de tipuri de clase cu dimensiuni și forme foarte variate, atât obiecte înalte (persoane, girafe), cât și obiecte late (mașini, autobuze, avioane). După cum am observat în figurile din secțiunile trecute, setul de date virtual conține doar oameni în poziție verticală și de relativ aceleași proporții și orientări. Prin urmare, ar fi mult mai eficient pentru algoritm să folosească numai cadre de ancorare înalte pentru persoane, cadre de ancorare pătrățite pentru obiectele din categoria cap și corp și să nu mai folosească deloc cutii late.

Pentru calcularea cadrelor de ancorare am folosit algoritmul K-means [51], care face parte din categoria metodelor de învățare nesupervizată și este folosit pentru identificarea de structuri în setul de date și împărțirea (clustering) acestora în K grupuri. Pentru a măsura cât de apropiate sunt elementele în setul de date se folosesc diferite funcții pentru calcularea distanței, exemplu distanța Euclidiană sau Manhattan. Astfel putem defini algoritmul K-means:

- 1) Se citesc toate datele în memorie și se amestecă;
- 2) Se selectează K centroide în mod aleator;
- 3) Cât timp valoarea curentă a centroidelor este diferită de cea precedentă, iterează:
 - a. Calculează distanța dintre fiecare element din setul de date cu fiecare centroid;
 - b. Se atribuie elementele către centroidul care este cel mai apropiat;
 - c. Se calculează pentru fiecare centroid media elementelor atribuite acestuia.

În Figura 5.6 avem reprezentarea grafică a algoritmului K-means pentru 3 centroide.

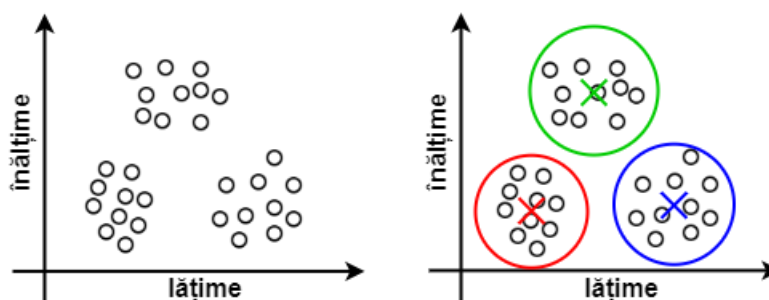


Figura 5.6 Algoritmul K-means

Centroidele reprezintă, pur și simplu, centrul unei grupări de elemente identificate în setul de date. Pentru cadrele de ancorare, datele sunt formate din două proprietăți, lățimea și înălțimea cutiei, iar centrele identificate reprezintă cea mai bună estimare a valorilor acestora.

În cazul detecției de obiecte, conform autorului principal al modelelor YOLOv1-3, în lucrarea [18], utilizarea distanței standard (Euclidiene), aplicată separat pentru înălțimea, respectiv lățimea cutiilor din setul de date, generează erori din cauză că nu ține cont de proporția obiectelor la scările de mărime în care apar. Prin urmare, o distanță mai bună pe care o putem folosi este indicele Jaccard care are forma (26) și se bazează pe IoU-ul dintre cutii și centroide:

$$d(box, centroid) = 1 - IoU(box, centroid) \quad (26)$$

Aplicând acest algoritm pe datele virtuale folosite la antrenament, am obținut următoarele valori ale cadrelor de ancorare, care se pot vedea la finalul lucrării, în Anexa A.

5.3. Procesul de antrenare

În inteligența artificială, precum și în ingineria sistemelor, nu există o formulă magică pe care trebuie să o urmărim pentru a obține cele mai precise rezultate. Deși modelele pe care le dezvoltăm sunt foarte bine definite din punct de vedere matematic și există un motiv întemeiat pentru care folosim anumite tehnici, arhitecturi și componente, de cele mai multe ori modelele sunt imprevizibile și se comportă ca un black-box⁸. În cadrul lucrării am iterat prin mai multe modele și am schimbat diverși hiper-parametri în cadrul procesului de antrenare pentru a observa comportamentul sistemului și a obține o acuratețe cât mai bună.

Am antrenat în total 7 modele de tip YOLOv3, un model de tip YOLOv3-tiny și 4 modele de tip YOLOv4. Modelul tiny a fost antrenat pe un set de date real, ce conține etichete doar pentru căștile de protecție și culorile acestora. Modele v3 și v4 au fost antrenate pe seturile de date cu 7 clase, cu o medie de 6-7 zile pentru setul de date virtual, aproximativ 9-10 ore pe zi (durata unei sesiuni) și o zi pentru post-antrenare pe datele reale. Am folosit 2 conturi de Gmail pentru a putea antrena în paralel câte două modele pe platforma Google Colab. În medie, durează aproximativ 56 de ore pentru antrenarea modelelor doar pe datele virtuale, la care se mai adaugă încă 7 ore pentru transferul datelor pe mașina virtuală și 8 ore pentru post-antrenare și testare, însumând un total de 71 de ore dedicate doar pentru un singur model și un total de aproximativ 1000 de ore pentru toate cele 11 modele YOLO. La acestea se mai pot adăuga încă cel puțin 200 ore de lucrat efectiv pentru scrierea codului aplicației, realizarea de diverse script-uri pentru automatizarea sarcinilor, înțelegerea codului implementărilor open-source și prelucrarea locală a datelor.

În Anexa A avem un exemplu de fișier de configurare folosit pentru antrenarea modelelor YOLOv3 și YOLOv4, atât în Tensorflow, cât și în Darknet, urmând să prezint aici o parte din cei mai importanți hiper-parametri pe care i-am ajustat.

⁸ black-box: un sistem necunoscut, care nu știm cum funcționează. Putem doar să îl stimulăm cu diverse intrări și să observăm rezultatele.

Pentru modelele antrenate în Tensorflow am folosit batch-uri de 16, deoarece modelul nu este optimizat să folosească și memoria RAM pentru stocarea datelor, ci folosește doar memoria video. În perioada respectivă, încă nu folosisesem metoda de conversie a pozelor de la format PNG la JPG pentru reducerea volumului, prin urmare datele pentru antrenare ocupau peste 50 GB și nu puteau fi încărcate toate pe mașina virtuală. Am împărțit setul de date în două părți și am antrenat 15 mini-epoci pe jumătate din poze și 15 mini-epoci pe cealaltă jumătate, însumând un total de 15 epoci complete. Tot pentru modele dezvoltate în Tensorflow, mi-am definit o funcție de callback⁹, care reduce în mod dinamic pasul de învățare atunci când modelul începe să fie divergent sau dacă intră în overfitting.

Pentru modele antrenate în Darknet, am antrenat pentru un număr de 24000 de batch-uri (aproximativ 12 epoci) și am scăzut pasul de învățare cu un factor de 0.1 atunci când modelul atinge 16000, respectiv 20000 de batch-uri. De asemenea, am folosit batch-uri de câte 64 de poze, deoarece implementarea este optimizată să folosească și memoria RAM. O tehnică interesantă de antrenament implementată în acest proiect se numește „burn in”, care schimbă pasul de învățare în primele $burn_in$ batch-uri după formula (27). $learning_rate_i$ este pasul de învățare la iterația i , $p = 4$ și $burn_in = 1000$.

$$learning_rate_i = learning_rate \cdot \left(\frac{i}{burn_in}\right)^p. \quad (27)$$

Pentru dimensiunea imaginii la intrarea în rețea am folosit mai multe tipuri de rezoluții dintre 640×416 , 416×416 , 608×608 sau 512×512 , primele 3 pentru arhitectura YOLOv3, iar ultima pentru modelul YOLOv4. Motivul pentru care, în versiunea 3 se folosesc valori care împărțite la 32 dau un număr par, este că astfel se generează o celulă de grid fix în centrul imaginii. YOLOv4 nu are probleme în a detecta obiectele centrate pe imagine, dar am preferat rezoluția 512×512 , deoarece este cea mai mare rezoluție pe care o puteam folosi în Google Colab, fără să se oprească programul din cauza lipsei de memorie.

Pentru algoritmul de optimizare și metodele standard de augmentare a datelor (hue, saturație, luminozitate / expunere, rotații) am folosit valorile implicite din Anexa A. Am ajustat în schimb valorile pentru 3 hiper-parametri de augmentare a datelor: blur, zgomot și augmentarea de tip mozaic. Un alt parametru interesant folosit este cel de *random_scale*, care redimensionează cu un procent de $\pm 40\%$ imaginile în ultimul strat al arhitecturii, pentru a permite detecția obiectelor la diferite scări de măsură (mici, medii, mari).

Tot pentru antrenare am folosit paradigma de învățare prin transfer de cunoștințe (Transfer Learning). Pornind de la modelul original, antrenat pe setul de date MS COCO, am înghețat primele 52 de straturi convoluționale din cele 75 ale arhitecturii YOLOv3, primele 72 de straturi convoluționale din cele 110 ale arhitecturii YOLOv4 și am modificat ultimele straturi astfel încât să fie compatibile cu dimensiunea vectorului de ieșire.

În Anexa D avem graficul procesului de antrenare pentru un model YOLOv4. Se poate observa că, până în jurul iterației 16000 graficul se aplatizează, iar după aceasta coboară sub pragul de 4. Acest lucru se datorează faptului că am redus pasul de învățare cu un factor de 0.1.

⁹ callback: funcție care se execută atunci când are loc un anumit eveniment.

Pentru partea de fine-tuning, am antrenat încă maxim 1500 de batch-uri pe diverse combinații dintre cele 10000 de imagini virtuale rămase, cele 220 de poze din setul de date real și pozele de pe șantierul din România. Am folosit aceleași fișiere de configurare cu cele ale modelelor pe care au fost pre-antrenate, în capitolul 6 urmând să prezint o analiză comparativă între rezultatele pe care l-am obținut prin ajustarea acestor hiper-parametri.

Între modele YOLOv3 și YOLOv4 diferă dimensiunile implicite ale cadrelor de ancorare, valorile pragurilor pentru IoU și scor de încredere, o serie de alți hiper-parametri pe care nu i-am prezentat, precum și configurația arhitecturii generale a sistemelor.

5.4. Probleme întâmpinate

5.4.1. Probleme de implementare

Deși problemele discutate în capitolul de prelucrare a datelor sunt mai interesante prin prisma soluțiilor pe care le-am găsit, erorile de implementare sunt cele care mi-au consumat cel mai mult timp în dezvoltarea proiectului. Dat fiind faptul că am pornit de la un proiect open-source [48], destul de nou, care nu are implementate toate tehnicile de optimizare și creștere a performanței din proiectul original [47], rezultatele pe care le-am obținut nu au fost tocmai cele mai bune. Am rezolvat o parte dintre aceste probleme analizând celelalte proiecte și consultând forum-urile de discuții de pe GitHub, precum și lucrările științifice ale autorilor. În final, am reușit chiar să obțin niște îmbunătățiri în performanțele modelelor, dar nu suficient cât să poată fi integrate într-o aplicație reală.

Principala problemă cu care am avut de-a face a fost aceea că toate modelele ajungeau în overfitting, înainte să își atingă punctul de minim. În Figura 5.7 avem reprezentarea costului J al modelului în funcție de numărul de epoci. Cu portocaliu este costul pe setul de antrenare, iar cu albastru este costul pe setul de validare. Se observă că începând cu epoca 10, costul de validare rămâne constant, în timp ce costul de antrenare continuă să scadă. Acest lucru este o problemă deoarece performanța reală a modelului nu se îmbunătățește în timp. Dacă am fi continuat antrenamentul pentru încă câteva epoci, am fi observat cum costul de validare ar fi început să crească, modelul ajungând în overfitting. Principala cauză pe care o intuiesc că a provocat acest fenomen este faptul că nu am folosit metode de augmentare a datelor suficient de puternice.

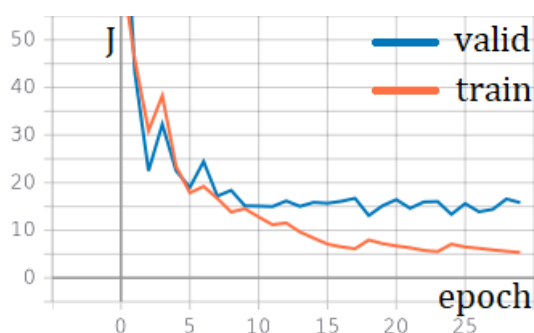


Figura 5.7 Graficul procesului de antrenare

5.4.2. Rezoluția imaginilor la antrenare

Setul meu de date este format din poze în format 16:9. Am vrut să păstrez o proporție relativ apropiată de cea originală pentru imagini după procesul de redimensionare, prin urmare am modificat implementarea algoritmului ca să fie compatibilă cu formatul meu (640×416). Din observațiile personale pe care le-am făcut, modelul nu este proiectat să lucreze cu imagini dreptunghiulare. În timpul procesului de antrenare, în momentul în care se face ajustarea gird-ului pentru cele 3 scări de măsură prezentate anterior, faptul că în plan orizontal avem mai multe căsuțe decât în plan vertical, îmi deformează imaginea și afectează modul în care sunt învățate trăsăturile. Pentru rezolvarea problemei, am citit în articolele autorilor faptul că ei folosesc o metodă de redimensionare numită *letter_box*, care păstrează proporțiile imaginii prin umplerea spațiului gol creat cu valori de 0 (padding). O astfel de redimensionare necesită și modificarea etichetelor în plan orizontal sau vertical, în funcție de direcția din care se face umplerea.

5.4.3. Ajustarea cadrelor de ancorare

O altă problemă pe care am întâmpinat-o legată de modul de funcționare al algoritmului, în implementarea originală a modelului YOLO, autorii folosesc 9 cadre de ancorare ce sunt specifice pentru geometria setului de date pe care a fost antrenat, în acest caz MS COCO. De asemenea, este menționat faptul că aceste cadre de ancorare pot fi modificate astfel încât să încadreze cât mai corect obiectele din setul de date pe care îl folosim. În cazul de față, în pozele generate virtual, obiectele au o formă dreptunghiulară cu înălțimea mai mare decât lățimea și o dimensiune destul de mică, prin urmare, nu voi avea nevoie de cadre de ancorare late folosite în modelul original. Pentru generarea acestora, am folosit un algoritm de grupare (K-means clustering), aplicat direct pe setul virtual. Problema este că nu am luat în considerare faptul că aceste cadre de ancorare trebuie să fie compatibile și cu imaginile reale pe care urmează să fac post-antrenarea și testarea modelului. Deoarece modelul învață trăsături în funcție de aceste dimensiuni, modificarea ulterioară a cutiilor o să anuleze valorile parametrilor inițiali pe care i-a învățat modelul. Astfel am obținut modele foarte precise pentru setul de date virtual, dar care nu reușesc să generalizeze pe cazul real din cauza diferenței de formă a obiectelor. În final am decis să folosesc cadrele de ancorare implicite, întrucât acestea au fost proiectate să acopere cât mai bine toate formele ce pot fi întâlnite într-un set de date și știm sigur că sunt funcționale.

5.5. Asamblarea componentelor

După cum am precizat și la începutul lucrării, scopul final al proiectului este crearea unui sistem de supraveghere și asistență, adică dezvoltarea unui produs software ce poate fi comercializat și integrat în cadrul unui șantier real de construcții. Până în acest punct am vorbit separat despre fiecare componentă software și hardware în parte. În acest capitol voi prezenta cum elementele de IA (detector, tracker și aplicația de recunoaștere facială) vor fi conectate, modul în care acestea vor funcționa și cum vor putea fi integrate mai departe pe o plăcuță de dezvoltare și incluse apoi în cadrul unei arhitecturi software complexe.

5.5.1. Workflow

Modul de conectare al elementelor este asemănător cu o arhitectură de tip pipeline¹⁰. Algoritmul de detecție o să fie aplicat pe cadrele din secvențele video înregistrate la un interval de t_1 secunde. Dacă avem obiecte identificate, atunci informațiile extrase de detector vor fi trimise mai departe către algoritmul de tracking și se vor atribui ID-uri pentru fiecare persoană din imagine. Dat fiind faptul că este mult mai ușor de identificat prezența echipamentului de protecție decât absența acestuia, dacă detectorul identifică persoane care nu poartă echipament de protecție atunci va porni un cronometru, care după t_2 secunde va emite o alarmă și, dacă nici atunci persoana nu își pune echipamentul de protecție, se va încerca identificarea facială a acesteia și camera va începe să înregistreze secvența respectivă. Dacă în schimb se detectează prezența echipamentului de protecție, atunci se poate trece la un alt caz de utilizare, de exemplu, se poate încerca identificarea culorii căștii de protecție a muncitorilor prin aplicarea modelului YOLOv3-tiny pe întreaga imagine și atribuirea corectă a căștilor către ID-ul persoanelor corespunzătoare, în funcție de indicele de suprapunere IoU. Culoarea căștilor se poate folosi pentru determinarea rolului persoanelor (galben pentru muncitor normal, alb pentru diriginte de șantier). Niște valori uzuale pentru t_1 și t_2 ar putea fi 10, respectiv 30 de secunde. Din acest punct nu există limitări în legătură cu scopul în care poate fi programat sistemul. În funcție de zona din șantier pe care o supraveghează camera, activitatea urmărită și cerințele clientului, se pot crea scenarii diferite de utilizare, asemănător cu cel descris anterior.

5.5.2. Integrare hardware

Din punct de vedere al performanței și scalabilității întregului sistem, este mult mai eficient ca identificarea și tot „workflow”-ul descris anterior să se realizeze la nivel local, deoarece într-o arhitectură de tip Client-Server intervine costul comunicațiilor. De asemenea, dat fiind faptul că sistemul urmează a fi integrat într-un șantier de construcții, prin urmare o să se afle într-o rețea închisă, server-ul pe care s-ar face procesarea ar trebui să fie destul de performant ca să poată realiza procesarea pentru toate camerele din sistem. După cum se observă și în Figura 5.8, în prototipul pe care îl dezvolt voi folosi plăcuța de dezvoltare NVIDIA Jetson Nano prezentată în lucrare, o cameră de Raspberry Pi compatibilă și un sistem de alarmare care poate să fie un buzzer conectat la pinii GPIO ai plăcuței. Plăcuța va avea, de asemenea, o bază de date locală cu trăsăturile fețelor muncitorilor de pe șantier.

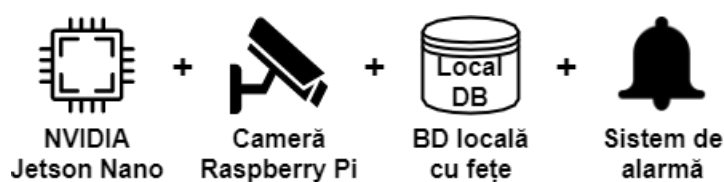


Figura 5.8 Integrarea hardware

¹⁰ pipeline: conectarea serială a unor componente, unde ieșirea unui element reprezintă intrarea pentru alt element.

5.5.3. Integrare software

În final, nodurile operaționale (plăcuțele de dezvoltare) o să realizeze identificarea echipamentului de protecție în diverse zone ale șantierului. Acestea vor putea transmite mai departe, către un server, informațiile sub formă de flux video pentru a fi salvate înregistrările, sau pentru a putea fi urmărite în timp real pe un calculator de către administratorul sistemului, prin intermediul unei interfețe grafice. În cazul în care se adaugă un nou muncitor în sistem prin intermediul interfeței de admin, trăsăturile feței persoanei adăugate vor fi calculate și distribuite mai departe către toate nodurile din rețea, ca în Figura 5.9.

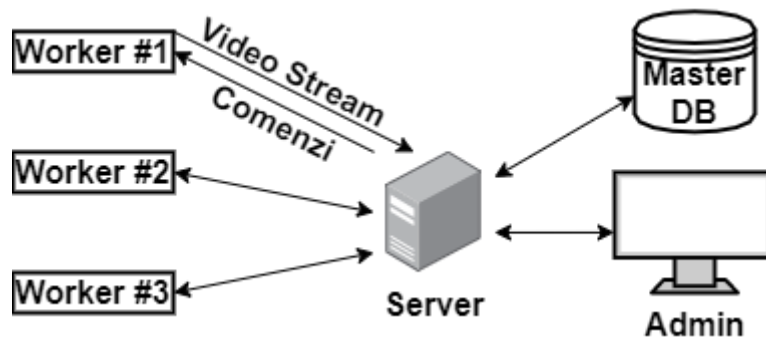


Figura 5.9 Arhitectura software a sistemului

6. REZULTATE OBȚINUTE

În acest capitol o să prezint o analiză comparativă între performanțele diverselor modele pe care le-am dezvoltat, precum și o serie de explicații / justificări legate de rezultatele obținute și problemele pe care acestea le prezintă.

În Tabel 6.1 putem să observăm rezultatele celor 8 modele pe care le-am antrenat în framework-ul Darknet. Am indexat fiecare model cu un ID, pentru a le face mai ușor de identificat în viitoarele tabele. Coloanele din stânga delimitatorului vertical reprezintă caracteristicile individuale ale modelelor, mai exact hiper-parametrii pe care i-am modificat în cadrul diverselor serii de antrenament. Deoarece autorii setului de date generat virtual [30] folosesc alte 350 de imagini pentru validarea modelului (notate V350), decât cele 3500 pe care le-am selectat prin filtrarea obiectelor suprapuse (notate V3500), am trecut în tabel, scorurile mAP obținute pentru ambele seturi de date. În acest capitol, mAP este definit ca media între AP-urile obținute pentru fiecare clasă de obiect la un IoU=0.5 și un scor de încredere de 0.25. Iar mIoU reprezintă media IoU-urilor dintre etichete și predicții pentru fiecare obiect în parte. De asemenea, tot în tabel avem trecută și iterația (It.) în număr de batch-uri la care am obținut cele mai bune rezultate, performanța modelelor fiind definită ca media armonică dintre mAP și mIoU, asemănător cu scorul F1. Indicii din dreptul iterației, (S) și (M), reprezintă tipul setului de date pe care am făcut validarea, V350, respectiv V3500. De menționat că valorile pentru Blur, Noise, mIoU și mAP sunt în procente (%), iar L-Box este parametrul *letter_box*.

Tabel 6.1 Rezultate validare pe seturile de date virtuale V350 (S) și V3500 (M)

| ID | Model | Size | L-Box | Blur | Noise | It. ^(S) | mIoU | mAP | It. ^(M) | mIoU | mAP |
|-----|--------|------|-------|------|-------|--------------------|------|-------------|--------------------|------|-------------|
| D#1 | YOLOv3 | 416 | NU | 0 | 0 | 22k | 75.1 | 91.8 | 22k | 75.2 | 95.4 |
| D#2 | YOLOv3 | 608 | NU | 0 | 0 | 24k | 78.5 | 96.4 | 21k | 77.7 | 97.7 |
| D#3 | YOLOv3 | 608 | DA | 0 | 0 | 24k | 79.3 | 95.8 | 24k | 78.9 | 96.1 |
| D#4 | YOLOv3 | 608 | DA | 25 | 25 | 24k | 77.6 | 94.8 | 24k | 77.2 | 95.9 |
| D#5 | YOLOv4 | 512 | NU | 0 | 0 | 24k | 78.9 | 95.1 | 22k | 78.3 | 97.6 |
| D#6 | YOLOv4 | 512 | NU | 25 | 25 | 19k | 78.1 | 95.0 | 21k | 77.1 | 97.7 |
| D#7 | YOLOv4 | 512 | DA | 25 | 25 | 24k | 77.3 | 95.2 | 20k | 77.9 | 98.2 |
| D#8 | YOLOv4 | 512 | DA | 50 | 50 | 21k | 77.1 | 95.4 | 20k | 77.4 | 97.9 |

Putem să observăm că scorurile mAP pentru V3500 sunt mai mari decât cele pentru V350, deoarece și setul de date este unul mai ușor, nemaivând obiecte suprapuse. Nu se observă diferențe majore între performanțele modelelor și nici măcar diferențe majore între versiunile de arhitecturi, pe setul de date V350, YOLOv3 reușind să obțină scoruri puțin mai bune. Acest lucru se datorează în primul rând dimensiunii imaginii la intrarea în rețea, dar și faptului că seturile de date folosite pentru validare sunt neconcludente, fiind foarte asemănătoare cu cele folosite la antrenament.

În Tabel 6.2 avem o comparație pe setul de date V350, între o parte din modelele anterioare cu cele mai bune rezultate, precum și modelul obținut de autori în lucrarea lor [30] și pe care l-am notat cu A#1. Ce este interesant de observat este că, pentru modelul D#1 am folosit aproximativ aceiași parametri de antrenare folosiți și de autori în A#1, dar cu toate acestea am obținut un scor mAP mult mai bun. Justificarea ar fi aceea că în etapa de prelucrare a datelor am eliminat obiectele suprapuse și am convertit imaginile la format JPG (reducerea calității imaginii echivalent cu augmentarea datelor). De asemenea, tot în tabel putem să observăm și rezultatele predicțiilor pentru diversele tipuri de clase și, se observă o dificultate mai ridicată în a identifica clasa mască de sudură (Mask).

Tabel 6.2 Rezultate validare pe setul de date virtual V350 (S)

| ID | It. | Head | Helmet | Mask | Headset | Chest | Vest | Person | mAP |
|-----|-----|------|--------|------|---------|-------|------|--------|-------------|
| A#1 | 21k | 89.7 | 86.7 | 75.5 | 89.0 | 89.7 | 90.0 | 89.7 | 87.2 |
| D#1 | 22k | 95.2 | 94.0 | 80.4 | 93.1 | 92.0 | 96.4 | 91.9 | 91.8 |
| D#2 | 24k | 97.7 | 97.8 | 94.5 | 98.3 | 95.9 | 97.4 | 93.6 | 96.4 |
| D#8 | 21k | 96.6 | 97.1 | 86.1 | 98.7 | 96.5 | 98.0 | 95.0 | 95.4 |

În Tabel 6.3 avem rezultatele modelelor antrenate pe setul de date virtual și testate pe imagini reale. După cum era de așteptat, scorurile sunt mai mici, însă problema este că niciun model din cele dezvoltate de mine nu depășește rezultatele autorilor. Motivul este că setul de date folosit de autori este diferit față de cel pe care am făcut testarea, în lucrare [30] este specificat faptul că folosesc 180 de poze pentru testare, dar în resursele pe care le pun la dispoziție doar 110 dintre imagini sunt marcate pentru testare. Se vede în schimb o diferență între performanțele arhitecturilor YOLOv3 și YOLOv4, rezultatele obținute la validarea pe date virtuale sunt necorelate cu cele obținute la testarea pe date reale. Toate modelele au dificultate în detectarea clasei Mask (în special YOLOv3), în schimb, putem observa că rezultatele pentru clasele Helmet și Vest sunt foarte bune, ceea ce ne confirmă ipoteza anterioară că sunt ușor de identificat datorită culorii și formei. De asemenea, avem rezultate bune și pentru clasa Person, argumentul fiind acela că am folosit paradigma Transfer Learning și am pornit de la un model pre-antrenat, ce conținea deja această clasă.

Tabel 6.3 Rezultate testare pe setul de date real (R)

| ID | It. | Head | Helmet | Mask | Headset | Chest | Vest | Person | mIoU | mAP |
|-----|-----|------|--------|------|---------|-------|------|--------|------|-------------|
| A#1 | 20k | 36.3 | 74.1 | 27.3 | 55.6 | 45.7 | 69.9 | 76.9 | - | 55.1 |
| D#1 | 16k | 9.7 | 48.1 | 0.7 | 51.1 | 25.0 | 65.2 | 60.9 | 42.2 | 37.2 |
| D#2 | 14k | 27.5 | 56.0 | 1.9 | 33.1 | 31.0 | 63.1 | 67.0 | 41.2 | 40.0 |
| D#3 | 24k | 17.7 | 58.3 | 6.9 | 33.6 | 28.5 | 51.2 | 57.2 | 41.2 | 36.3 |
| D#4 | 20k | 25 | 50.2 | 0.9 | 45.4 | 22.2 | 45.3 | 46.8 | 38.4 | 33.7 |

| | | | | | | | | | | |
|------------|-----|------|------|------|------|------|------|------|------|-------------|
| D#5 | 7k | 37.7 | 68.8 | 25.6 | 47.5 | 35.5 | 70.5 | 73.6 | 40.5 | 51.0 |
| D#6 | 12k | 42.5 | 58.2 | 16.1 | 69.9 | 36.8 | 60.3 | 74.6 | 40.1 | 51.2 |
| D#7 | 8k | 24.8 | 60.2 | 16.4 | 30.1 | 45.9 | 71.6 | 75.8 | 39.7 | 46.4 |
| D#8 | 9k | 25.0 | 68.7 | 15.3 | 49.7 | 41.4 | 72.9 | 74.4 | 38.3 | 49.6 |

Analizând rezultatele tabelelor anterioare, am decis să merg mai departe cu modelul D#2 pentru arhitectura YOLOv3 și cu modelul D#6 pentru arhitectura YOLOv4, deși chiar varianta D#5 ar fi fost una foarte bună. Dat fiind faptul că pentru partea de post-antrenare autorii pun la dispoziție doar 110 imagini, am considerat că strategia de antrenare folosită în D#6 este una mai bună pentru a preveni overfitting-ul. În ambele arhitecturi, parametrul *letter_box* nu a adus rezultate pozitive antrenării, iar în cazul arhitecturii YOLOv3, nici metodele de augmentare de tip blur sau zgomot nu aduc îmbunătățiri. Prin urmare, în etapa de post-antrenare (fine-tuning) am continuat antrenarea pentru încă 1500 de batch-uri pe setul de date real al autorilor și am obținut rezultatele din Tabel 6.4.

Tabel 6.4 Rezultate testare pe setul de date real (R) după post-antrenare

| ID | Head | Helmet | Mask | Headset | Chest | Vest | Person | mAP |
|------------|------|--------|------|---------|-------|------|--------|-------------|
| R#1 | 44.1 | 52.2 | 42.3 | 62.0 | 59.1 | 60.7 | 80.6 | 57.3 |
| A#1 | 78.8 | 73.3 | 66.3 | 74.0 | 74.7 | 78.6 | 87.1 | 76.1 |
| D#2 | 55.1 | 80.2 | 45.6 | 78.7 | 71.5 | 78.3 | 85.8 | 70.7 |
| D#6 | 80.3 | 82.5 | 63.4 | 92.5 | 82.3 | 84.0 | 89.3 | 82.0 |

R#1 este un model creat de autori antrenat exclusiv pe imaginile din setul de date real pe care îl vom folosi ca punct de referință. Putem observa că ambele modele pe care le-am dezvoltat au obținut rezultate foarte bune, depășind pragul R#1, iar modelul de tip YOLOv4 depășește inclusiv cel mai bun model al autorilor, A#1. Luând în considerare rezultatele din tabelele precedente, dar și faptul că autorii au folosit alte 180 de imagini pentru testare, diferite de cele 110 pe care le-au pus la dispoziție, putem trage concluzia că proiectul și-a atins obiectivele și înclină scorul final în favoarea mea (2-1, pentru rezultatele din cele 3 tabele cu comparații).

Pentru a vedea și niște rezultate mai practice, am atașat în Anexa F poze cu predicțiile modelelor dezvoltate de mine pe diferite imagini (interesante) din setul de test, iar în Anexa E avem predicțiile modelelor dezvoltate în Tensorflow, indexate cu T#, pentru a observa diferențele. Am folosit aceeași imagine de test pentru a avea un punct referință față de care să comparăm rezultatele. Problema cu modelele dezvoltate în Tensorflow este aceea că ajungeau în overfitting înainte de a converge către un minim. Cu toate acestea, în exemplele din anexă, modele reușesc să identifice ușor persoanele (datorită utilizării paradigmei de Transfer Learning și a modelului pre-antrenat ce conține deja clasa persoană), căștile de protecție (ușor de recunoscut datorită culorii și formei) și au acuratețe mai bună pentru obiectele de dimensiune mică (datele virtuale conțin obiecte mici).

7. CONCLUZII

Sumarizarea rezultatelor

Autorii setului de date virtual au demonstrat în lucrarea lor [30] că folosirea jocurilor video pentru generarea de imagini artificiale este o soluție viabilă, ce poate fi folosită în anumite probleme ale vederii artificiale pentru a compensa cu lipsa unui set de date real. Din rezultatele pe care le-am obținut în capitolul anterior nu pot confirma afirmația acestora, că antrenarea exclusiv pe un set de date virtual produce rezultate bune și modele viabile. Testând modelele pe imagini reale și secvențe video am reușit într-adevăr să obținem predicții corecte pe acestea, însă modelele obținute nu sunt stabile și dau foarte multe erori. Scorurile de încredere pentru obiectele identificate sunt mici (sub 90%) și foarte variabile. În schimb, pot afirma faptul că post-antrenarea pe un set de date mic (110 exemple) cu imagini reale, după ce modelele au fost pre-antrenate doar pe date virtuale, produce într-adevăr rezultate bune, reușind să obțină o îmbunătățire cu peste 30% a scorului mAP în simulări. De asemenea, testând modelele și în practică, se observă stabilitatea acestora, cu scoruri de încredere constante și de peste 97% pentru clasele ușor identificabile (cască, vestă, persoană), peste 95% pentru cap și piept și peste 90% pentru restul. Nu în ultimul rând, mi-am atins obiectivul principal al lucrării și am obținut un model cu 5.9% (scor mAP) mai bun decât cel al autorilor, utilizând tehnologii de ultimă generație.

Deși nu am avut cele mai bune rezultate încă de la început, nu le-am perceput niciodată ca pe un impediment. Așa cum a afirmat și Thomas Edison, „*nu am dat greș, ci am descoperit 10000 de idei care nu funcționează*”, și eu am dezvoltat modele care fie nu și-au atins obiectivul pe care mi l-am propus, fie nu mi-au satisfăcut exigențele personale pentru a le considera viabile și demne de implementat în producție. Important este că am învățat mai mult din greșeli, decât dacă aș fi reușit din prima încercare, iar în etapele ulterioare ale dezvoltării proiectului știu ce pași va trebui să urmez pentru a ajunge la performanța dorită. Inteligența artificială nu este în profunzime o știință exactă, nimeni nu știe formula magică pentru a obține cele mai bune / „perfecte” modele. Este în schimb un proces ingineresc, iterativ, care îți permite să greșești, să analizezi și să descoperi lucruri noi.

Moralitate în cercetare

Joseph Redmon [52], autorul principal al primelor 3 lucrări din seria YOLO, vorbește în ultimul său articol [19] despre problema moralității în domeniul cercetării și cum realizările și descoperirile oamenilor de știință pot fi folosite în alte scopuri decât cele pentru care au fost create. Într-o lucrare foarte atipică, lipsită de formalism și îmbibată cu foarte mult umor, autorul ne vorbește atât despre aspectele pozitive ale cercetării sale, cum oamenii folosesc detectoarele în scopuri nobile pentru „a număra zebrele dintr-o grădină zoologică” sau „a urmări pisica cum se plimbă prin casă”, dar și despre părțile negative, în care modelele pe care le-a dezvoltat au fost folosite în scopuri militare (rachete teleghidate) pentru a omorî oameni. Acesta a considerat că aplicațiile în care sunt folosiți acești algoritmi produc mai mult rău decât bine, prin urmare a decis să renunțe în a mai face cercetare în domeniul vederii artificiale. Autorul ne roagă pe

noi, utilizatorii, dar și pe cercetători, să ne ridicăm un semn de întrebare dacă proiectele pe care le dezvoltăm aduc cu adevărat un beneficiu real omenirii și, de asemenea, să luăm în calcul posibilele efectele negative și daune pe care le-ar putea provoca.

În calitate de entuziast al domeniului vederii artificiale și de utilizator al acestei tehnologii, consider că este datoria mea morală să îmi pun și eu această întrebare, dacă proiectul pe care l-am dezvoltat aduce cu adevărat un element de siguranță șantiierelor de construcții sau, dacă este doar o unealtă ce poate fi folosită și în alte scopuri decât cea pentru care a fost creată. Termenul „monitorizare” din titlul lucrării este sinonim cu „supraveghere”, iar oamenii asociază această supraveghere cu o încălcare a intimității. Desigur, șantierele de construcții nu reprezintă un spațiu public, muncitorii fiind nevoiți să semneze anumite documente legate de protecția muncii, sistemul creat de mine putând fi considerat o cerință parțială cu care vor trebui să fie de acord pentru a putea participa la activitățile de pe șantier. În cazul ideal, sistemul creat va salva într-adevăr vieți și va impune o conduită mai responsabilă a angajaților atunci când lucrează în medii periculoase. În cazul nefavorabil, sistemul ar putea fi folosit ca o unealtă de către directorii firmelor de construcții pentru a penaliza angajații de fiecare dată când fac o greșală, iar acest lucru se va reflecta asupra lor sub formă de stres și chiar ar putea să crească riscul de accidentare sau deces. Indiferent de scopul în care va fi folosit, părerile vor fi împărțite asupra moralității acestei aplicații prin simplul fapt că monitorizarea are foarte multe conotații negative și, știm foarte bine, că în anumite state această supraveghere a populației este realizată imoral, ilegal și împotriva voinței cetățenilor.

Ce urmează mai departe?

Încă de dinainte de a-mi alege tema lucrării de diplomă mi-am dorit să lucrez la un proiect care să aibă un impact real asupra societății, să nu fie doar o idee ce urmează a fi abandonată odată cu terminarea facultății. Imaginile de pe șantierul din România provin de la un ONG [\[53\]](#) în care am activat și care se ocupă cu construirea de case solare, sustenabile și eficiente energetic. Planul meu de viitor este să testez și să integrez prototipul pe care l-am dezvoltat în viitorul lor proiect, în speranța că îi va ajuta la capitolul siguranță în muncă atunci când vor construi noua casă și că le va aduce puncte bonus la capitolul inovație în concursul la care urmează să participe, Solar Decathlon Europe, în vara anului 2021. Nu putem anticipa scopurile în care vor fi utilizate aplicațiile noastre, putem doar să sperăm că, lăsându-le în mâini bune, vor avea un impact pozitiv asupra societății.

BIBLIOGRAFIE

- [1] „Global trends on occupational accidents and diseases,” International Labour Org., 2015. https://www.ilo.org/legacy/english/osh/en/story_content/external_files/fs_st_1-ILO_5_en.pdf. [Accesat Mai 2020].
- [2] „Health and safety at work - Summary statistics for Great Britain,” Health and Safety Executive, 2019. <https://www.hse.gov.uk/statistics/overall/hssh1819.pdf>. [Accesat Mai 2020].
- [3] „Accidents at work statistics - European Union,” Eurostat, 2017. <https://ec.europa.eu/eurostat/statistics-explained/pdfscache/11539.pdf>. [Accesat Mai 2020].
- [4] „Situatii accidente de muncă - România,” Inspectia Muncii, 2019. <https://www.inspectiamuncii.ro/statistici-accidente-de-munca>. [Accesat Mai 2020].
- [5] „Accidents at work - statistics by economic activity,” Eurostat, 2017. <https://ec.europa.eu/eurostat/statistics-explained/pdfscache/70752.pdf>. [Accesat Mai 2020].
- [6] B. Alexe, „Curs Inteligență Artificială,” Facultatea de Matematică și Informatică - Universitatea din București, 2019-2020.
- [7] A. Turing, „Computing Machinery and Intelligence,” în *Mind: A Quarterly Review of Psychology and Philosophy*, Oxford Academic Journals, 1950, pp. 433-460.
- [8] G. Press, „A Very Short History Of Artificial Intelligence (AI),” Forbes, 2016. <https://www.forbes.com/sites/gilpress/2016/12/30/a-very-short-history-of-artificial-intelligence-ai>. [Accesat Mai 2020].
- [9] Andrew Ng, „Machine Learning Course,” Coursera, 2011.
- [10] Andrew Ng, „Deep Learning Specialization,” Coursera, 2017.
- [11] S. Ruder, „An overview of gradient descent optimization algorithms,” în *arXiv preprint arXiv:1609.04747*, 2017.
- [12] S. Ruder, „An overview of gradient descent optimization algorithms,” 2016. <https://ruder.io/optimizing-gradient-descent>. [Accesat Iunie 2020].
- [13] K. Simonyan și A. Zisserman, „Very Deep Convolutional Networks for Large-Scale Image Recognition,” în *arXiv preprint arXiv:1409.1556*, 2014.
- [14] K. He, X. Zhang, S. Ren și J. Sun, „Deep Residual Learning for Image Recognition,” în *arXiv preprint arXiv:1512.03385*, 2015.

- [15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke și A. Rabinovich, „Going Deeper with Convolutions,” în *arXiv preprint arXiv:1409.4842*, 2014.
- [16] J. Redmon, „YOLO: Real-Time Object Detection,” <https://pjreddie.com/darknet/yolo>. [Accesat Iunie 2020].
- [17] J. Redmon, S. Divvala, R. Girshick și A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection,” în *arXiv preprint arXiv:1506.02640*, 2015.
- [18] J. Redmon și A. Farhadi, „YOLO9000: Better, Faster, Stronger,” în *arXiv preprint arXiv:1612.08242*, 2016.
- [19] J. Redmon și A. Farhadi, „YOLOv3: An Incremental Improvement,” în *arXiv preprint arXiv:1804.02767*, 2018.
- [20] A. Bochkovskiy, C.-Y. Wang și H.-Y. M. Liao, „YOLOv4: Optimal Speed and Accuracy of Object Detection,” în *arXiv preprint arXiv:2004.10934*, 2020.
- [21] S. R. Maiya, „DeepSORT: Deep Learning to Track Custom Objects in a Video,” Nanonets, 2019. <https://nanonets.com/blog/object-tracking-deepsort>. [Accesat Iunie 2020].
- [22] G. Ning, Z. Zhang, C. Huang, Z. He, X. Ren și H. Wang, „Spatially Supervised Recurrent Convolutional Neural,” în *arXiv preprint arXiv:1607.05781*, 2016.
- [23] N. Wojke, A. Bewley și D. Paulus, „Simple Online and Realtime Tracking with a Deep Association Metric,” în *arXiv preprint arXiv:1703.07402*, 2017.
- [24] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. V. Essen, A. A. S. Awwal și V. K. Asari, „A State-of-the-Art Survey on Deep Learning Theory and Architectures,” *Electronics*, vol. 8, nr. 10.3390/electronics8030292, p. 292, 2019.
- [25] Y. LeCun, C. Cortes și C. J. Burges, „THE MNIST DATABASE of handwritten digits,” National Institute of Standards and Technology, 1998. <http://yann.lecun.com/exdb/mnist>. [Accesat Iunie 2020].
- [26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li și L. Fei-Fei, „ImageNet: A Large-Scale Hierarchical Image Database,” 2009. <http://www.image-net.org>. [Accesat Iunie 2020].
- [27] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn și A. Zisserman, „The PASCAL Visual Object Classes (VOC) Challenge,” 2010. <http://host.robots.ox.ac.uk/pascal/VOC>. [Accesat Iunie 2020].
- [28] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick și P. Dollar, „Microsoft COCO: Common Objects in Context,” Microsoft, 2015. <http://cocodataset.org>. [Accesat Iunie 2020].

- [29] Kaggle, <https://www.kaggle.com>. [Accesat Iunie 2020].
- [30] M. D. Benedetto, E. Meloni, G. Amato, F. Falchi și C. Gennaro, „Learning Safety Equipment Detection using Virtual Worlds,” *2019 International Conference on Content-Based Multimedia Indexing (CBMI)*, nr. 10.1109/CBMI.2019.8877466, pp. 1-6, 2019.
- [31] J. Hui, „mAP (mean Average Precision) for Object Detection,” Medium, 2018. https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173. [Accesat Iunie 2020].
- [32] A. Bochkovskiy, „YOLOv4: The most accurate object detector on MS Coco dataset,” Medium, 2020. <https://medium.com/@alexeyab84/yolov4-the-most-accurate-real-time-neural-network-on-ms-coco-dataset-73adfd3602fe>. [Accesat Iunie 2020].
- [33] M. D. Benedetto, E. Meloni, G. Amato, F. Falchi și C. Gennaro, „Virtual World Personal Protection Equipment dataset (VW-PPE),” *Artificial Intelligence for Multimedia Information Retrieval (AIMIR)*, 2019. <http://aimir.isti.cnr.it/vw-ppe>. [Accesat Mai 2020].
- [34] Python, <https://www.python.org>. [Accesat Iunie 2020].
- [35] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes și J. Saraiva, „Energy Efficiency across Programming Languages: How does energy, time, and memory relate?,” Green Software Lab, 2017. <https://sites.google.com/view/energy-efficiency-languages/results>. [Accesat Iunie 2020].
- [36] Numba, <http://numba.pydata.org>. [Accesat Iunie 2020].
- [37] Anaconda, <https://www.anaconda.com>. [Accesat Iunie 2020].
- [38] NumPy, <https://numpy.org>. [Accesat Iunie 2020].
- [39] Matplotlib, <https://matplotlib.org>. [Accesat Iunie 2020].
- [40] OpenCV, <https://opencv.org>. [Accesat Iunie 2020].
- [41] Tensorflow, <https://www.tensorflow.org>. [Accesat Iunie 2020].
- [42] Keras, <https://keras.io>. [Accesat Iunie 2020].
- [43] Google Colaboratory, <https://colab.research.google.com>. [Accesat Iunie 2020].
- [44] Jupyter Notebook, <https://jupyter.org>. [Accesat Iunie 2020].
- [45] NVIDIA Jetson Nano, <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>. [Accesat Iunie 2020].
- [46] pjreddie, „Darknet,” GitHub, <https://github.com/pjreddie/darknet>. [Accesat Iunie 2020].
- [47] AlexeyAB, „Darknet: Yolo-v4 and Yolo-v3/v2 for Windows and Linux,” GitHub, <https://github.com/AlexeyAB/darknet>. [Accesat Iunie 2020].

- [48] zzh8829, „YoloV3 Implemented in TensorFlow 2.0,” GitHub, <https://github.com/zzh8829/yolov3-tf2>. [Accesat Iunie 2020].
- [49] nwojke, „Deep SORT,” GitHub, https://github.com/nwojke/deep_sort. [Accesat Iunie 2020].
- [50] ageitgey, „Face Recognition,” GitHub, https://github.com/ageitgey/face_recognition. [Accesat Iunie 2020].
- [51] I. Dabbura, „K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks,” towards data science, 2018. <https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a>. [Accesat Iunie 2020].
- [52] J. Redmon. <https://pjreddie.com>. [Accesat Iunie 2020].
- [53] EFdeN, <https://efden.org>. [Accesat Iunie 2020].

ANEXE

Anexa A. Fișier de configurare pentru antrenare

```
### INFORMATII GENERALE DESPRE STRUCTURA RETELEI
num_classes = 7 # cate clase de obiecte am in setul de date
width = 512 # am folosit urmatoarele combinatii: 416x416, 512x512, 608x608, 640x416
height = 512 # cu cat dimensiunea este mai mare, cu atat creste si acuratetea,
channels = 3 # dar scade viteza de predictie si creste timpul de antrenare.

### PARAMETRII PENTRU ANTRENARE
max_batches = 24000 # pentru Darknet
num_epochs = 15 # pentru Tensorflow
batch_size = 64 # am folosit 16 pentru Tensorflow si 64 pentru Darknet

### AJUSTAREA ACURATETII PENTRU NMS
iou_thresh = 0.5 # YOLOv3 - am folosit 0.5 pentru iou_thresh si conf_thresh
conf_thresh = 0.5 # YOLOv4 - am folosit valorile implicite recomandate de autori

### ALGORITM DE OPTIMIZARE
learning_rate = 0.001 # pasul de invatare initial
momentum = 0.9 # pentru primele 1000 de batch-uri se modifica adaptiv
decay = 0.0005 # la batch-urile 16000 si 20000 il scad cu un factor de 0.1

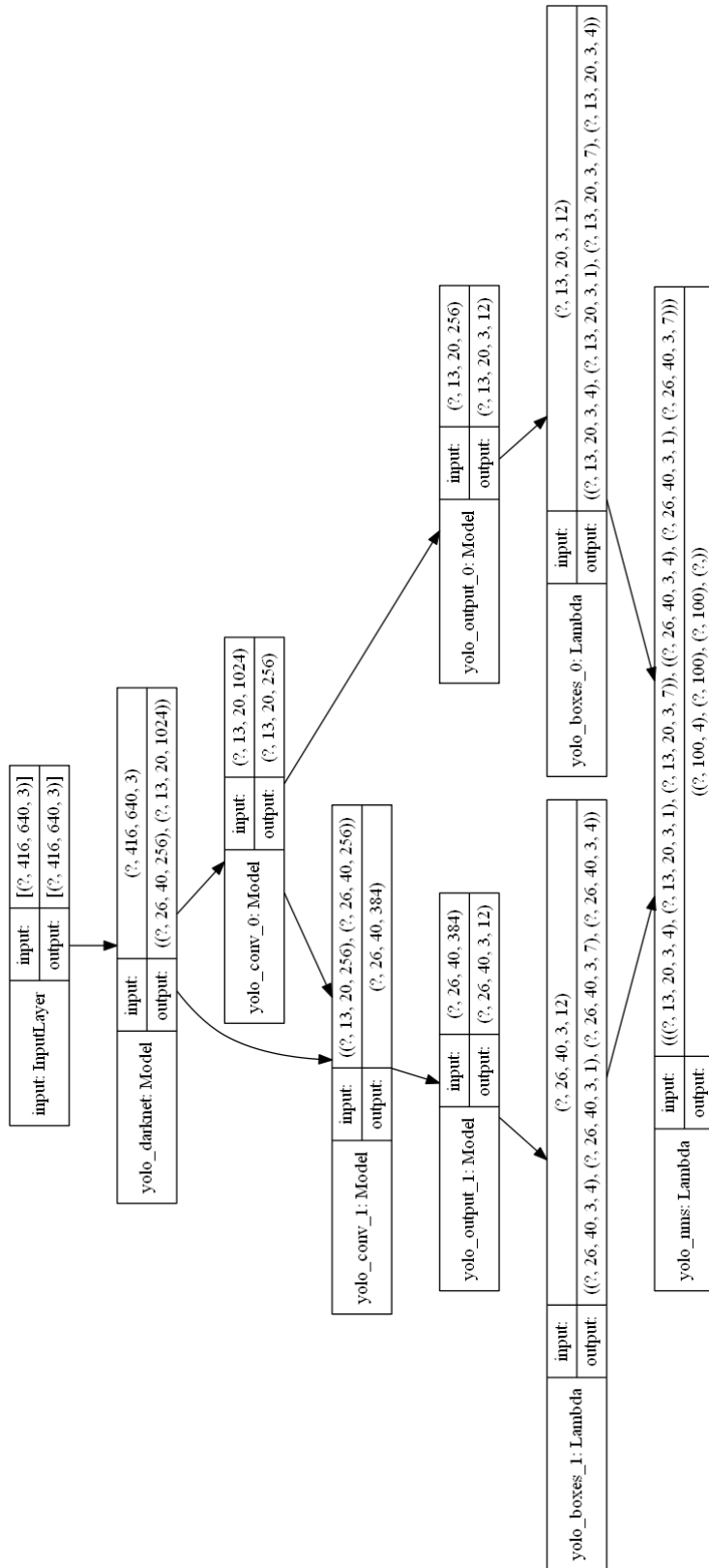
### AUGMENTARE FOTOMETRICA
hue = 0.1
saturation = 1.5
exposure = 1.5
blur = 8 # interval: 0-31; valori folosite: 0, 8, 16.
gaussian_noise = 32 # interval: 0-127; valori folosite: 0, 32, 64.

### AUGMENTARE GEOMETRICA
mosaic = 1 # imбина 4 imagini in una singura (folosit pentru YOLOv4)
flip = 1 # ogingeste imaginea in plan orizontal
random_scale = 1 # modifica dimensiunea imaginii aleator la fiecare 10 batch-uri
letter_box = 1 # 0 sau 1, pastreaza proportia imaginii si umple spatiul gol cu 0

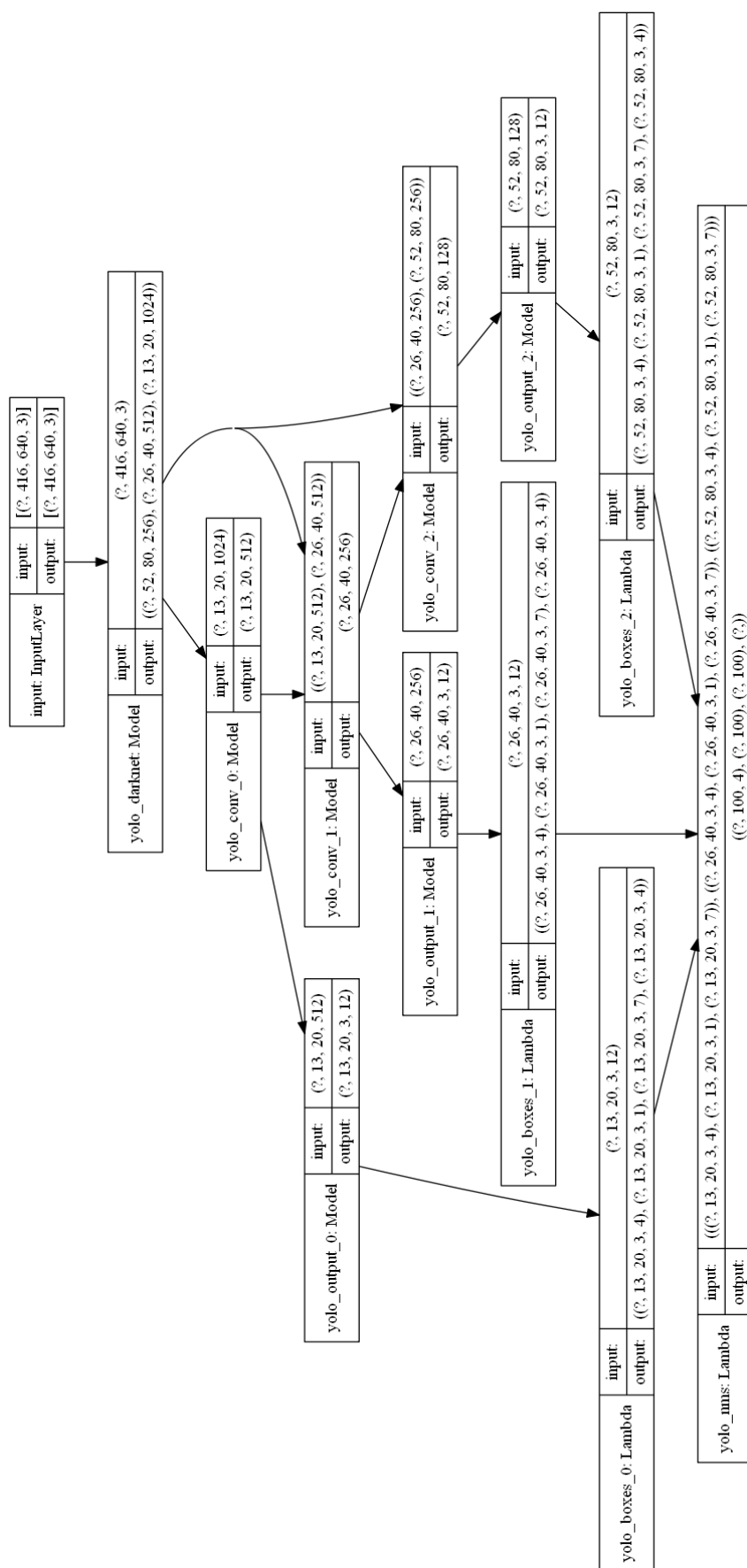
### DIMENSIONAREA CADRELOR DE ANCORARE
anchors = [[10,14], [23,27], [37,58], [81,82],
            [135,169], [344,319]] # YOLOv3-tiny original // size = 416x416
anchors = [[10,13], [16,30], [33,23], [30,61], [62,45], [59,119],
            [116,90], [156,198], [373,326]] # YOLOv3 original // size = 416x416
anchors = [[12,16], [19,36], [40,28], [36,75], [76,55], [72,146],
            [142,110], [192,243], [459,401]] # YOLOv4 original // size = 512x512
anchors = [[16,19], [24,27], [34,38], [39,87], [49,51], [60,122],
            [72,75], [100,153], [160,293]] # K-means generated // size = 416x416
```

Anexa B. Arhitecturi

B.1. Modelul YOLOv3-tiny



B.2. Modelul YOLOv3



Anexa C. Sumarul parametrilor

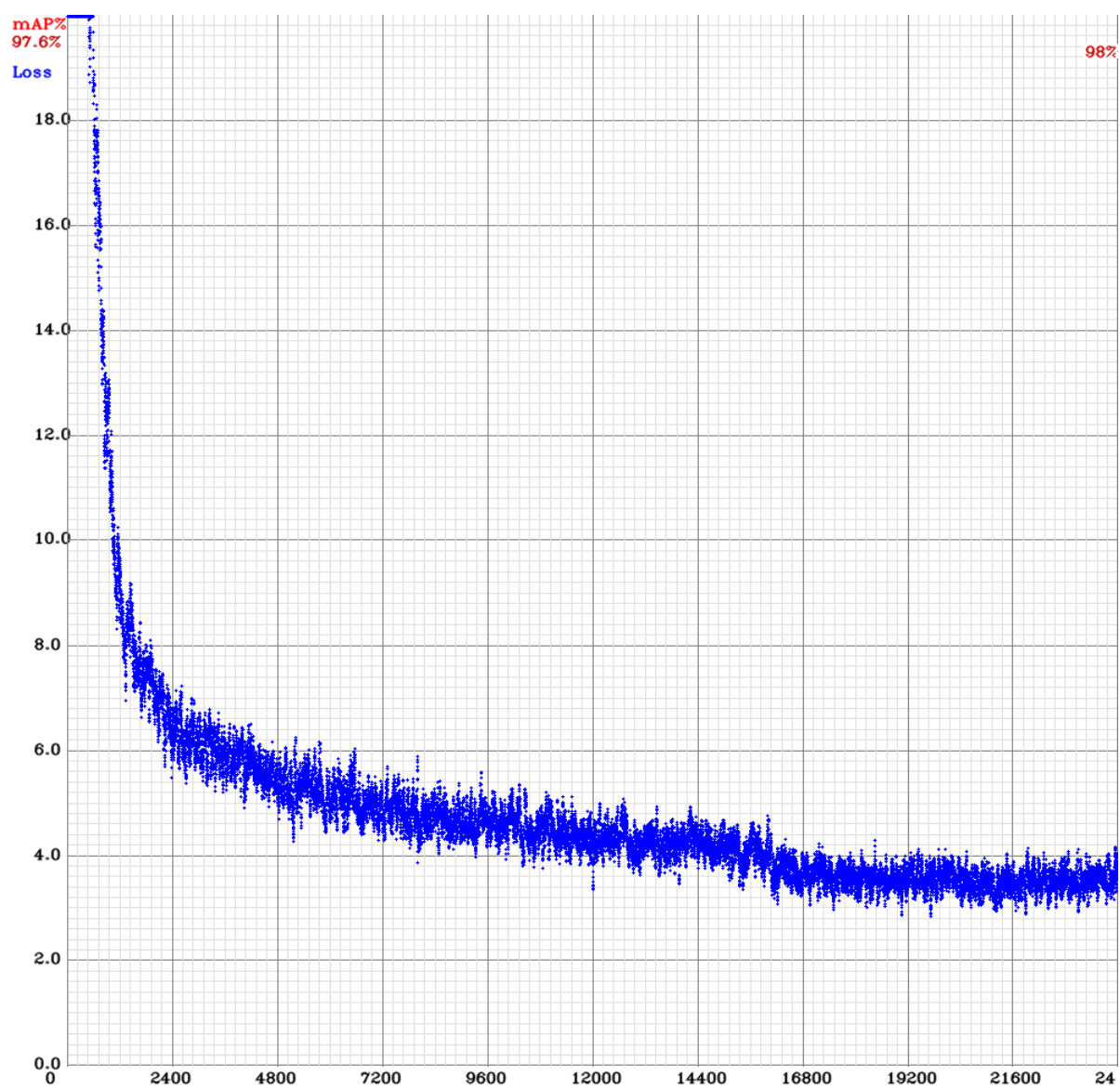
C.1. Sumar YOLOv3-tiny

| Model: "yolov3_tiny" | | | |
|-----------------------------|------------------------------|---------|--|
| Layer (type) | Output Shape | Param # | Connected to |
| input (InputLayer) | [(None, 416, 416, 3) 0 | | |
| yolo_darknet (Model) | ((None, 26, 26, 256) 6298480 | | input[0][0] |
| yolo_conv_0 (Model) | (None, 13, 13, 256) 263168 | | yolo_darknet[1][1] |
| yolo_conv_1 (Model) | (None, 26, 26, 384) 33280 | | yolo_conv_0[1][0] yolo_darknet[1][0] |
| yolo_output_0 (Model) | (None, 13, 13, 3, 12 1200164 | | yolo_conv_0[1][0] |
| yolo_output_1 (Model) | (None, 26, 26, 3, 12 895012 | | yolo_conv_1[1][0] |
| yolo_boxes_0 (Lambda) | ((None, 13, 13, 3, 4 0 | | yolo_output_0[1][0] |
| yolo_boxes_1 (Lambda) | ((None, 26, 26, 3, 4 0 | | yolo_output_1[1][0] |
| yolo_nms (Lambda) | ((None, 100, 4), (No 0 | | yolo_boxes_0[0][0] yolo_boxes_0[0][1] yolo_boxes_0[0][2] yolo_boxes_1[0][0] yolo_boxes_1[0][1] yolo_boxes_1[0][2] |
| Total params: 8,690,104 | | | |
| Trainable params: 8,683,736 | | | |
| Non-trainable params: 6,368 | | | |

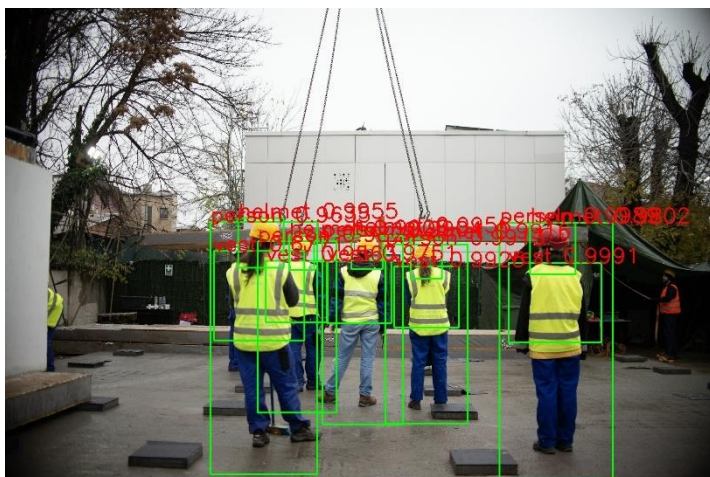
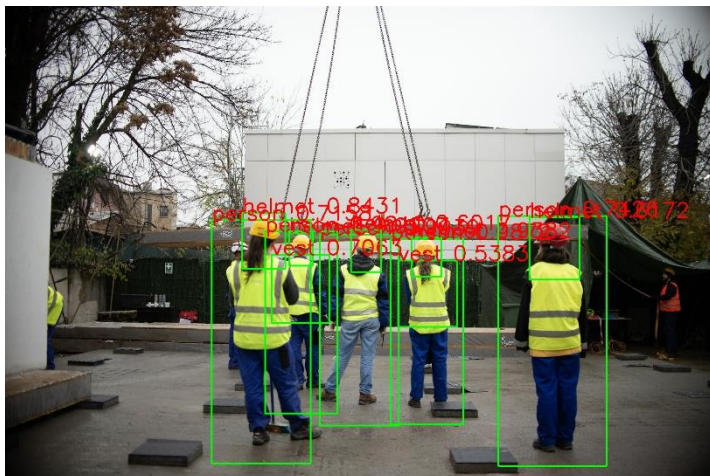
C.2. Sumar YOLOv3

| Model: "yolov3" | | | |
|------------------------------|-------------------------------|---------|--|
| Layer (type) | Output Shape | Param # | Connected to |
| input (InputLayer) | [(None, 416, 416, 3) 0 | | |
| yolo_darknet (Model) | ((None, 52, 52, 256) 40620640 | | input[0][0] |
| yolo_conv_0 (Model) | (None, 13, 13, 512) 11024384 | | yolo_darknet[1][2] |
| yolo_conv_1 (Model) | (None, 26, 26, 256) 2957312 | | yolo_conv_0[1][0] yolo_darknet[1][1] |
| yolo_conv_2 (Model) | (None, 52, 52, 128) 741376 | | yolo_conv_1[1][0] yolo_darknet[1][0] |
| yolo_output_0 (Model) | (None, 13, 13, 3, 12 4759588 | | yolo_conv_0[1][0] |
| yolo_output_1 (Model) | (None, 26, 26, 3, 12 1200164 | | yolo_conv_1[1][0] |
| yolo_output_2 (Model) | (None, 52, 52, 3, 12 305188 | | yolo_conv_2[1][0] |
| yolo_boxes_0 (Lambda) | ((None, 13, 13, 3, 4 0 | | yolo_output_0[1][0] |
| yolo_boxes_1 (Lambda) | ((None, 26, 26, 3, 4 0 | | yolo_output_1[1][0] |
| yolo_boxes_2 (Lambda) | ((None, 52, 52, 3, 4 0 | | yolo_output_2[1][0] |
| yolo_nms (Lambda) | ((None, 100, 4), (No 0 | | yolo_boxes_0[0][0] yolo_boxes_0[0][1] yolo_boxes_0[0][2] yolo_boxes_1[0][0] yolo_boxes_1[0][1] yolo_boxes_1[0][2] yolo_boxes_2[0][0] yolo_boxes_2[0][1] yolo_boxes_2[0][2] |
| Total params: 61,608,652 | | | |
| Trainable params: 61,556,044 | | | |
| Non-trainable params: 52,608 | | | |

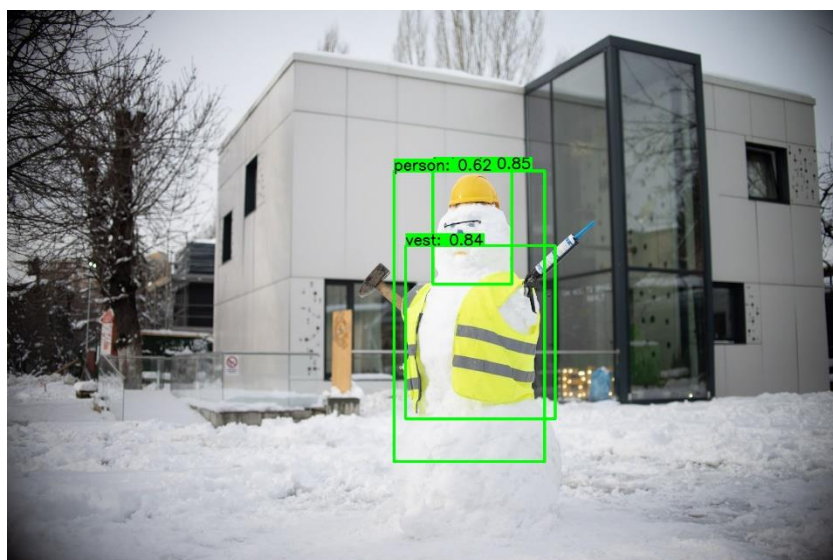
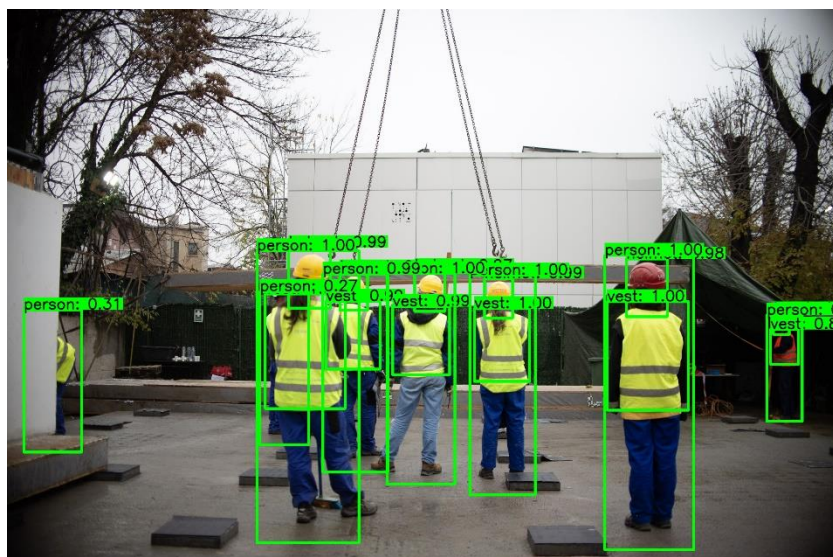
Anexa D. Graficul procesului de antrenare - YOLOv4



Anexa E. Rezultatele modelelor antrenate în Tensorflow



Anexa F. Detecții pe imagini diverse



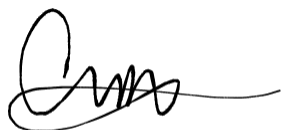
Anexa G. Declarație de onestitate academică

Prin prezenta, declar că lucrarea cu titlul *Sistem suport dedicat monitorizării siguranței muncii pe un șantier*, prezentată în cadrul Facultății de Automatică și Calculatoare, Universitatea POLITEHNICA din București, ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul Ingineria Sistemelor, programul de studiu Automatică și Informatică Aplicată, este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe internet, sunt indicate în lucrare ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie o infracțiune și se sancționează conform legilor în vigoare. Declar că toate rezultatele simulărilor, experimentelor și măsurărilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, Iunie 2020.

Absolvent: Andrei-Cosmin MARIA



.....