

UNIVERSITATEA DIN BUCUREŞTI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENȚĂ

SISTEM DE ESTIMARE A DISTANȚEI ÎNTRE MAȘINI

Coordonator:

Conf. Dr. Alexe Bogdan

Absolvent:

Iordache Adrian Răzvan

BUCUREŞTI, IULIE 2020

Rezumat

Estimarea distanțelor dintre mașini bazată pe intrări vizuale este una dintre cele mai importante probleme în conducerea asistată. Această problemă reprezintă unul dintre numeroșii pași pe care trebuie să îi facem pentru dezvoltarea unui sistem care poate lua decizii de conducere sau pentru monitorizarea șoferului.

În această teză, analizăm problema propunând o soluție mono-vizuală care utilizează o abordare de detectare a obiectelor pentru a detecta mai întâi toate mașinile din imaginea de intrare, urmată de identificarea mașinii care se află în fața camerei noastre.

În prima parte a acestei teze, Capitolul 2, vă prezintăm câteva concepe avansate de vedere artificială, care ne vor permite să descriem ulterior arhitecturi de rețele neuronale convoluționale utilizate ca parte fundamentală a soluției noastre.

În capitolul 3 vom rezuma tehnologiile utilizate la dezvoltarea proiectului.

Capitolul 4 descrie în detaliu soluția propusă, care constă în două arhitecturi de rețele neuronale.

Prima rețea neuronală este utilizată pentru detectarea tuturor mașinilor din imaginea de intrare și, de asemenea, pentru cuantificarea distanțelor dintre poziția camerei și mașinile detectate.

A doua rețea neuronală are ca intrări detectiile de la rețeaua de detectare a obiectelor și o caracteristică suplimentară pentru a rezolva problema detectării mașinii în fața vehiculului nostru.

Experimentele și observațiile sunt prezentate în detaliu în acest capitol.

În capitolul 5 concluzionăm soluția noastră, prezentând rezultatele finale pentru sistemul de estimare a distanțelor în situații reale.

Abstract

Estimating distances between cars based on visual input is one of the most important problems in assisted driving. This problem represents one of the many steps we need to take for developing a system that can take driving decisions or for driver monitoring.

In this thesis, we tackle the problem by proposing a mono-visual solution that uses an object detection approach to first detect all the cars in the input image, followed by identifying the car which is in front of our camera.

In the first part of this thesis, Chapter 2, we present some advanced computer vision concepts which will allow us to later describe convolutional neural network architectures used as a fundamental part of our solution.

In Chapter 3 we summarize the technologies used in developing the project.

Chapter 4 describes in detail the solution proposed, which consists of two deep neural network architectures.

The first neural network is used for detecting all the cars in the input image and also for quantifying the distances between the camera position and the detected cars.

The second neural network takes as input the detections from the object detection network and one additional feature to solve the problem of detecting the car in the front of our vehicle.

Experiments and observations are presented in detail in this chapter.

In Chapter 5 we conclude our solution, presenting the final results for estimating distances system in real-life situations.

Cuprins

Lista de figuri	6
Lista de tabele	9
1 Introducere	10
1.1 Prezentarea contextului	10
1.2 Descrierea problemei	11
1.3 Repere istorice și contribuție proprie	14
1.4 Structura lucrării	16
2 Fundamentare Teoretică	17
2.1 Convoluții Separabile Depthwise	17
2.2 Arhitectura MobileNetV1	23
2.3 Nivele Reziduale Inversate și Bottleneck-uri Liniare	27
2.3.1 Nivele Reziduale Inversate	27
2.3.2 Bottleneck-uri Liniare	30
2.4 MobileNetV2	31
2.5 Mean Average Precision (mAP)	33
2.6 SSD: Single-Shot MultiBox Detector	37
3 Tehnologii Aplicate	44
3.1 Dezvoltarea rețelelor neuronale	44
3.1.1 Tensorflow Research	44

3.1.2	Pytorch	44
3.2	Adnotarea datelor	45
3.3	Manipularea datelor	45
3.3.1	Scikit-learn, OpenCV, Pandas, Numpy	45
3.4	Vizualizarea datelor și a rezultatelor	46
4	Descriere Soluție	47
4.1	Setul de date	48
4.2	Antrenarea arhitecturii pentru detectare de obiecte	54
4.2.1	Rezultate SSD-MobileNetV1 pentru detectare de vehicule	54
4.2.2	Rezultate Inferență Validare SSD-MobileNetV1	56
4.2.3	Rezultate SSD-MobileNetV2 pentru detectare de vehicule	58
4.2.4	Rezultate Inferență Validare SSD-MobileNetV2	60
4.3	Problema de calibrare a detectiilor	61
5	Rezultate Finale și Concluzii	66
5.1	Rezultate corecte ale sistemului	66
5.2	Rezultate eronate ale sistemului de estimare a distanței	69
Referințe		71

Lista de figuri

1.1	Exemple de cazuri favorabile	12
1.2	Exemple de cazuri nefavorabile	12
1.3	Exemple de cazuri pe timp de noapte	13
1.4	Tipuri de numere de înmatriculare	15
1.5	Tipuri de spate de mașini	15
2.1	Convoluția Standard	18
2.2	Convoluția Depthwise: Faza de filtrare	19
2.3	Convoluția Pointwise: Faza de combinare	20
2.4	Arhitectura unui layer convolutional standard cu normalizare și ReLU non-liniaritate prezentată comparativ cu arhitectura unui layer de convoluții separabile prezente în MobileNetV1 [1]	25
2.5	Exemplu de conexiune reziduală între nivele de convoluție	28
2.6	Exemplu de conexiune reziduală inversată între nivele de convoluție	29
2.7	Comparație între activare liniară și non-liniară în ultimul nivel [2]	30
2.8	Rezultate între diverse tipuri de legături reziduale [2]	30
2.9	Ilustrare Precizie și Recall [3]	34
2.10	Ilustrare Coeficientul Iou [4]	34
2.11	Curba de precize și recall comparativ Curbei de precizie interpolată și recall	36
2.12	Arhitectura Modelului VGG-16	38
2.13	Arhitectura parțială a modelului Single-Shot Detector Modificată din articol [5]	38

2.14 Arhitectura Single-Shot Detector [5]	39
4.1 Distribuția Claselor pe setul de date	49
4.2 Distribuția Claselor pe setul de Train	52
4.3 Distribuția Claselor pe setul de Validare	53
4.4 Distribuția Claselor pe setul de Test	53
4.5 Variații de mAP pentru SSD-MobileNetV1	54
4.6 Funcțiile de pierdere pentru clasificare și localizare la SSD-MobileNetV1	55
4.7 Funcțiile de pierdere totală la SSD-MobileNetV1	56
4.8 Inferență Validare SSD-MobileNetV1 - Imaginea 1	56
4.9 Inferență Validare SSD-MobileNetV1 - Imaginea 2	57
4.10 Inferență Validare SSD-MobileNetV1 - Imaginea 3	57
4.11 Inferență Validare SSD-MobileNetV1 - Imaginea 4	57
4.12 Variații de mAP pentru SSD-MobileNetV2	58
4.13 Funcțiile de pierdere pentru clasificare și localizare la SSD-MobileNetV2	59
4.14 Funcțiile de pierdere totală la SSD-MobileNetV2	59
4.15 Inferență Validare SSD-MobileNetV2 - Imaginea 1	60
4.16 Inferență Validare SSD-MobileNet2 - Imaginea 2	60
4.17 Inferență Validare SSD-MobileNetV2 - Imaginea 3	60
4.18 Inferență Validare SSD-MobileNetV2 - Imaginea 4	61
4.19 Schița descriptivă a problemei de calibrare	62
4.20 Distribuția de etichete pentru problema de calibrare	63
4.21 Distribuția de offset-uri pentru problema de calibrare	63
4.22 Plotarea centrelor ferestrelor pentru detectii la offset 0	64
4.23 Acuratețea pentru rețeaua neuronală de calibrare	65
4.24 Funcțiile de pierdere pentru rețeaua neuronală de calibrare	65
5.1 Rezultate Finale - Imaginea 1	66

5.2	Rezultate Finale - Imaginea 2	67
5.3	Rezultate Finale - Imaginea 3	67
5.4	Rezultate Finale - Imaginea 4	68
5.5	Rezultate Finale - Imaginea 5	68
5.6	Rezultate Finale - Imaginea 6 (Eroare de detectie)	69
5.7	Rezultate Finale - Imaginea 7 (Eroare de calibrare)	70

Lista de tabele

2.1	Arhitectura Rețelei Convolutionale MobileNetV1 [1]	24
2.2	Resurse pentru fiecare tip de nivel [1]	25
2.3	Comparație între MobileNetV1 și modele populare [1]	26
2.4	Comparație între MobileNetV1 și alte modele de dimensiuni reduse [1]	26
2.5	Rezultate MobileNetV1 în sisteme pentru detectare de obiecte [1]	27
2.6	Exemplu Bloc Rezidual de tip Bottleneck, cu factor de expansiune t [2]	31
2.7	Arhitectura lui MobileNetV2 [2]	32
2.8	Rezultate pe setul de date ImageNet, comparație între diferite arhitecturi [2]	33
2.9	Determinarea categoriilor detecției	35
2.10	Rezultate Exemplu Precizie, Recall, Precizie Interpolată	35
2.11	Pascal VOC2007 rezultate setul de test [5]	42
2.12	Pascal VOC2012 rezultate setul de test [5]	42
2.13	Rezultate pe setul de date Pascal VOC2007 test pentru timpul de inferenta	43

Capitolul 1

Introducere

1.1 Prezentarea contextului

Problema estimării distanței între mașini în contextul acestei lucrări a apărut din necesitatea dezvoltării unui dispozitiv încorporat (embedded) capabil să evaluateze performanța și eficiența șoferilor din cadrul flotelor de mașini din diverse industrii.

Dispozitivul embedded menționat anterior, pe lângă specificațiile de bază precum: camera interioară și exterioară cu înregistrare continuă, localizator GPS, transmitere a datelor prin Bluetooth, 3G și Wi-Fi, slot de card SD pentru stocarea locală a datelor, mai conține și un accelerator grafic (GPU) utilizat pentru aplicații de vedere artificială (Computer Vision).

Printre aplicațiile de vedere artificială dezvoltate pe acest dispozitiv se află atât estimarea distanței între mașini și detectarea mașinii din față, cât și detectare stării de oboseală a șoferilor, detectarea stării de neatenție, detectarea centurii de siguranță și recunoașterea facială a șoferilor.

Aceste detalii sunt necesare de enunțat deoarece aplicațiile vor concura pentru resursele oferite acceleratorului grafic, aspect important în alegerea metodologiei de inferență.

1.2 Descrierea problemei

În cadrul acestei lucrări, problema abordată, estimarea distanței între mașini bazată pe input vizual, prezintă un număr ridicat de restricții și dificultăți pentru a putea genera o soluție viabilă și scalabilă din punct de vedere industrial.

Pentru a căpăta o mai bună înțelegere asupra problemei prezentate este necesară înțelegerea restricțiilor întâlnite frecvent în lumea reală, urmată de înțelegerea limitărilor tehnice prezente în situația curentă.

Luând în considerare contextul amplu și plin de posibilități în care soluția urmează a fi abordată, am decis ca eroarea de precizie a algoritmului de estimare a distanței să fie una mai permisivă în favoarea unei mai bune generalizări a situațiilor posibile.

Aceste cerințe fiind fixate, am ales ca opțiune posibilă discretizarea intervalului continuu în categorii de distanță, problema devenind una de detectare a vehiculelor din imagine dorind păstrarea noțiunii de regresie prezentă în problema initială.

Într-un context ideal, lipsit de limitări hardware, soluția abordată necesită doar găsirea unui algoritm pentru detectarea vehiculelor aflate în fața noastră, iar timpul de inferență nu ar reprezenta un factor puternic decizional.

Principala problemă în această situație constă în găsirea unei metodologii capabile să reprezinte matematic fiecare imagine de intrare și să generalizeze toate posibilele clase ale detecțiilor, respectiv în cazul nostru toate tipurile, formele și culorile vehiculelor.

În același timp dorim ca algoritmul să fie invariant la complexitatea spațiului de situații posibile al lumii reale precum factori externi de lumină, precipitații sau alte condiții.

Pentru a înțelege dificultatea acestei probleme vom prezenta câteva perechi de imagini de input însotite de outputul dorit al unui posibil algoritm de estimare a distanței pentru diverse scenarii probabile.

Pentru început vom lua în calcul cazurile favorabile unde numărul de vehicule din imagine este variabil, dar nu avem condiții meteorologice dificile precum: zăpadă, ceată sau precipitații puternice.

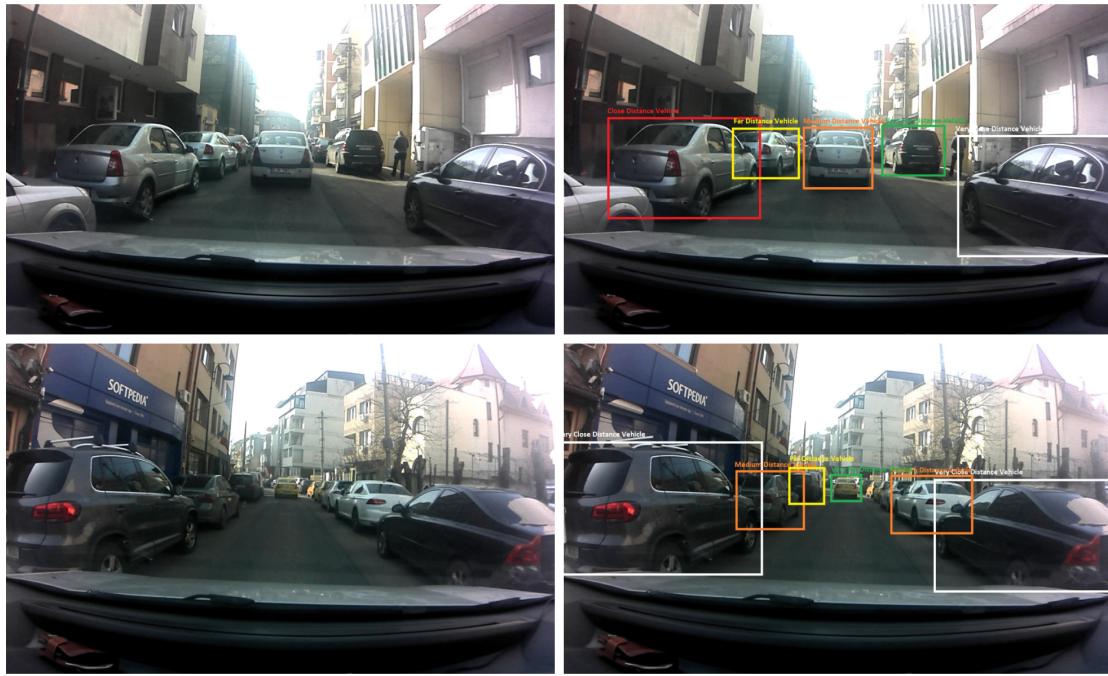


Figura 1.1: Exemple de cazuri favorabile

Următorul set de imagini va înfățișa scenariul situațiilor mai puțin favorabile, unde întâmpinăm diversi factori externi precum condiții meteorologice sau luminozitate scăzută.

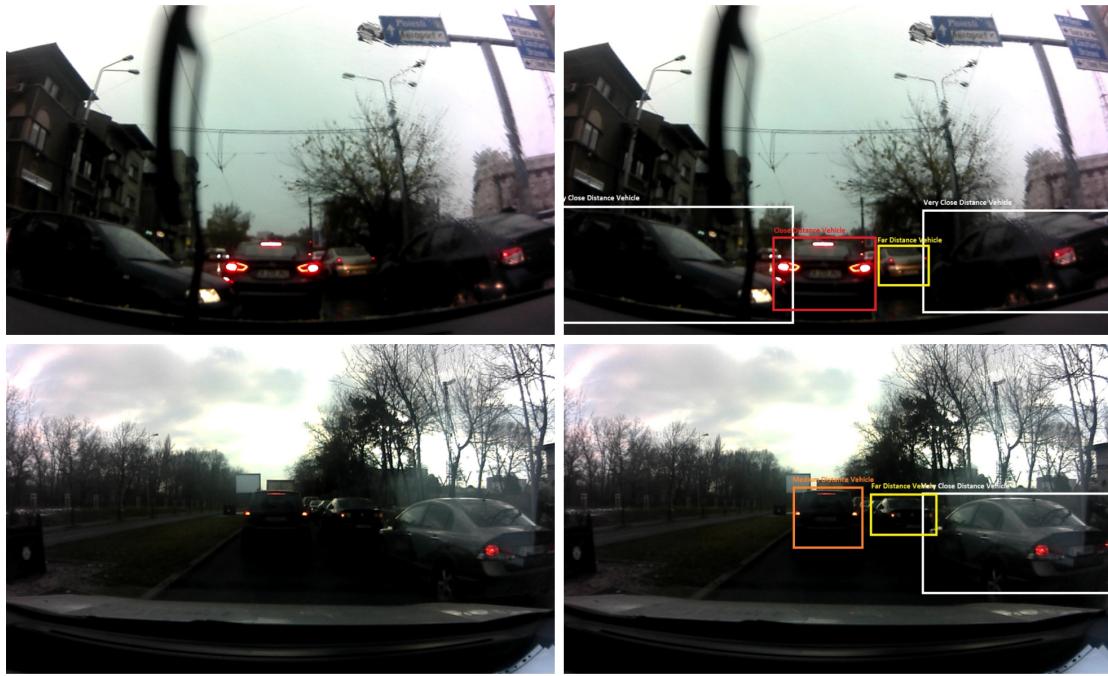


Figura 1.2: Exemple de cazuri nefavorabile

Ultimul set de imagini din acest paragraf va înfățișa unul din frecvențele cazuri de dificultate ridicată, cazul detecțiilor pe timp de noapte. Aici condițiile de luminozitate tind spre 0.

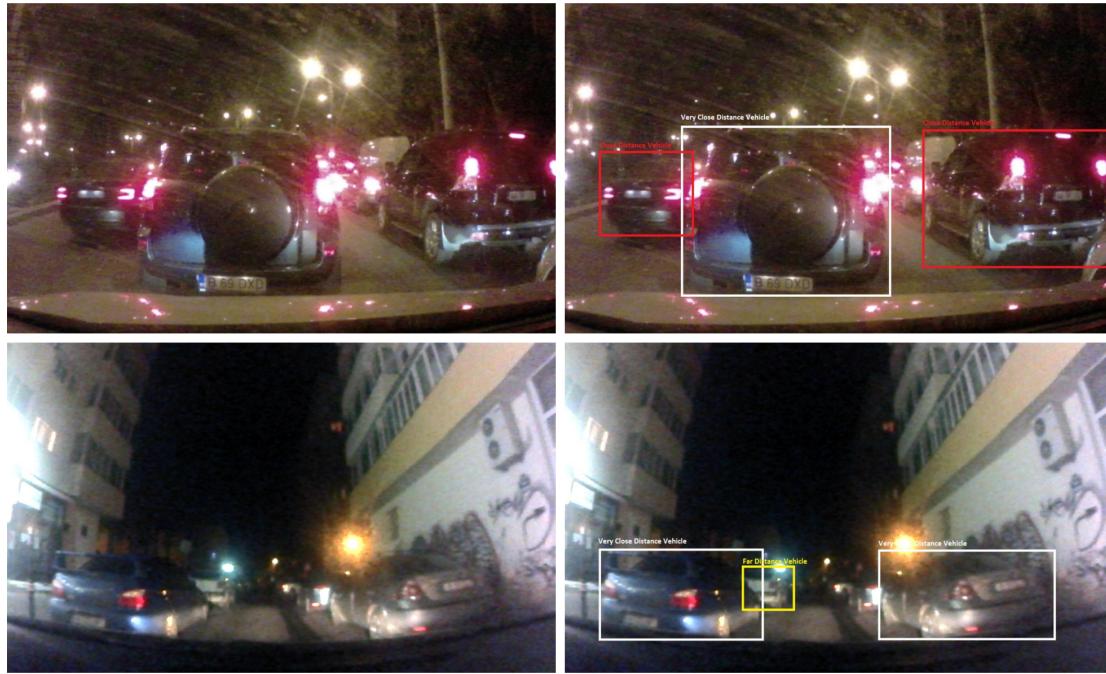


Figura 1.3: Exemple de cazuri pe timp de noapte

Setul de etichete folosit pentru urmatoarele imagini este:

- Very Close Distance Vehicle
- Close Distance Vehicle
- Medium Distance Vehicle
- Far Distance Vehicle
- Very Far Distance Vehicle

Pentru a putea răspunde la multitudinea scenariilor posibile existente în lumea reală, căutăm un algoritm și un mod de reprezentare al imaginilor de input care să fie invariant la majoritatea factorilor externi precum cei prezentați anterior.

Mult timp aceaste probleme de reprezentare a inputului vizual și de generalizare au prezentat o dificultate pentru crearea aplicațiilor de vedere artificială. Însă prin evoluția recentă a domeniului, datorată apariției conceptului de rețele neuronale convolutionale în septembrie 2012 de către Alex Krizhevsky în cadrul competiției de recunoaștere de obiecte ILSVRC2012 [6], o varietate de arhitecturi au început să fie dezvoltate ca posibile soluții pentru probleme actuale.

Alte dificultăți nu au ezitat să apară, cu fiecare pas făcut în dezvoltarea acestui concept s-a realizat că arhitecturile noi apărute au dus atât la creșterea dimensiunii și numărului de parametri învătați de rețea, cât și la creșterea timpului de antrenare și de inferență a acesteia.

Aceste observații ne vor scoate din cadrul contextului ideal unde suntem lipsiți de limitări hardware și ne direcționează spre contextul practic unde adesea întâmpinăm atât probleme la timpul de antrenare, din necesitatea de optimizare a hiper-parametrilor, cât și în pasul de inferență.

Rezolvarea acestor probleme enunțate mai sus o să reprezinte obiectivul lucării. Vom lua în considerare în permanență un echilibru între precizia și acuratețea algoritmului și latența timpului de inferență.

1.3 Repere istorice și contribuție proprie

În faza inițială a dezvoltării aplicației, o perioadă de timp a fost alocată pentru documentarea subiectului și cercetarea diverselor metodologii deja existente.

Cum am specificat mai sus, problema estimării distanței se va transforma într-o problemă de localizare și clasificare a vehiculelor, urmând ca din acele detecții doar cele care să află în fața vehiculului nostru să fie luate în considerare, acest lucru este motivat de decizii ce urmează a fi descrise în capitolele următoare.

Pentru problema detecției din soluția prezentată mai sus, anumite abordări au fost considerate posibile fiecare prezentând avantaje și dezavantaje:

- Detectarea numărului de înmatriculare al mașinii din față prin metode clasice de vedere artificială: dilatare, eroziune, detectare de muchii [7]

- Detectarea vehicului prin metoda ”Histogram of Oriented Gradients with Support Vector Machines” (HOG SVM) [8]

Motivul pentru care niciuna dintre cele două abordări nu a reprezentat o posibilă opțiune pentru soluția finală a fost cauzat de lipsa de generalizare.



Figura 1.4: Tipuri de numere de înmatriculare



Figura 1.5: Tipuri de spate de mașini

Atât în cazul detecției numărului de înmatriculare, cât și în cazul detecției vehiculului în sine cu ajutorul unui HOG SVM, algoritmul nu a putut să generalizeze toate posibilitățile, motiv pentru care am recurs la abordări mai complexe și mai puternic computaționale, și anume rețele neuronale convoluționale.

Cum am specificat anterior, este necesar ca din toate vehiculele detectate doar cele care sunt filtrate ca fiind în fața noastră să fie luate în considerare ca și categorii de distanță valide.

Pentru această subproblemă există posibilitatea unui algoritm pentru detectia liniilor marcaj de pe asfalt [9].

Motivul pentru care și această metodă era impracticabilă erau condițiile externe fiind foarte variabile, pentru că nu toate drumurile prezintă marcaje vizibile sau chiar marcaje.

Această subproblemă va fi referită ulterior ca ”problema de calibrare”, iar și pentru aceasta vom folosi o metodă de învățare automată, care va fi descrisă ulterior în capitolele care urmează.

1.4 Structura lucrării

Fundamentare Teoretică: În cadrul acestui capitol vom prezenta concepte avansate de computer vision precum: Convoluții Separabile Depthwise, Nivele Reziduale Inversabile sau Bottleneckuri Liniare. Acestea sunt necesare deoarece reprezintă conceptele principale ale arhitecturilor pe baza cărora vom experimenta ulterior.

Vom prezenta arhitecturi de rețele convolutionale specializate pe echilibrul între eficiența timpului de inferență și precizia predicției.

Vom încheia acest capitol prin prezentarea unor metrii specifice sistemelor pentru detectarea de obiecte, împreună cu arhitectura principală a algoritmului pentru detectare a vehiculelor.

Tehnologii Aplicate: În acest capitol vom prezenta sumar tehnologiile care au permis dezvoltarea acestei aplicații, specificând pentru fiecare rolul acesteia în cadrul lucrării.

Descriere Soluție: În acest capitol vom explica abordarea folosită.

Vom porni de la manipularea datelor, prezentând modalitatea de împărțire pentru a asigura echilibrul distribuțiilor pentru fiecare set de date.

Vom prezenta rezultatele a două rețele convolutionale pentru detectarea de vehicule atât grafic, cât și vizual. Vom adăuga observații și comparații în detaliu pe baza acestora.

După prezentarea rezultatelor pentru modelele de detectare a vehiculelor, o să urmeze "problemă de calibrare a detecțiilor" care va implica un nou set de date cu vizualizări aferente, urmat de prezentarea rezultatelor unei arhitecturi secundare de rețea neuronală pentru rezolvarea unei probleme de clasificare binară.

Pe baza acesteia vom detecta mașinile din fața noastră.

Rezultate Finale și Concluzii: Acesta va fi capitolul final al lucrării în care vom prezenta rezultatele provenite din evenimente din viața reală, exemplificând atât cazurile favorabile cât și pe cele defavorabile.

Capitolul 2

Fundamentare Teoretică

Urmează să descriem concepte și arhitecturi avansate de rețele neuronale convolutionale necesare în dezvoltarea sistemelor de detectie specializate în reducerea timpului de inferență.

2.1 Conoluții Separabile Depthwise

Prima noțiune abordată în acest capitol o să fie conceptul de conoluție separabilă depthwise, aceasta reprezintă o formă de factorizare făcută asupra tipului standard de conoluție.

Forma de factorizare constă în separarea procesului de filtrare și de combinare a caracteristicilor în doi pași distincti, spre deosebire de o conoluție normală unde acest proces se face într-un singur pas. Acest procedeu este reprezentat prin aplicarea unui singur filtru asupra fiecărui canal din volumul de intrare (input), urmată de combinarea volumelor de ieșire (output) printr-o conoluție cu un filtru de dimensiune 1×1 , procedeu numit conoluție pointwise.

Acest proces de factorizare prezentat mai sus a dus atât la o îmbunătățire semnificativă a timpului computațional, cât și la scăderea drastică a numărului de parametri necesari a fi învățați. Totuși pentru a putea înțelege atât procesul descris, cât și îmbunătățirile aduse este necesară o diferențiere mai aprofundată din punct de vedere matematic asupra procesului standard de conoluție și a

convoluțiilor separabile depthwise.

Pentru aceasta o să pornim de la un context fixat în ambele cazuri și vom observa diferențele.

Considerăm că avem un volum de input de forma $D_F \times D_F \times M$ unde D_F reprezintă înălțimea și lățimea volumului, iar M reprezintă adâncimea sau numărul de canale, în cazul unei imagini acel M o să fie 3 ce ar corespunde numărului canalelor de culoare (exemplul: RGB).

Asupra acestui volum inițial vom aplica un filtru de dimensiune $D_K \times D_K \times M$ rezultând un nou volum de dimensiune $D_G \times D_G \times 1$, iar după aplicare a N filtre de acest tip vom obține volumul de output de forma $D_G \times D_G \times N$.

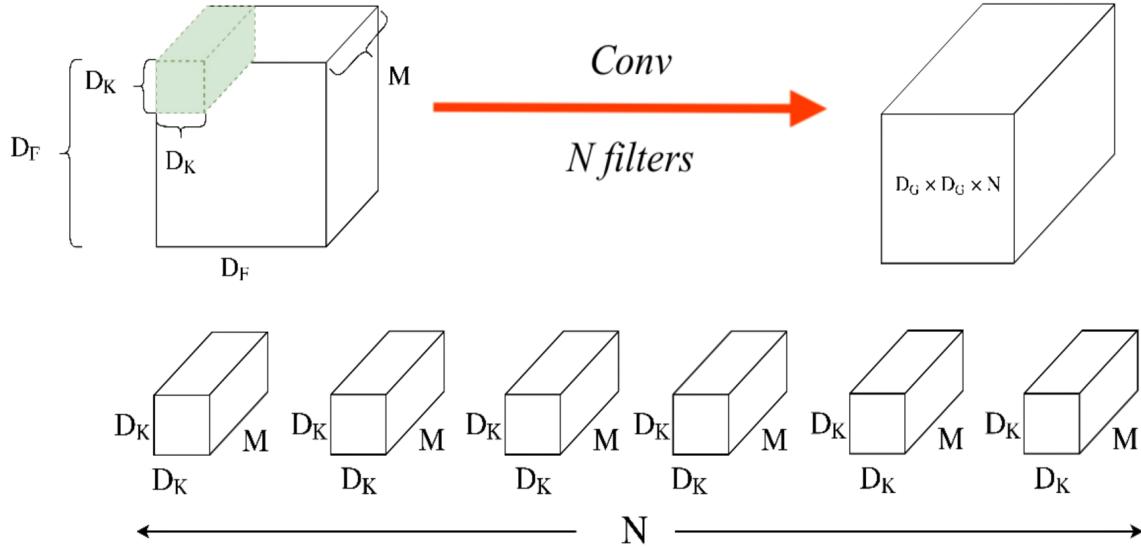


Figura 2.1: Convoluția Standard

Cum am spus anterior, pentru o mai bună comparație între cele două procedee este necesară o analiză a costului computațional. Pentru aceasta vom face referire la numărul de înmulțiri execuțate, comparativ cu numărul de adunări deoarece cele menționate anterior (numărul de înmulțiri) au un timp de execuție mai mare.

Pentru o conoluție standard cu padding și pas (stride) de 1 vom considera formula:

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,j,m,n} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \quad (2.1)$$

[1]

Aceasta având un cost computațional de:

$$D_K \cdot D_K \cdot M \cdot D_G \cdot D_G \cdot N \quad (2.2)$$

Se poate remarcă cum $D_K \cdot D_K \cdot M$ reprezintă dimensiunea unui filtru (kernel), acesta fiind aplicat complet pentru a genera un singur canal de dimensiune $D_G \times D_G$, acest proces va fi repetat de N ori (numărul de filtre) pentru a crea volumul final de dimensiune $D_G \times D_G \times N$.

Acum că am stabilit costul computational al unei convoluții standard, putem trece mai departe la procedeul de convoluție separabilă depthwise.

Față de convoluția standard ce are efectul de a extrage caracteristici pe baza filtrelor convolutionale și de a le combina pentru a produce noi reprezentări, vom folosi o formă de convoluție factorizată numită convoluție depthwise pentru a separa procedeul anterior în două etape una de filtrare și una de combinare a caracteristicilor extrase. Astfel vom rupe interacțiunea între numărul de canale de ieșire și dimensiunea filtrului, lucru ce va produce o substanțială reducere a costului computational.

[1]

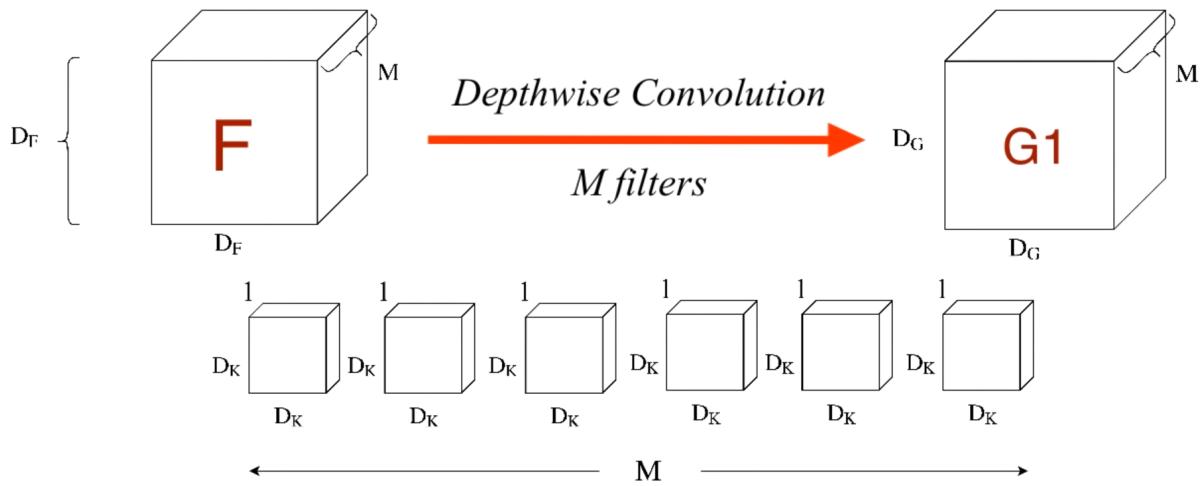


Figura 2.2: Convoluția Depthwise: Faza de filtrare

O să revenim la contextul inițial unde avem un volum de dimensiuni $D_F \times D_F \times M$, pentru prima fază, faza de filtrare, vom aplica pentru fiecare canal al volumului de intrare o filtru de

dimensiune $D_K \times D_K \times 1$, rezultând în M filtre de acest tip.

Pentru aplicarea fiecărui filtru de dimensiune $D_K \times D_K \times 1$ vom obține un volum de ieșire de dimensiune $D_G \times D_G \times 1$, iar după concatenarea acestor canale vom obține un volum de ieșire din prima fază de dimensiuni $D_G \times D_G \times M$.

Faza a doua constă în crearea unei combinații liniare, prin aplicare de filtre de conoluție de dimensiune $1 \times 1 \times M$.

În acest fel prin aplicarea unui filtru de dimensiune $1 \times 1 \times M$ asupra volumului obținut prin conoluție depthwise de la faza de filtrare vom avea un canal de dimensiune $D_G \times D_G \times 1$, iar prin aplicare a N asemenea filtre vom avea un output final de dimensiune $D_G \times D_G \times N$, acesta având aceeași dimensiune precum volumul obținut printr-o conoluție standard.

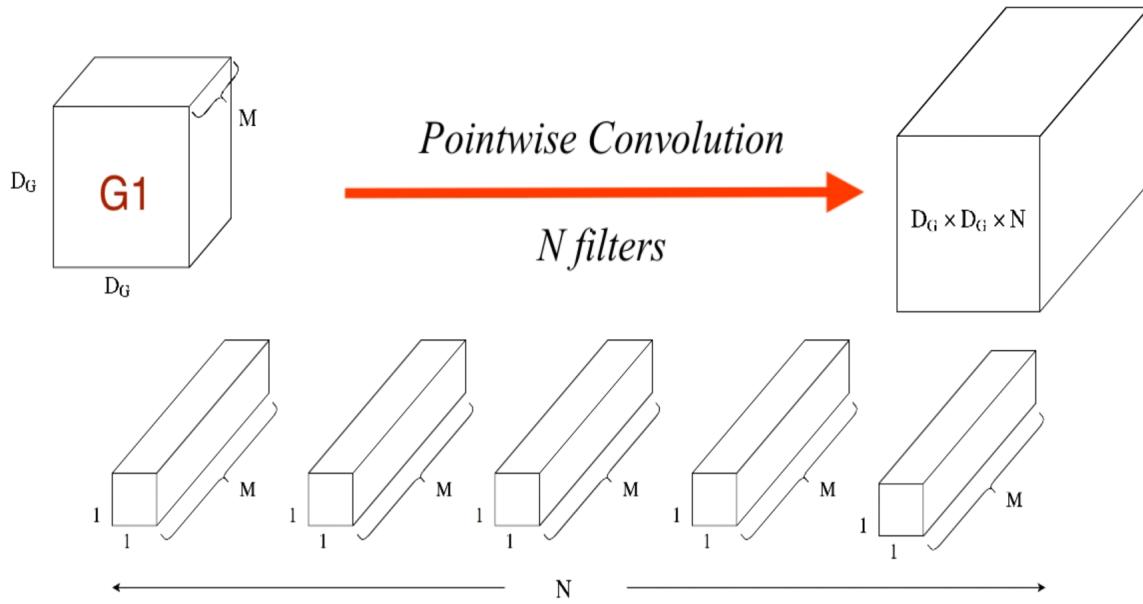


Figura 2.3: Conoluția Pointwise: Faza de combinare

Formula corespunzătoare conoluțiilor depthwise cu un singur filtru aplicat pentru fiecare canal din volumul de intrare poate fi scrisă astfel:

$$\hat{\mathbf{G}}_{k,l,m} = \sum_{i,j} \hat{\mathbf{K}}_{i,j,m} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \quad (2.3)$$

unde

- $\hat{\mathbf{K}}$ reprezintă filtrul de conoluție de dimensiune $D_K \times D_K \times M$
- filtrul numărul m din $\hat{\mathbf{K}}$ este aplicat canalului m din volumul \mathbf{F} producând volumul final $\hat{\mathbf{G}}$

[1]

Cum conoluțiile separabile depthwise sunt alcătuite din faza de filtrare și cea de combinare a caracteristicilor, respectiv conoluția depthwise și conoluția pointwise, putem calcula costul computațional ca fiind o sumă din cele două etape.

Pentru etapa întâi cea de conoluție depthwise avem costul computațional:

$$D_K \cdot D_K \cdot D_G \cdot D_G \cdot M \quad (2.4)$$

Se poate remarcă faptul că factorul $D_K \cdot D_K$ reprezintă dimensiunea filtrului sau poate fi exprimată ca fiind $D_K \times D_K \times 1$, unde 1 reprezintă aplicarea a câte unui filtru pe fiecare canal din volumul inițial, urmată de glisarea aceluia filtru pentru a crea un canal de dimensiune $D_G \times D_G$ acest proces fiind repetat de M ori pentru fiecare canal din volumul inițial.

Pentru etapa a doua cea de conoluție pointwise avem costul computațional:

$$M \cdot D_G \cdot D_G \cdot N \quad (2.5)$$

Unde M reprezintă numărul de înmulțiri dintr-un filtru de dimensiune $1 \times 1 \times M$ urmată de glisarea acestuia peste volumul ieșit din faza de conoluție depthwise de dimensiune $D_G \times D_G \times M$ astfel obținându-se un volum de dimensiune $D_G \times D_G \times 1$, iar prin repetarea acestui procedeu de N ori se va obține volumul final de dimensiune $D_G \times D_G \times N$.

Costul final computațional al conoluțiilor separabile depthwise fiind reprezentat de suma celor două costuri prezentate anterior și anume:

$$D_K \cdot D_K \cdot D_G \cdot D_G \cdot M + M \cdot D_G \cdot D_G \cdot N \quad (2.6)$$

Având calculate cele două costuri computaționale, cel pentru conoluția standard și respectiv

cel pentru convoluția separabilă depthwise, putem calcula un raport pentru a determina eficiența din punct de vedere computațional în funcție de numărul de operații de multiplicare corespunzător fiecăreia:

$$\frac{\text{Nr. Multi. Convoluție Separabilă}}{\text{Nr. Multi. Convoluție Standard}} = \frac{D_K \cdot D_K \cdot D_G \cdot D_G \cdot M + M \cdot D_G \cdot D_G \cdot N}{D_K \cdot D_K \cdot M \cdot D_G \cdot D_G \cdot N} = \frac{1}{N} + \frac{1}{D_K^2} \quad (2.7)$$

Pentru cazul în care am înlocui N cu 1024 și D_K cu 3, în formula de mai sus am obținere 0.112 reprezentând faptul că în acest caz convoluțiile separabile depthwise sunt de aproximativ 9 ori mai rapide decât convoluțiile standard.

Din punct de vedere al numărului de parametri necesari a fi învățați de către rețea convolutională este sesizabil o reducere semnificativă, luând în considerare următoarele:

Pentru convoluția standard avem $N \cdot M \cdot D_K \cdot D_K$ parametri, unde N este numărul de filtre aplicate, iar $D_K \cdot D_K \cdot M$ provine de la dimensiunea filtrului de $D_K \times D_K \times M$.

Pentru convoluția separabilă depthwise avem $M \cdot D_K \cdot D_K + N \cdot M$ parametri, unde $M \cdot D_K \cdot D_K$ reprezintă numărul de parametri din partea de convoluție depthwise în care cele M filtre de dimensiune $D_K \times D_K$, iar $N \cdot M$ sunt numărul de parametri din convoluția pointwise unde avem N filtre de dimensiune $1 \times 1 \times M$.

Știind numărul de parametri din ambele procedee putem determina eficiența și din punct de vedere a dimensiunii rețelei, unde numărul redus al parametrilor este direct proporțional cu timpul necesar învățării acestor parametri.

$$\frac{\text{Nr. Parametri Convoluție Separabilă}}{\text{Nr. Parametri Convoluție Standard}} = \frac{M \cdot D_K \cdot D_K + N \cdot M}{N \cdot M \cdot D_K \cdot D_K} = \frac{1}{N} + \frac{1}{D_K^2} \quad (2.8)$$

După cum se poate vedea obținem un raport egal cu cel calculat pentru costul computațional, motiv care justifică argumentul eficienței și din punct de vedere al dimensiunii rețelei.

Aceste aspecte și argumente fiind prezentate putem avansa în acest capitol în prezentarea unor arhitecturi de rețele neuronale convolutionale numite MobileNets. Acestea dispun atât de un timp

de inferență scăzut, cât și de un număr de parametri redus specific pentru dezvoltarea aplicațiilor de vedere artificială pe dispozitive embedded.

2.2 Arhitectura MobileNetV1

Prezentând în secțiunea anterioară avantajele convoluțiilor separabile depthwise față de convoluțiile standard, putem trece la îmbunătățirile practice aduse de acestea comunității prin diverse arhitecturi de rețele neuronale convolutionale.

Arhitectura prezentată, MobileNetV1, este prima dintr-o serie de modele de rețele convolutionale eficiente atât din punct de vedere computațional, cât și din punct de vedere al numărului de parametri necesari a fi învățați. Acestea sunt concepute special pentru realizarea aplicațiilor de vedere artificială pe sisteme embedded sau dispozitive mobile.

Cum am precizat anterior, de la apariția rețelelor neuronale convolutionale comunitate de computer vision a încercat să împingă acest concept prin dezvoltarea de arhitecturi mai "adânci" și mai complexe cu scopul de obține rezultate mai bune. Chiar dacă aceste avantaje pot îmbunătăți acuratețea rețelei, nu implică și eficiența din punct de vedere a timpului și spațiului. Majoritatea abordărilor în această direcție pot fi descrise ca și comprimarea unor arhitecturi pre-antrenate prin cuantizare, modalitate prin care se poate îmbunătăti dimensiunea rețelei, dar nu și viteza, sau antrenarea directă a unor arhitecturi mai mici. [1]

În cele ce urmează vom prezenta arhitectura rețelei, parametri particulari și rezultatele aferente extrase din articolul oficial [1], pentru a putea înțelege aplicabilitatea acesteia în contextul estimării distanței prin detectarea de vehicule.

Factorul principal al eficienței pentru MobileNetV1 este datorat construcției aproape integrale pe baza conceputului de convoluții separabile depthwise, cu excepția primului nivel (layer) care va rămâne un nivel complet convolutional, aşa cum se poate observa din arhitectura prezentată mai jos din tabelul 2.1.

Tabel 2.1: Arhitectura Rețelei Convolutionale MobileNetV1 [1]

Type / Stride	Filter Shape	Input Size
Conv / s2	3 x 3 x 3 x 32	224 x 224 x 3
Conv dw / s1	3 x 3 x 32 dw	112 x 112 x 32
Conv / s1	1 x 1 x 32 x 64	112 x 112 x 32
Conv dw / s2	3 x 3 x 64 dw	112 x 112 x 64
Conv / s1	1 x 1 x 64 x 128	56 x 56 x 64
Conv dw / s1	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 128	56 x 56 x 128
Conv dw / s2	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 256	28 x 28 x 128
Conv dw / s1	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 256	28 x 28 x 256
Conv dw / s2	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 512	14 x 14 x 256
5 x Conv dw / s1	3 x 3 x 512 dw 1 x 1 x 512 x 512	14 x 14 x 512 14 x 14 x 512
Conv / s1	1 x 1 x 512 x 1024	7 x 7 x 512
Conv dw / s2	3 x 3 x 1024 dw	7 x 7 x 1024
Conv / s1	1 x 1 x 1024 x 1024	7 x 7 x 1024
Avg Pool / s1	Pool 7 x 7	7 x 7 x 1024
FC / s1	1024 x 1000	1 x 1 x 1024
Softmax / s1	Classifier	1 x 1 x 1000

Un alt detaliu, poate nu atât de important, dar totuși necesar de expus, este faptul că între fiecare nivel (layer) de conoluție depthwise și cel de conoluție 1×1 este adăugată normalizare (batch normalization) și un layer de activare non-liniar de tip ReLU.

Finalul nivelelor de conoluție este dat de un layer de extragere al mediei pentru reducerea dimensionalității (average pooling), acesta fiind urmat de un nivel conectat complet cu un clasificator softmax.

Arhitectura rețelei MobileNet a fost concepută astfel încât aproximativ 95% din timpul computațional este petrecut în nivele de conoluție 1×1 care în același timp dețin și 75% din numărul total de parametri necesari a fi învățați, restul provenind din nivelele conectate complet, așa cum poate fi observat din tabelul 2.2 extras din articolul aferent.

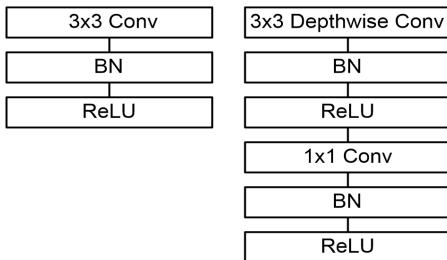


Figura 2.4: Arhitectura unui layer convolutional standard cu normalizare și ReLU non-liniaritate prezentată comparativ cu arhitectura unui layer de conoluții separabile prezente în MobileNetV1 [1]

Tabel 2.2: Resurse pentru fiecare tip de nivel [1]

Type	Mult-Adds	Parameters
Conv 1 x 1	94.86%	74.59%
Conv DW 3 x 3	3.06%	1.06%
Conv 3 X 3	1.19%	0.02%
Fully Connected	0.18%	24.33%

Dezvoltatorii arhitecturii MobileNetV1 au prezentat în cadrul articolului doi hiper-parametri optimizabili pentru construirea rețelelor de diferite dimensiuni în funcție de constrângerile problemei respective. Cei doi hiper-parametri sunt:

- Multiplicatorul de Lățime (Width Multiplier)
- Multiplicatorul de Rezoluție (Resolution Multiplier)

În librăria folosită pentru dezvoltarea aplicației de estimare a distanței este implementat doar multiplicatorul de lățime, multiplicatorul de rezoluție fiind setat în general din dimensiunea imaginii de intrare, aşa cum este precizat în articol. Din acest motiv o să punem accent doar pe primul hiper-parametru, iar pentru cel de-al doilea prezentând doar posibila îmbunătățire din punct de vedere computațional.

Vom considera multiplicatorul de lățime α , iar multiplicatorul de rezoluție ca fiind ρ , precum în articol. [1]

Rolul multiplicatorului de lățime este acela de a subția rețea uniform la fiecare nivel. Pentru un nivel dat și un multiplicator de lățime α , numărul de canale de intrare M devine αM , iar pentru N canale de ieșire devin αN [1], unde costul computațional devine:

$$D_K \cdot D_K \cdot D_G \cdot D_G \cdot \alpha M + \alpha M \cdot D_G \cdot D_G \cdot \alpha N \quad (2.9)$$

unde $\alpha \in (0, 1]$, acest hiper-parametru reducând costul computațional și numărul de parametri cu un factor de α^2 fiind folosit adesea în cazuri în care antrenăm o rețea fără ponderi (weights) învățate anterior.

În finalul acestei scurte prezentări a modelului MobileNetV1 voi adăuga rezultatele obținute de autorii articolului atât ca rețea independentă pe setul de date ImageNet, cât și parte integrată a unui sistem de detectare de obiecte tabel 2.5 fiind antrenată pe setul de date COCO (Common Objects in Context). Scopul este acela de a putea înțelege mai bine aplicabilitatea acestei arhitecturi în situația estimării distanței între mașini, cu respect la echilibrul între acuratețe și viteză sau dimensiuni.

Compararea arhitecturii MobileNetV1 ca model independent va fi făcută atât față de arhitecturi mai complexe, cu dimensiuni semnificativ mai mari (tabelul 2.3), cât și față de arhitecturi de același tip care încearcă să găsească un echilibru între acuratețe și latență (tabelul 2.4).

Tabel 2.3: Comparatie între MobileNetV1 și modele populare [1]

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Tabel 2.4: Comparatie între MobileNetV1 și alte modele de dimensiuni reduse [1]

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
SqueezeNet	57.5%	1700	1.25
AlexNet	57.2%	720	60

După aceste scoruri comparative, cum am spus anterior, mai rămâne să prezintăm rezultatele modelului în cadrul sistemelor pentru detectare a obiectelor, cu precizarea că metrica folosită este mAP (Mean Average Precision), acesta fiind specific sistemelor de acest tip urmând a fi prezentat ulterior în acest capitol.

Tabel 2.5: Rezultate MobileNetV1 în sisteme pentru detectare de obiecte [1]

Framework Resolution	Model	mAP	Billion Mult-Adds	Million Parameters
SSD 300	deeplab-VGG	21.1%	34.9	33.1
	Inception V2	22.0%	3.8	13.7
	MobileNet	19.3%	1.2	6.8
Faster-RCNN 300	VGG	22.9%	64.3	138.5
	Inception V2	15.4%	118.2	13.3
	MobileNet	16.4%	25.2	6.1
Faster-RCNN 600	VGG	25.7%	149.6	138.5
	Inception V2	21.9%	129.6	13.3
	MobileNet	19.8%	30.5	6.1

După cum putem observa din toate detaliile prezentate anterior, MobileNetV1 reprezintă un puternic candidat în soluționarea problemei abordate în cadrul acestei lucrări de licență.

Până la a experimenta aceasta arhitectură în contextul actual o să continuăm să acoperim diverse noțiuni precum și arhitectura succesorului MobileNetV2 pentru a ne extinde multimea de posibile soluții.

2.3 Nivele Reziduale Inversate și Bottleneck-uri Liniare

Pentru a putea prezenta arhitectura lui MobileNetV2 este necesar să introducem conceptele de nivel rezidual inversabil (Inverted Residuals) și de bottleneck-uri liniare (Linear Bottlenecks).

2.3.1 Nivele Reziduale Inversate

În literatură, conceptul de nivel rezidual este reprezentat de legătura directă (skip connection) între un bloc convoluțional de la început și altul ulterior din rețea. Prin acest procedeu există posibilitatea de a accesa în mod direct activările de la începutul rețelei, acestea fiind nemodificate de blocul convoluțional. Acest tip de bloc a apărut din necesitatea dezvoltării arhitecturilor de rețele neuronale mai complexe, de adâncimi ridicate. Acestea permitând înlăturarea problemei de dispariție a gradientătilor (vanishing gradients).

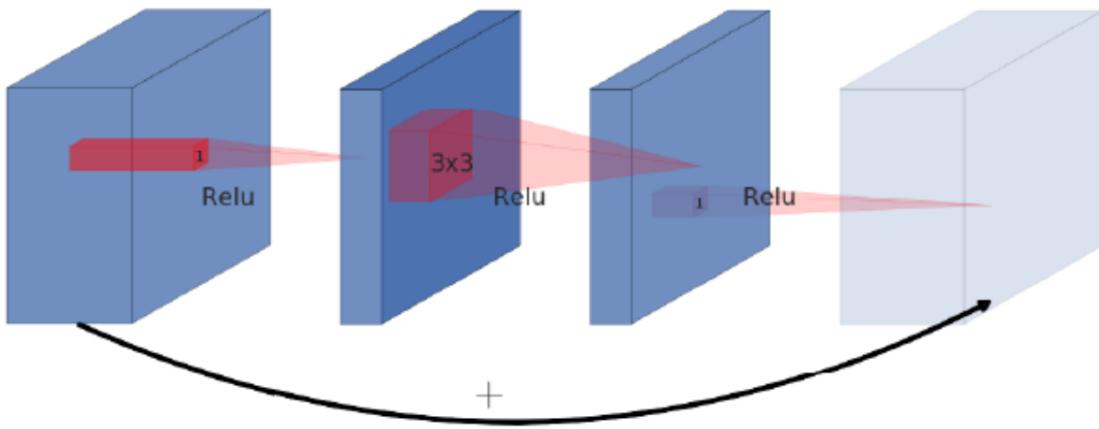


Figura 2.5: Exemplu de conexiune reziduală între nivele de conoluție

Din exemplul de mai sus, figura 2.5, se poate observa cum legătura reziduală este făcută între două blocuri convolutionale cu număr mare de canale. Volumul de input este compresat cu ajutorul unei conoluții pointwise (1×1), această operație fiind urmată de o conoluție cu un filtru de dimensiune 3×3 . Pentru a se forma legătura reziduală volumul final trebuie să aibă aceeași dimensiune cu cel inițial, iar pentru aceasta vom folosi un altă conoluție pointwise (1×1) pentru a crește numărul de canale.

Aceasta ar fi reprezentarea standard a unui bloc convolutional rezidual.

Pentru a putea înțelege și prezenta conceptul de nivel rezidual inversat (inverted residuals layer) trebuie să plecăm de la următoarele ipoteze:

- Hărțile de caracteristici (feature maps) pot fi încadrate / encodeate în spații dimensionale de dimensiuni reduse
- Activările non-liniare prezintă pierderi de informație, contrar abilității lor de a crește complexitatea reprezentării

În mod analog legăturilor reziduale standard, acest tip de nivel primește ca input un volum căruia îi sunt aplicate succesiv trei operații de conoluție, dar diferența principală este dată de numărul de canale al fiecărui volum. Acest tip de legătură reziduală fiind complementară (inversă) celei standard.

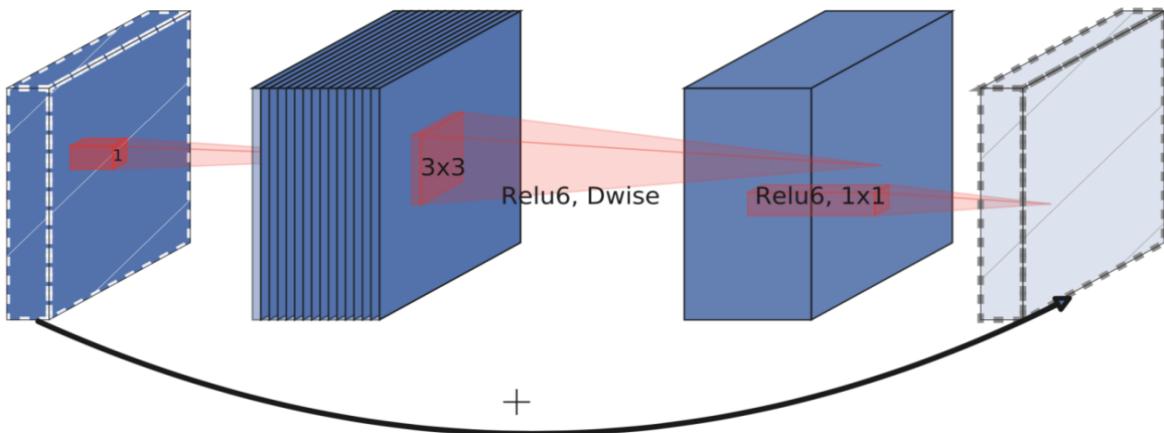


Figura 2.6: Exemplu de conexiune reziduală inversată între nivele de convoluție

Primul tip de convoluție aplicat este unul pointwise (1×1) care duce la expandarea volumului de intrare cu dimensionalitate redusă într-o dimensiune ridicată, aceasta fiind potrivită unei operații de activare non-liniara, unde va fi aplicat ReLU6. [10]

Această activare non-liniară este urmată de o convoluție depthwise cu filtre de dimensiune 3×3 și altă aplicare de ReLU6 pentru realizarea unei filtrări de caracteristici dintr-un spațiu dimensional vast obținându-se tot un tensor de dimensionalitate ridicată.

În cele din urmă spațiul filtrat obținut prin aplicarea activării non-liniare este proiectat într-un spațiu de aceeași dimensiune cu volumul inițial printr-o altă convoluție pointwise (1×1) pentru a se putea realiza legătura reziduală. Proiectarea propriu-zisă într-un spațiu dimensional redus poate duce la pierderi de informație, așa că este necesară o activare liniară precum este precizat în articolul [2].

Autorii lucrării care au dezvoltat conceptul de nivele reziduale inversabile au demonstrat empiric următoarele ipoteze:

- Ultimul nivel de convoluție pointwise (1×1) care proiectează harta de caracteristici într-o dimensiune redusă ar trebui urmat de o activare liniară (Figura 2.7)
- Legăturile reziduale ar trebui stabilite între volume cu dimensionalitate scăzută (Figura 2.8)

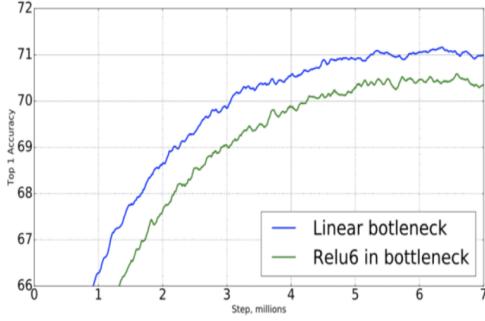


Figura 2.7: Comparație între activare liniară și non-liniară în ultimul nivel [2]

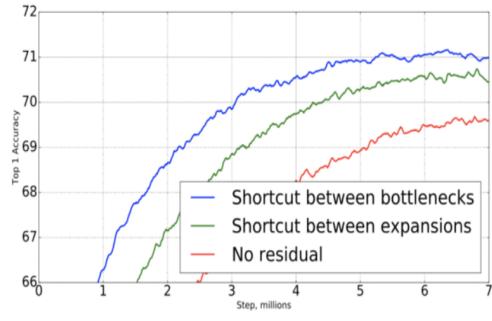


Figura 2.8: Rezultate între diverse tipuri de legături reziduale [2]

Acetea fiind prezentate în cazul nivelelor reziduale inversabile putem trece la următoarea subsecțiune Bottleneck-uri Liniare, fiind deja aduse în discuție și în rezultatele anterioare.

2.3.2 Bottleneck-uri Liniare

În cazul rețelelor neuronale, în absența activărilor non-liniare, orice operație matriceală ar putea fi considerată o combinație liniară ce poate fi învățată de către rețea. Acest lucru cauzând reducerea multiplicărilor de matrici la o singură operație, indiferent de numărul de nivele adăugat.

Astfel pentru construirea rețelelor neuronale cu mai multe nivele (layere) este necesară prezența activărilor non-linare.

În același timp funcții de activare non-liniare precum ReLU, care elimină valorile mai mici ca 0 prin construcția ei, au dezavantajul pierderii de informație, aceasta putând fi evitată prin creșterea numărului de canale, implicit creșterea capacitatei rețelei.

Cu ajutorul straturilor reziduale inversabile, nivelele cu dimensionalitate mai mică sunt adunate, motiv pentru care pierderile de informație sunt necesar de evitat, astfel autorii articolului recomandă utilizarea de activări liniare în acele regiuni ale rețelei.

$$ReLU(x) = \max(0, x) \quad (2.10)$$

$$ReLU6(x) = \min(\max(0, x), 6) \quad (2.11)$$

2.4 MobileNetV2

Având explicate sumar concepțele de Nivele Reziduale Inversabile și Bottleneck-uri Liniare putem trece mai departe să prezentăm arhitectura principalului candidat pentru soluționarea problemei noastre, MobileNetV2.

Această arhitectură este dezvoltată pe baza conceptului de bloc rezidual de tip bottleneck cu convoluții separabile depthwise. Pentru prezentarea acestei secțiuni vom analiza structura unui bloc de tip bottleneck, urmat de arhitectura în sine a lui MobileNetV2, iar în cele din urmă vom prezenta rezultate și observații făcute de autorii articolului ce au dezvoltat algoritmul.

Blocul Rezidual de tip Bottleneck: Luând în considerare informațiile prezentate mai sus în subsecțiunile 2.3.1 și 2.3.2 ne mai rămâne să facem doar o reprezentare formală a conceptului de bloc rezidual de tip bottleneck. Prin urmare vom considera Blocul Rezidual de tip Bottleneck ca fiind o funcție $\mathcal{F}(x)$ cu $\mathcal{F}(x) : \mathcal{R}^{s \times s \times k} \rightarrow \mathcal{R}^{s' \times s' \times k'}$ astfel încât $\mathcal{F}(x) = [A \circ \mathcal{N} \circ B]x$, unde avem A o transformare liniară cu $A : \mathcal{R}^{s \times s \times k} \rightarrow \mathcal{R}^{s \times s \times n}$, \mathcal{N} o transformare non-liniară aplicată pe fiecare canal al volumului cu $\mathcal{N} : \mathcal{R}^{s \times s \times n} \rightarrow \mathcal{R}^{s' \times s' \times n}$, iar în cele din urmă B care reprezintă tot o transformare liniară cu $B : \mathcal{R}^{s' \times s' \times n} \rightarrow \mathcal{R}^{s' \times s' \times k'}$, în cazul lui MobileNetV2 avem $\mathcal{N} = \text{ReLU6}$ o dwise o ReLU6 așa cum prezintă autorii articolului [2].

După prezentarea matematică a blocului rezidual de tip bottleneck putem încerca să vizualizăm acest concept într-o modalitate mai intuitivă și anume prin tabelul 2.6, extras din articoului lui MobileNetV2.

Tabel 2.6: Exemplu Bloc Rezidual de tip Bottleneck, cu factor de expansiune t [2]

Input	Operator	Output
$h \times w \times k$	$1 \times 1 \text{ conv2d, ReLU6}$	$h \times w \times (tk)$
$h \times w \times tk$	$3 \times 3 \text{ dwise S=s, ReLU6}$	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	$\text{linear } 1 \times 1 \text{ conv2d}$	$\frac{h}{s} \times \frac{w}{s} \times k'$

După prezentarea acestei structuri de layer, putem să ne îndreptăm atenția spre arhitectura propriu-zisă a lui MobileNetV2.

Aceasta conține inițial un nivel complet conoluțional cu 32 de filtre, urmată de o serie de blocuri de tip bottleneck fiecare fiind reprodus în funcție de un termen n prezent în tabelul 2.7, extras din articolul [2]. Un alt detaliu important este reprezentat de înlocuirea funcției de activare ReLU cu ReLU6, aceasta fiind folosită deoarece oferă robustețe în cazul operațiilor computaționale de precizie mică.

Tabel 2.7: Arhitectura lui MobileNetV2 [2]

Input	Operator	t	c	n	s
224 x 224 x 3	conv2d	-	32	1	2
112 x 112 x 32	bottleneck	1	16	1	1
112 x 112 x 16	bottleneck	6	24	2	2
56 x 56 x 24	bottleneck	6	32	3	2
28 x 28 x 32	bottleneck	6	64	4	2
14 x 14 x 64	bottleneck	6	96	3	2
14 x 14 x 96	bottleneck	6	160	3	1
7 x 7 x 160	bottleneck	6	320	1	2
7 x 7 x 320	conv2d 1 x 1	-	1280	1	1
7 x 7 x 1280	avgpool 7 x 7	-	-	1	1
1 x 1 x 1280	conv2d 1 x 1	-	k	-	-

Unde t reprezintă factorul de expansiune prezent în articol, n reprezintă lungimea secvenței de blocuri identice, c reprezintă numărul de canale. Primul nivel din fiecare secvență are un pas (stride) s și toate celelalte folosesc un stride de 1.

Dimensiunea filtrului de conoluție în blocul rezidual este, în mod constant, 3×3 , utilizând regularizare dropout și normalizare de tip batch (batch normalization) în timpul antrenamentului.

Factorul de expansiune prezent în articol are efectul de a redimensiona tensorul intermediu precum urmează, pentru un tensor de intrare care are n canale de intrare și ar avea un output de m canale acesta o să treacă după prima operație de conoluție în faza intermedieră de expansiune unde va avea $n \times t$ canale.

Din punct de vedere al hiper-parametrilor, aceștia rămân la fel precum în arhitectura lui MobileNetV1 [1], unde Multiplicatorul de Rezoluție (Resolution Multiplier) este dat de dimensiunea imaginii de intrare și Multiplicatorul de Lățime (Width Multiplier sau Depth Multiplier după implementarea din Tensorflow) care oferă control asupra numărului de canale din rețea.

Hiper-parametri ce permit scalarea arhitecturii în funcție de complexitatea problemei.

Arhitectura fiind explicită putem prezenta câteva rezultate comparative pe setul de ImageNet față de diferite arhitecturi, inclusiv față de MobileNetV1.

Tabel 2.8: Rezultate pe setul de date ImageNet, comparație între diferite arhitecturi [2]

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	3.4M	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	72.0	3.4M	300M	75ms
MobileNetV2 (1.4)	74.7	6.9M	585M	143ms

Din tabelul 2.8 putem observa superioritatea arhitecturii MobileNetV2 față de predecesorul său pe setul de date ImageNet.

Acest aspect nu este suficient pentru a ne finaliza decizia în alegerea arhitecturii, iar din acest motiv o să mai fie necesare atât experimentele și rezultatele etapei de antrenare, cât și experimentele practice din viața reală.

Toate aceste aspecte fiind expuse încheiem scurta prezentare a arhitecturii MobileNetV2 urmând prezentarea metricii și a sistemului pentru detectare de obiecte.

2.5 Mean Average Precision (mAP)

Cum am arătat și anterior, problema estimării distanței urmează să devină o problemă de localizare și clasificare a vehiculelor din imaginea de intrare. Pentru aceasta urmează să descriem metrica folosită atât în cazul lucrărilor științifice pentru sistem de detectare a obiectelor, cât și în cazul problemei noastre și anume Mean Average Precision (mAP).

Inițial vom considera definiția matematică urmând ca pe parcurs să aducem explicații necesare de înțelegere a metricii.

Astfel avem:

$$MAP = \frac{\sum_{q=1}^Q AveP(q)}{Q} \quad (2.12)$$

unde Q reprezintă numărul de interogări

Vom porni de la premisa că metriki standard pentru clasificare precum precizie sau recall (Figura 2.9) sunt deja cunoscute, asemenea și în cazul coeficientului IoU (Intersection Over Union) (Figura 2.10).

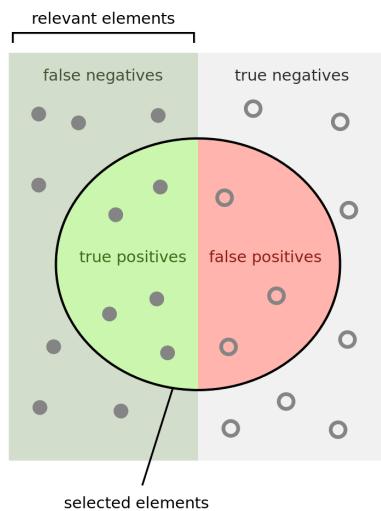


Figura 2.9: Ilustrare Precizie și Recall [3]

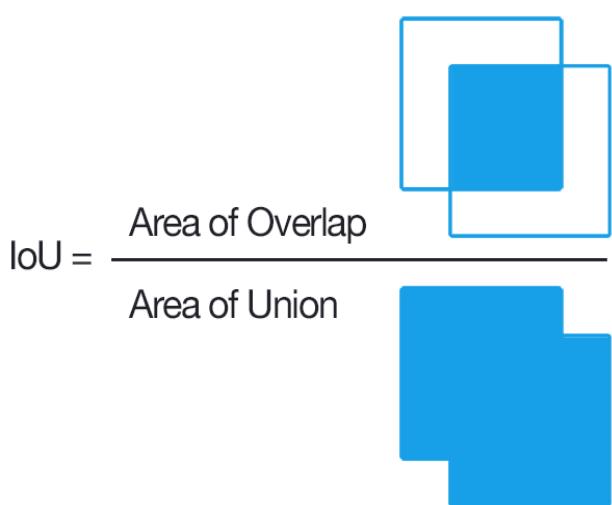


Figura 2.10: Ilustrare Coeficientul IoU [4]

Acstea ipoteze fiind fixate urmează să definim categoriile:

- Adevărat Pozitive (True Positive)
- Fals Pozitive (False Positive)
- Fals Negative (False Negative)

Categoria de Adevărat Negativ (True Negative) nu este necesară a fi definită deoarece pentru fiecare input vom considera că obiectul se află în imagine.

Principalul aspect pentru definirea categoriilor de mai sus o să fie reprezentat de coeficientul IoU, acesta determinând când două ferestre (cel al etichetei de adevăr și cel prezis) sunt suprapuse, astfel:

Tabel 2.9: Determinarea categoriilor detecției

Adevărat Pozitive	Fals Pozitive	Fals Negative
$IoU > 0.5$	$IoU < 0.5$	Modelul nu a efectuat nici o detecție
-	Modelul a prezis ferestre de detectie suprapuse	Avem $IoU > 0.5$, dar modelul a clasificat greșit

Acste categorii fiind definite formal putem calcula curba formată de precizie și recall (Precision-Recall Curve). Pentru aceasta vom calcula și sorta detectiile făcute după nivelul de încredere dat de modelul antrenat. Vom lua un exemplu pentru o prezentare intuitivă.

Pentru acest exemplu este necesară cunoașterea unui alt concept introdus în [11] și anume Precizia Interpolată. Aceasta este calculată pentru fiecare valoare de recall luând precizia maximă posibilă pentru acea valoare de recall. Formula respectivă este:

$$p_{\text{interp}}(r) = \max_{\tilde{r}: \tilde{r} \geq r} p(\tilde{r}) \quad (2.13)$$

Pentru un set de date care conține 5 imagini, cunoaștem că 3 din acele imagini conțin clasa pe care dorim să-o detectăm. Vom lua predicțiile făcute de modelul nostru, sortate descrescător după un nivel de încredere, iar pe baza acestora în funcție de coeficientul de IoU , vom stabili calitatea de adevăr a acelei detectii, mai precis dacă o detectie este Adevărat Pozitivă (TP) sau Fals Pozitivă (FP).

Astfel vom avea următorul tabel cu rezultate:

Tabel 2.10: Rezultate Exemplu Precizie, Recall, Precizie Interpolată

TP / FP	Precizie	Recall	Precizie Interpolată
TP	$1 / 1 = 1$	$1 / 3 = 0.33$	1
FP	$1 / 2 = 0.5$	$1 / 3 = 0.33$	1
TP	$2 / 3 = 0.67$	$2 / 3 = 0.67$	0.67
FP	$2 / 4 = 0.5$	$2 / 3 = 0.67$	0.67
TP	$3 / 5 = 0.6$	$3 / 3 = 1$	0.6

Dacă am plota atât precizia peste recall, cât și precizia interpolată peste recall, acestea ar arăta astfel:

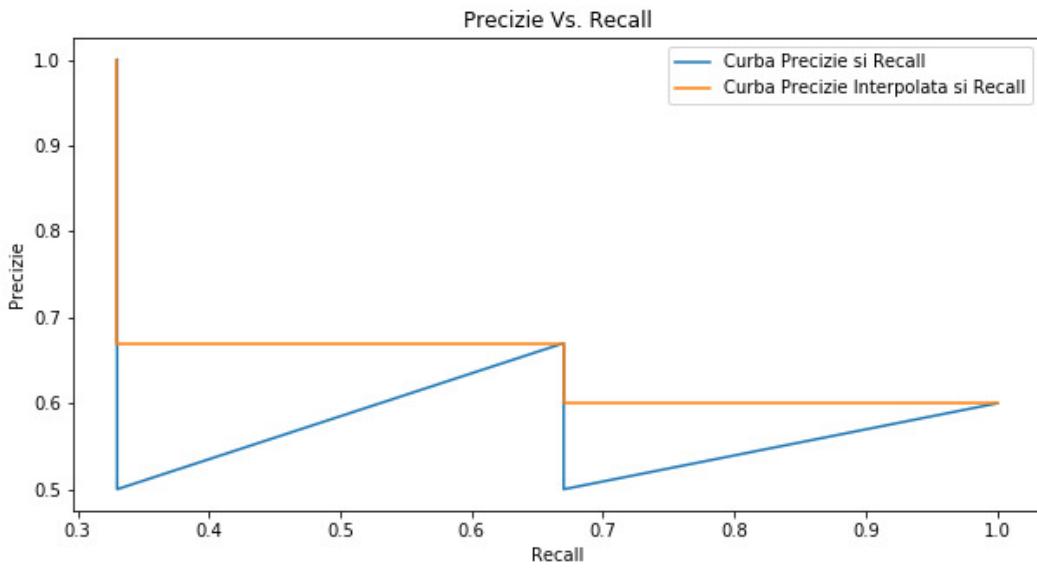


Figura 2.11: Curba de precize și recall comparativ Curbei de precizie interpolată și recall

Precizia interpolată este folosită pentru a reduce oscilațiile prezente în graficul inițial al curbei pentru precizie și recall cauzate de variații în clasamentul detectiilor.

Astfel AP (Average Precision) este calculat luând aria de sub curba preciziei interpolate cu recall. Acest lucru este făcut prin segmentarea intervalului de recall în 11 valori $\{0, 0.1, 0.2, \dots, 0.9, 1\}$. Vom avea formula:

$$\begin{aligned} AP &= \frac{1}{11} \sum_{r \in \{0.0, \dots, 1.0\}} AP_r \\ &= \frac{1}{11} \sum_{r \in \{0.0, \dots, 1.0\}} p_{\text{interp}}(r) \end{aligned} \quad (2.14)$$

Acesta fiind Average Precision, metrica mAP reprezintă media AP-ului calculată peste toate clasele ce dorim să le detectăm.

Metrica mAP poate varia prin pragul coeficientului IoU, spre exemplu $AP@IoU = 0.5$ (cel standard descris anterior), $AP@IoU = 0.75$ (unde coeficientul de IoU din condițiile anterioare este înlocuit).

Alte varietăți care vor fi utile, în special în cazul antrenamentelor pe setul de date al problemei

noastre, sunt:

- $mAP(small)$: mAP pentru obiecte mici: $aria < 32^2px$
- $mAP(medium)$: mAP pentru obiecte medii: $32^2px < aria < 96^2px$
- $mAP(large)$: mAP pentru obiecte mari: $aria > 96^2px$

Astfel finalizăm prezentarea metricii standard utilizată pentru sistemele de detectare a obiectelor în imagini.

Urmează să prezentăm arhitectura unui sistem pentru detectare de obiecte dezvoltat special pentru îmbunătățirea timpului de inferență.

2.6 SSD: Single-Shot MultiBox Detector

Single-Shot Multibox Detector (SSD) reprezintă una dintre ideile revoluționare pentru comunitatea de Computer Vision. Aceasta reprezintă arhitectura unui sistem pentru detectare de obiecte bazat pe conceptul de rețele neuronale convoluționale.

Inovația adusă de Single-Shot Detector constă în lipsa necesității unor zone de inters propuse (region proposals) folosind în schimb ferestre (bounding boxes) la diferite dimensiuni pentru a-și ajusta predicțiile.

Față de alte arhitecturi de vârf, precum FasterRCNN care folosește pentru propunerea de regiuni de interes o altă rețea convolutională separată RPN (Region Proposal Network), arhitectura SSD reușește prin abordarea "feedforward" a unui singur pas prin rețeaua convolutională să reducă semnificativ timpul de inferență păstrând sau chiar depășind scorul de mAP pe setul de date COCO (Common Objects in Context) al lui FasterRCNN.

Însă aceste detalii comparative le vom prezenta ulterior pe parcursul acestei secțiuni după ce descriem arhitectura și concepte specifice pentru Single-Shot Detector.

Arhitectura Single-Shot Detector se bazează pe două concepte fundamentale, nivelele de extragere (feature extraction layers) ale hărților de caracteristici (feature maps) și aplicarea de filtre convolutionale pentru detectarea de obiecte. Nivelele de extragere vor fi provenite de la o rețea de bază. Vom exemplifica cele spuse anterior utilizând ca rețea de bază VGG-16. [12]

Având ilustrată mai jos arhitectura modelului VGG-16 din lucrarea [13] putem observa cum filtrele convolutionale ale modelului VGG-16 devin în arhitectura SSD nivele de extragere ale hărților de caracteristici.

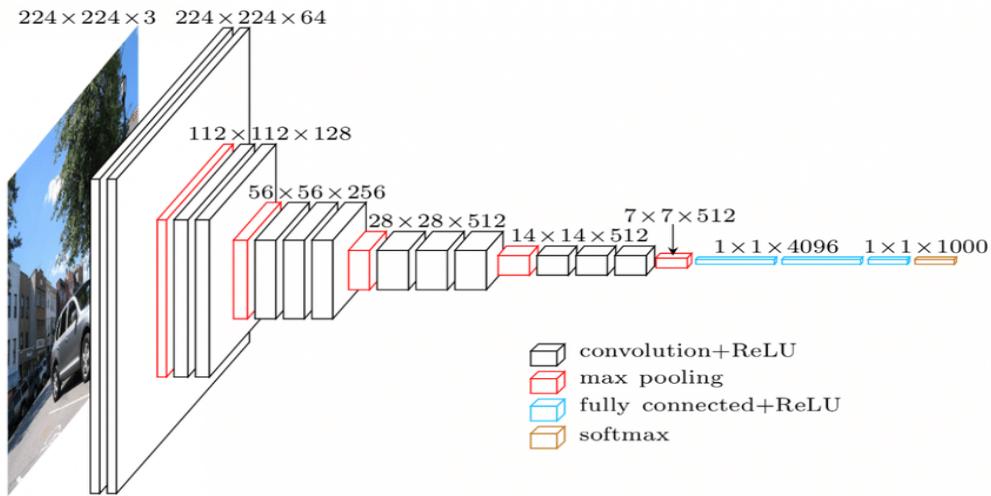


Figura 2.12: Arhitectura Modelului VGG-16

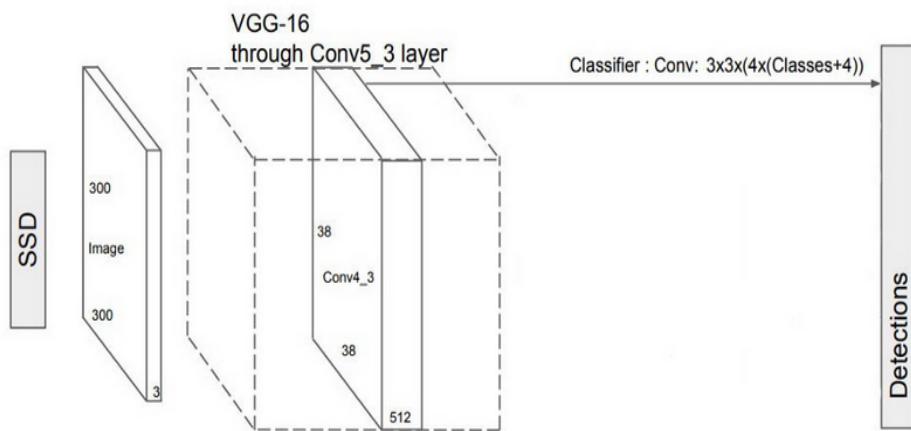


Figura 2.13: Arhitectura parțială a modelului Single-Shot Detector Modificată din articol [5]

Astfel arhitectura Single-Shot Detector folosește VGG-16 pentru extragerea de caracteristici.

Cum am prezentat anterior, pentru detectarea propiu-zisă a obiectelor nu este folosită o rețea convezională pentru propunerea regiunilor de interes, în schimb pentru a genera locația și scorurile pentru fiecare clasă vom aplica filtre convezionale de dimensiune mică (3×3) (Figura 2.13).

Luând spre exemplu primul filtru convezional de detecție Conv4_3 căruia îl se va aplica un filtru de convezie de dimensiune $3 \times 3 \times 4 \times (\text{Clase} + 4)$. Presupunem că există 20 de clase posibile pe care dorim să le detectăm. După extragerea de caracteristici, Single-Shot Detector aplică filtre convezionale de dimensiune 3×3 pentru fiecare celulă din harta de caracteristici extrasă, fiecare filtru având ca ieșire 25 de canale (20 de clase detectate, o clasă pentru fundal, 4 coordonate a ferestrei de detecție / bounding box).

Astfel pentru nivelul de extragere Conv4_3 vom aplica 4 filtre convezionale asupra unui volum de intrare cu 512 canale care vor genera un volum de output cu 25 de canale.

$$(38 \times 38 \times 512) \xrightarrow{(4 \times 3 \times 3 \times 512 \times (21+4))} (38 \times 38 \times 4 \times (21 + 4)) \quad (2.15)$$

Astfel este realizată detecția obiectelor la un nivel de extragere de caracteristici.

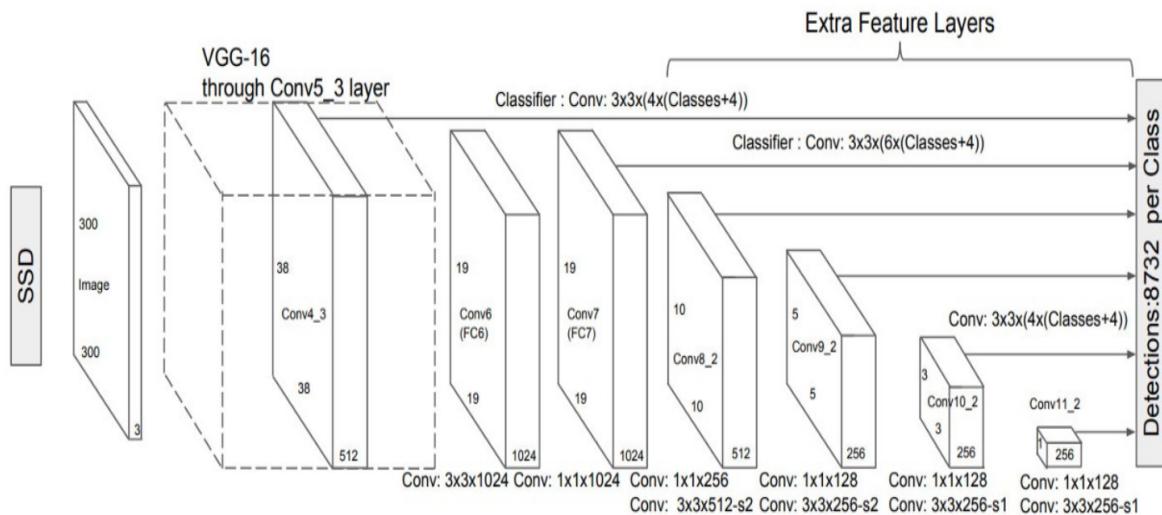


Figura 2.14: Arhitectura Single-Shot Detector [5]

Arhitectura Single-Shot Detector folosește mai multe nivele pentru o detecție independentă a obiectelor (multi-scale feature maps). În acest fel pe baza nivelor inițiale extrase din arhitectura lui VGG-16 vom adăuga mai multe layere de extragere convolutională pentru redimensionarea hărților de caracteristici. SSD folosește nivele convoluționale dinspre finalul rețelei pentru a detecta în bucăți mai mari din imaginea inițială, iar prin hărțile de caracteristici de rezoluție înaltă sunt detectate obiectele mai mici.

Luând în considerare faptul că detectiile noastre pot varia în dimensiune și aspect trebuie să pornim de la un set de ferestre de detecție standard pentru a ușura etapa de antrenament și a îmbunătăți calitatea detecției.

Din acest motiv, dezvoltatorii arhitecturii au ales să folosească un număr de ferestre de detecție pre-selectate manual. Single-Shot Detector folosește un număr de minimum (4 - 6) ferestre standard pentru încadrare. Ferestrele prezise de rețea vor fi relative la ferestrele standard, fiind deviate sau scalate cu un anumit offset.

Astfel pentru fiecare locație din harta de caracteristici vom încadra un număr variat de ferestre standard pentru o posibilă detecție. Iar pentru fiecare fereastră de detecție vom calcula C scoruri pentru clase posibile și 4 coordonate ale ferestrei.

Single-Shot Detector este o arhitectură cu un număr variat de nivele care produc posibile detectii. În cazul modelului standard prezentat în articolul aferent [5] numărul total de ferestre este de 8732, spre deosebire de un candidat precum arhitectura YOLO (You Only Look Once) [14], care dispune de o metodologie asemănătoare și folosește doar 98 de ferestre de detecție, producând scoruri semnificativ mai reduse.

Numărul detectiilor finale nu este de 8732, în funcție de coeficientul IoU între fereastra predicției și fereastra etichetată ca fiind corectă este realizat un proces de supresie non-maximală (non-maximum suppression).

Funcția de pierdere (Loss function) pentru Single-Shot Detector este reprezentată de o sumă ponderată între o funcție de penalizare pentru localizare și o funcție de penalizare pentru încredere a predicțiilor făcute.

$$L(x, c, l, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \quad (2.16)$$

Unde N este numărul de ferestre standard prezise corect. Dacă $N = 0$, vom seta funcția de pierdere la 0.

Funcția de penalizare pentru localizare este reprezentată de o funcție de pierdere *SmoothL1* între ferestra de detectie corectă (g) și fereastra de detectie prezisă (l), unde prezicem centrul (cx, cy) al ferestrei standard (d), lățimea (w) (width) și înălțimea (h) (height) [5].

$$\begin{aligned} L_{loc}(x, l, g) &= \sum_{i \in Pos}^N \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{Smooth}_{L1} (l_i^m - \hat{g}_j^m) \\ \hat{g}_j^{cx} &= (g_j^{cx} - d_i^{cx}) / d_i^w \\ \hat{g}_j^{cy} &= (g_j^{cy} - d_i^{cy}) / d_i^w \\ \hat{g}_j^w &= \log \left(\frac{g_j^w}{d_i^w} \right) \\ \hat{g}_j^h &= \log \left(\frac{g_j^h}{d_i^h} \right) \end{aligned} \quad (2.17)$$

Iar funcția de pierdere pentru încrederea clasificării este o funcție softmax aplicată pentru un număr multiplu de probabilități pentru (c) clase [5].

$$L_{conf}(x, c) = - \sum_{i \in Pos}^N x_{ij}^p \log (\hat{c}_i^p) - \sum_{i \in Neg} \log (\hat{c}_i^0) \quad \text{unde} \quad \hat{c}_i^p = \frac{\exp (c_i^p)}{\sum_p \exp (c_i^p)} \quad (2.18)$$

Cum am spus și în faza inițială a prezentării arhitecturii, Single-Shot Detector reprezintă un echilibru bine definit între viteza și acuratețea detecțiilor. Timpul de inferență fiind extrem de important, mai ales în cazul aplicațiilor dezvoltate pe dispozitive embedded.

Pentru etapa de antrenare, atât în cazul experimentelor din articol [5], cât și în modelele pre-antrenate din Tensorflow, este folosită minarea exemplelor puternic negative cu o ratie de (3 : 1) și augmentarea imaginilor pentru optimizarea mai rapidă și un caracter generic al detecțiilor.

Acestea fiind spuse, putem trece la rezultatele din articol [5]

Tabel 2.11: Pascal VOC2007 rezultate setul de test [5]

Method	Data	mAP
Fast [15]	07	66.9
Fast [15]	07+12	70.0
Faster [16]	07	69.9
Faster [16]	07+12	73.2
Faster [16]	07+12+COCO	78.8
SSD300	07	68.0
SSD300	07+12	74.3
SSD300	07+12+COCO	79.6
SSD512	07	71.6
SSD512	07+12	76.8
SSD512	07+12+COCO	81.6

Tabel 2.12: Pascal VOC2012 rezultate setul de test [5]

Method	Data	mAP
Fast [15]	07++12	68.4
Faste [16]	07++12	70.4
Faster [16]	07++12+COCO	75.9
YOLO [14]	07++12	57.9
SSD300	07++12	72.4
SSD300	07++12+COCO	77.5
SSD512	07++12	74.9
SSD512	07++12+COCO	80.0

Seturile de date folosite pentru Tabelele 2.11 si 2.12 sunt:

- "07" : VOC2007 trainval
- "07+12" : VOC2007 si VOC2012 trainval
- "07+12+COCO" : antrenat pe COCO trainval35k ajustările finale pe "07+12"
- "07++12" : VOC2007 trainval și test împreună cu VOC2012 trainval
- "07++12+COCO" : antrenat pe COCO trainval35k ajustările finale pe "07++12"

Din rezultatele prezentate mai sus putem observa diferență semnificativă de scor a arhitecturii Single-Shot Detector față de alte arhitecturi de vârf precum FastRCNN [15] sau FasterRCNN [16] în diverse experimente.

Din alte analize făcute asupra acestei arhitecturi singura dificultate pe care o prezintă este în cazul detecției obiectelor de dimensiuni mici, unde arhitectura FasterRCNN prezintă un avantaj semnificativ datorat abordării de inferență în doi pași, primul fiind o rețea convolutională pentru propunerea de regiuni de interes RPN (Region Proposal Network).

Chiar dacă arhitectura Single-Shot Detector prezintă o acuratețe ridicată, încă este necesară o analiză din punct de vedere al timpului de inferență. Pentru aceasta vom folosi o analiză făcută de autorii articolui [5] pe un Titan X și cuDNN v4 cu Intel Xeon E5-2667v3@3.20GHz.

Tabel 2.13: Rezultate pe setul de date Pascal VOC2007 test pentru timpul de inferenta

Method	mAP	FPS	batch size	#Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~6000	~1000 x 600
Fast YOLO	52.7	155	1	98	448 x 448
YOLO (VGG16)	66.4	21	1	98	448 x 448
SSD300	74.3	46	1	8732	300 x 300
SSD512	76.8	19	1	24564	512 x 512
SSD300	74.3	59	8	8732	300 x 300
SSD512	76.8	22	8	24564	512 x 512

Putem sesiza cum SSD300 obține un scor de 74.3% mAP la 59 FPS, iar SSD500 obține un scor de 76.8% mAP la 22 FPS, care depășește un FasterRCNN [16] (73.2% mAP la 7 FPS) sau un YOLO [14] (66.3% mAP la 21 FPS). Asfel SSD rămâne singura soluție viabilă în timp real pentru detectarea de obiecte, care depășește un scor de mAP de 70%.

În concluzie, am sumarizat arhitectura unui sistem pentru detectare de obiecte (Single-Shot Detector), care este mai rapid, din punct de vedere al timpului de inferență, față de alte modele similare (YOLO [14]) și semnificativ mai precis decât modele ce utilizează pentru stabilitate și acuratețe o rețea convolutională secundară pentru propunerea de regiuni de interes (FasterRCNN [16]).

Toate acestea fiind prezentate, aici încheiem capitolul de Fundamentare Teoretică și ne îndreptăm spre căutarea unei soluții pe baza celor de mai sus menționate pentru problema estimării distanței între mașini.

Capitolul 3

Tehnologii Aplicate

3.1 Dezvoltarea rețelelor neuronale

3.1.1 Tensorflow Research

În dezvoltarea acestei aplicații, principala componentă utilizată este reprezentată de librăria pentru cercetare și dezvoltare a rețelelor neuronale numită **Tensorflow** [17].

Tensorflow pune la dispoziție atât un API (set de funcții și proceduri) pentru antrenarea și lansarea în producție a sistemelor pentru recunoaștere de obiecte bazate pe rețele neuronale conoluționale (Tensorflow Object Detection API), cât și o colecție pentru arhitecturi de modele pre-antrenate (Tensorflow Models Zoo) utilizabile în funcție de contextul problemei pe care dorim să o rezolvăm.

Cu ajutorul acestui API vom putea antrena și experimenta diverse arhitecturi de sisteme pentru detectarea obiectelor, folosind modele pre-antrenate permitând portarea acestora pentru inferență pe dispozitivul embedded pentru soluționarea problemei de estimare a distanței.

3.1.2 Pytorch

O altă contribuție majoră în dezvoltarea aplicație a fost adusă de librăria **Pytorch** [18]. Aceasta, precum Tensorflow, permite dezvoltarea rețelelor neuronale într-o manieră programatică, ușor de

integrat, care oferă control la nivel de tensor (unitate fundamentală) în fiecare stagiu al dezvoltării arhitecturii.

Pytorch pe lângă capacitatea de a manipula și a face operații pe tensori, mai pune la dispoziție modele pre-antrenate, elemente necesare din arhitecturi de rețele neuronale, o ușoară accesibilitate la nivel de GPU pentru paralelizare și un sistem pentru diferențiere automată (Autograd) pentru toate operațiile făcute la nivel de tensor.

Pe baza acestei librării vom construi un clasificator binar cu scopul de a determina cu ajutorul coordonatelor unei ferestre de detecție și a unei caracteristici separate dacă un vehicul detectat se află sau nu în față noastră.

3.2 Adnotarea datelor

Un alt element foarte important în dezvoltarea aplicației și în același timp în orice problemă de învățare automată a rețelelor neuronale convoluționale este adnotarea datelor. Pentru aceasta a fost folosită o aplicație "open source" numită **labelImg** [19] care permite adăugarea ferestrelor de interes în jurul obiectelor pe care dorim să le detectăm.

Astfel este creat un fișier cu extensia .xml care să conțină toate detaliile necesare convertirii într-un fișier TFRecord folosit ulterior pentru antrenarea sistemului de detectare a vehiculelor.

3.3 Manipularea datelor

3.3.1 Scikit-learn, OpenCV, Pandas, Numpy

Pe parcursul dezvoltării acestei aplicații, accesarea, stocarea și manipularea datelor a reprezentat o puternică necesitate.

Astfel pentru metodologia de împărțire a datelor în cazul clasificatorului binar a fost folosită librăria **Scikit-learn** [20].

Pentru preprocesarea și manipularea inputului vizual a fost folosită librăria **OpenCV** (Open ComputerVision) [21].

În cele din urmă, pentru stocarea și accesarea tabelelor de valori sau alte operații computaționale au fost folosite librării precum **Pandas** [22] și **Numpy** [23].

3.4 Vizualizarea datelor și a rezultatelor

Pentru vizualizări pe parcursul dezvoltării aplicației, făcute atât pentru înțelegerea fenomenului descris de problemă, cât și a rezultatelor obținute, au fost folosite librării precum **Matplotlib** [24] și **OpenCV** [21].

Capitolul 4

Descriere Soluție

Revenim la problema inițială, estimarea distanței între mașini. Cum am precizat și în introducere, vom aborda această problemă pe baza unei metodologii de localizare a vehiculelor și clasificare a intervalului de distanță în categorii.

Pe baza rezultatelor provenite de la sistemul de detecție, vom aplica o arhitectură secundară de rețea neuronală pentru clasificarea detecțiilor cu scopul de a decide care mașini se află în fața vehiculului nostru.

Astfel, după acest proces de filtrare, vom aplica condiții de viteză pentru fiecare categorie de distanță, în funcție de care un eveniment de tip "Safe Distance Warning" (SDW) va fi declanșat.

O precizare importantă este faptul că această aplicație va fi rulată în mod constant pe un dispozitiv încorporat (embedded) cu resurse limitate, concurând cu alte aplicații similare printre care și o arhitectură de rețea convezională similară pentru camera de interior. Astfel, este de la sine înțeles că abordarea alegerii celei mai complexe și performante arhitecturi pentru detectarea de vehiculeiese din discuție.

O să fie necesar găsirea unui echilibru între reducerea timpului de inferență și acuratețea detecțiilor făcute de rețea convoluțională. Pentru acesta vom apela la setul de cunoștințe acumulat în capitolul de Fundamentare Teoretică, urmând experimentarea modelelor de arhitecturi de rețele neuronale convoluționale prezentate anterior.

4.1 Setul de date

În dezvoltarea rețelor neuronale adânci (Deep Neural Networks), în special în cazul rețelelor convoluționale, unul dintre cele mai importante aspecte constă în dimensiunea setului de date. Cum a fost observat și în cazul acestei probleme, unde odată cu creșterea numărului de date a fost sesizabilă o creștere în cazul metricii mAP și o convergență într-un punct de minim mai scăzut a funcției de pierdere.

Setul de date folosit în dezvoltarea aceastei aplicații constă în 19602 imagini adnotate manual, a căror etichete (label-uri) vor fi stocate în fișiere de tip .xml, acestea urmând a fi convertite în formatul TFRecords.

Cum am prezentat și în introducere, în faza inițială setul de etichete a fost reprezentat de următoarele categorii:

- Very Close Vehicle (Distance 1)
- Close Vehicle (Distance 2)
- Medium Vehicle (Distance 3)
- Far Vehicle (Distance 4)
- Very Far Vehicle (Distance 5)

Pe parcursul dezvoltării aplicației vom folosi în special setul de etichete din paranteză, și anume Distance X , unde $X \in \{1, 2, 3, 4, 5\}$. Din analiza pe baza experimentelor anterioare, precum detectarea de vehicule pe bază de HOG SVM, am putut sesiza o problemă de generalizare din motivul diferenței de aspect între spatele unei mașini de dimensiuni normale și spatele unui vehicul de tonaj mare. Din acest motiv am hotărât ca setul de etichete să fie particularizat și în funcție de aspectul și dimensiunea vehiculului detectat.

Astfel setul de etichete a devenit:

- Very Close **Average** Vehicle (Distance 1)
- Close **Average** Vehicle (Distance 2)
- Medium **Average** Vehicle (Distance 3)
- Far **Average** Vehicle (Distance 4)
- Very Far **Average** Vehicle (Distance 5)
- Very Close **Large** Vehicle (Large Distance 1)
- Close **Large** Vehicle (Large Distance 2)
- Medium **Large** Vehicle (Large Distance 3)
- Far **Large** Vehicle (Large Distance 4)
- Very Far **Large** Vehicle (Large Distance 5)

Aceste aspecte fiind prezentate, putem trece mai departe la descrierea setului de date.

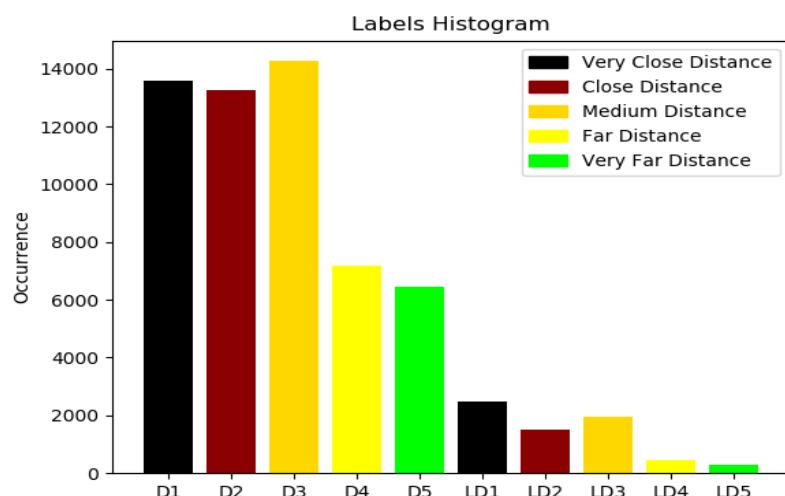


Figura 4.1: Distribuția Claselor pe setul de date

Unde "DX" este "Distance X", iar "LDX" este "Large Distance X", pentru $X \in \{1, 2, 3, 4, 5\}$.

O caracteristică importantă în problemele pe bază de învățare este natura problemei și cât de bine este reprezentată aceasta de setul nostru de date. Pentru asta urmează să folosim figura 4.1 de mai sus, ce reprezintă distribuția claselor din setul prezent de date.

De luat în calcul faptul că fiecare imagine poate conține un număr variat de vehicule, astfel este normal ca suma valorilor din distribuție să fie semnificativ mai mare decât lungimea setului de date.

Putem observa cum anumite clase precum Large Distance 4 sau Large Distance 5 au un număr de exemplare sesizabil mai mic față de restul claselor. Acest lucru este cauzat în principal de natura problemei, deoarece în trafic este mai probabil să întâlnim un vehicul de dimensiuni normale față de vehicule de dimensiuni mari, iar probabilitatea ca acel vehicul să fie observat la o distanță mai mare este semnificativ redusă.

Pe baza acestei informații vom putea considera pe viitor augmentarea specifică a imaginilor ce conțin acel set de clase, însă mai întâi trebuie să observăm rezultatele etapei de antrenare și cele din teren deoarece dorim ca distribuția setului de date să rămână apropiată de cea a lumii reale.

Astfel urmează stabilirea metodologiei de împărțire a datelor, luând în considerare faptul că inferența pe dispozitivul embedded prezintă resurse limitate nu putem lua în calcul o metodologie de cross-validation pentru că ar implica un sistem de votare bazat pe mai multe modele, aşa că suntem nevoiți să ne limităm la împărțirea setului de date în: **Train**, **Validation** și **Test**.

Tinând cont că majoritatea experimentelor pe care urmează să le facem vor fi bazate pe teste din mașină, am considerat:

- setul de Train să conțină 70 % din totalul datelor
- setul de Validare să conțină 20 % din totalul datelor, folosit pentru vizualizarea graficelor de mAP sau funcții de pierdere
- setul de Test (10 %) va fi folosit pentru fixarea de threshold-uri (limite) pentru nivelul de încredere al fiecărei clase, rămânând ca rezultatele vizibile să fie cele din teren.

Acestea fiind menționate mai rămâne un singur aspect extrem de important în împărțirea datelor, și anume distribuția fiecărui set de date. Luând în considerare setul de date nebalansat nu ne putem permite ca anumite clase să nu apară în setul de date, iar pentru acesta vom folosi o metodă de împărțire diferită.

Împărțirea datelor se va face astfel:

- Vom alege aleator din setul inițial o cantitate neglijabilă de date, aproximativ 100 de imagini pentru fiecare set nou de date
- Pentru fiecare imagine nouă aleasă din setul inițial vom calcula vectori de apariție cu fiecare clasă din cele trei seturi de date
- Astfel vom calcula o medie între valorile de la indicii claselor din imaginea nouă împărțind la suma valorilor din vectorii de apariție
- Urmând să alegem cea mai mică medie pentru determinarea setului de date unde plasăm imaginea nouă

Vom lua un exemplu pentru o mai bună înțelegere.

Avem o imagine cu următorul vector de apariție al claselor $X = [2, 0, 1, 0, 1]$, pentru comoditate alegem numărul de posibile clase să fie 5, asta însemnând că noua imagine conține două obiecte din clasa 0, un obiect din clasa 2 și un obiect din clasa 4, pentru o indexare de la 0.

Vom avea următoarele presupuse seturi de date reprezentate prin vectori de apariție cu fiecare clasă:

- $Train = [10, 20, 5, 30, 1]$
- $Validation = [30, 10, 20, 15, 5]$
- $Test = [20, 5, 1, 15, 30]$

Putem observa vizibil că setul de date $Train$ ar fi cel optim pentru adăugarea imaginii noi în încercarea de a echilibra histogramele.

Astfel vom calcula media pentru fiecare vector de apariție corespunzător unui set de date, în funcție de ce clase avem în noua imagine, atunci:

- $Train = \frac{10+5+1}{66} = 0.24$
- $Validation = \frac{30+20+5}{80} = 0.68$
- $Test = \frac{20+1+30}{71} = 0.71$

Subliniez din nou, numitorul mediei este suma elementelor din vector în cazul nostru. Astfel reiese cea mai mică medie ca fiind 0.24, acest aspect însemnând că pentru a echilibră histogrammele claselor pe fiecare set ar trebui să alegem setul *Train* pentru adăugarea noii imagini.

După aplicarea acestui metodologiei pe setul nostru de date vom obține următoarele histograme:

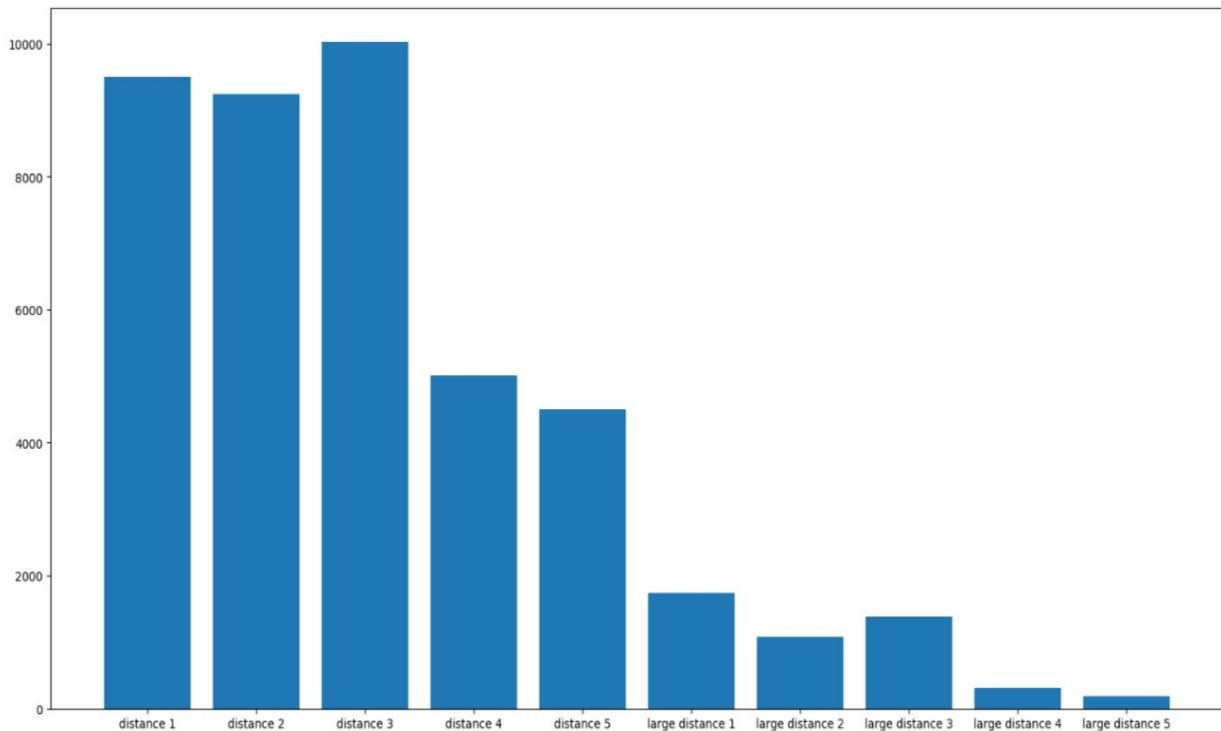


Figura 4.2: Distribuția Claselor pe setul de Train

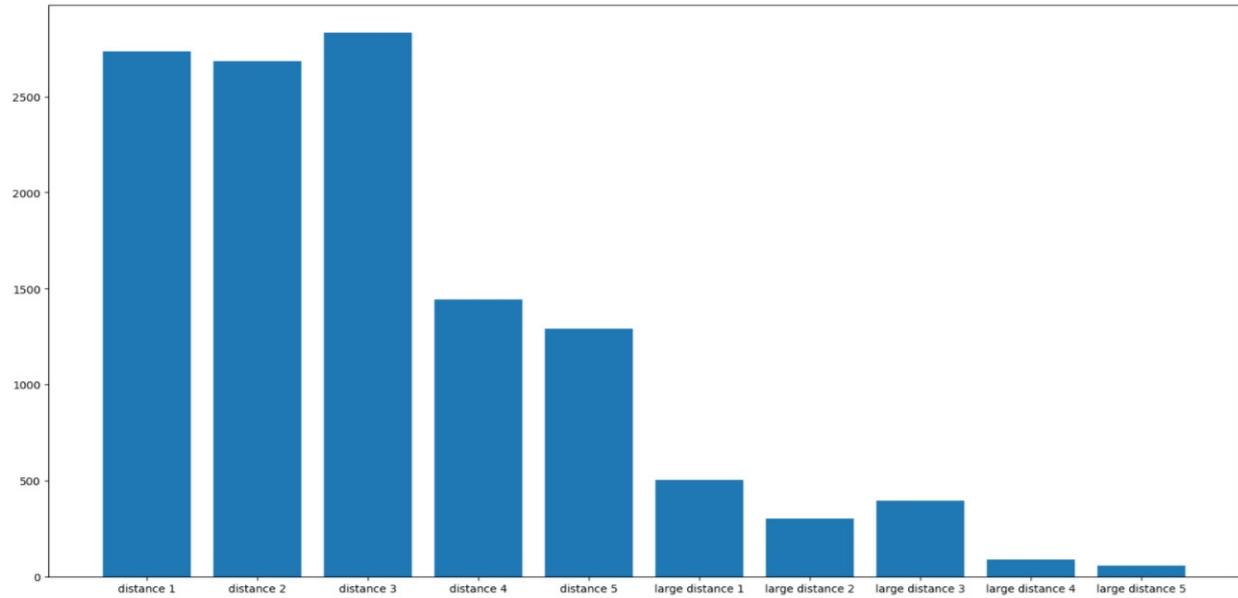


Figura 4.3: Distribuția Claselor pe setul de Validare

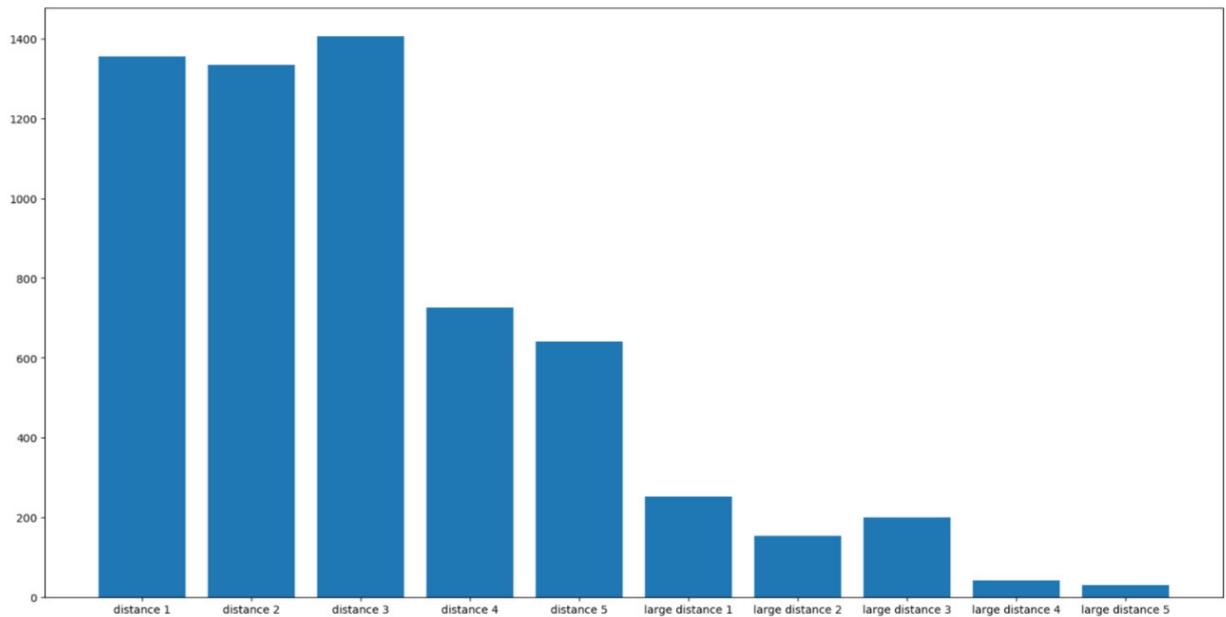


Figura 4.4: Distribuția Claselor pe setul de Test

Acestea fiind prezentate putem trece la faza de antrenare a rețelei convolutionale pentru detectarea de obiecte.

4.2 Antrenarea arhitecturii pentru detectare de obiecte

Luând în considerare atât limitările hardware cu care ne confruntăm, cât și cunoștințele pe care le-am dezvoltat în capitolul de Fundamentare Teoretică, am ales ca pentru sistemul de detectare de vehicule să folosim arhitectura Single-Shot Detector.

Dar în cazul nostru pentru optimizarea vitezei de inferență am decis să nu folosim rețea convolutională VGG-16 [12], utilizând în schimb ca rețea de bază, arhitecturi de modele convolutionale din familia MobileNets prezentată anterior.

Astfel în acest capitol vom analiza rezultatele comparative pentru antrenarea modelelor SSD-MobileNetV1 și SSD-MobileNetV2, ulterior urmând să decidem și necesitatea unor versiuni cu ponderi cuantizate.

4.2.1 Rezultate SSD-MobileNetV1 pentru detectare de vehicule

În mod inițial vom prezenta graficele rezultate pentru setul de validare pentru scorul de mAP cu diverse varietăți, urmate de evoluția funcțiilor de pierdere pentru Single-Shot Detector.

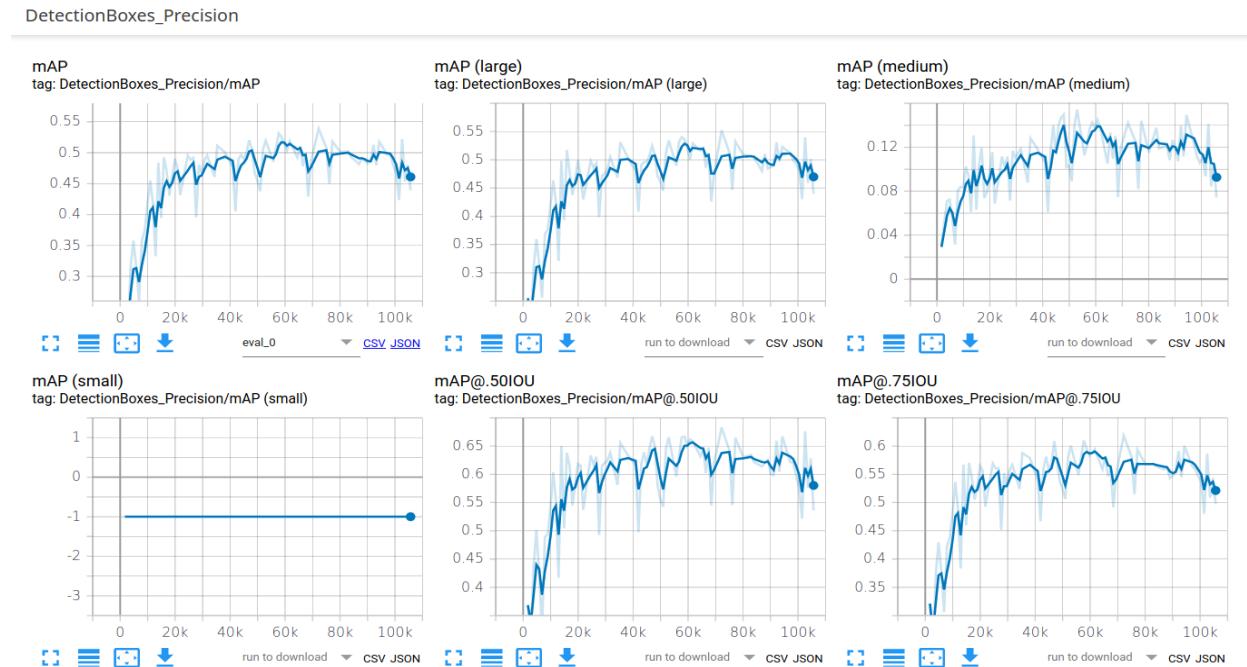


Figura 4.5: Variatii de mAP pentru SSD-MobileNetV1

Procesul de antrenare al rețelelor convecționale fiind extrem de scump computațional, impracticabil pe calculatorul local, a fost folosit un server separat cu o placă video dedicată pentru paralelizarea operațiilor bazate pe tensori. Placa video (GPU) folosită pentru experimente fiind un Nvidia GeForce RTX2080Ti cu 12GB VRAM.

Singurul element care ieșe în evidență din graficele anterioare (Figura 4.5) este scorul la categoria $mAP(small)$ care este egal cu -1. Acest lucru este cauzat de faptul că setul validare nu conține detectii mai mici de $32px^2$, astfel valoarea de -1 fiind valoarea de initializare (default) care va rămâne neschimbată.

Acest aspect rămânând posibil și în urma împărțirii datelor. Noi prin procesul de împărțire a datelor ne asigurăm că toate clasele sunt continute în fiecare set de date, dar aceasta nu implică și dimensiunea ferestrelor de detecție.

Urmează să prezentăm funcțiile de pierdere atât pentru clasificare și pentru localizare separat, dar și graficul funcției de pierdere totală reprezentată de combinația celor două anterioare.

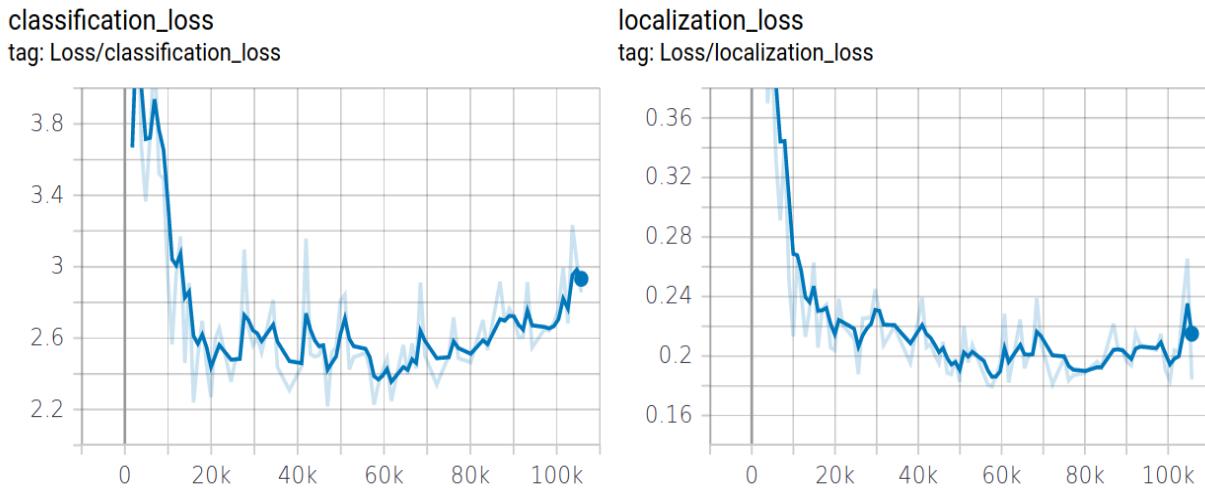


Figura 4.6: Funcțiile de pierdere pentru clasificare și localizare la SSD-MobileNetV1

Putem observa în finalul antrenării o ușoară creștere în cazul funcției de pierdere pentru clasificare, acest lucru este cauzat de încercarea algoritmului de optimizare pentru prezicerea corectă a claselor mai dificile, căutând un minim local mai bun, însă în acest timp alte clase vor fi prezise greșit.

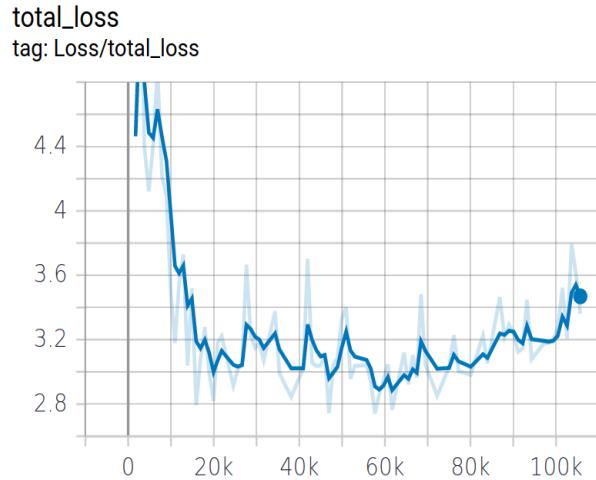


Figura 4.7: Funcțiile de pierdere totală la SSD-MobileNetV1

Vom alege ca model final iterarea unde este găsit minimul global.

Astfel arhitectura noastră va prezenta o valoare a funcției de **pierdere totală de 2.76**, cu o medie a scorurilor de **mAP** pentru diversi coeficienți IoU de aproximativ **52%**, unde scorul de mAP pentru genul acesta de arhitectură pe setul de date COCO (Common Objects in Context) este de aproximativ 19-21%, însă acolo atât numărul de clase, cât și complexitatea problemei sunt semnificativ mai ridicate.

4.2.2 Rezultate Inferență Validare SSD-MobileNetV1

În stânga sunt rezultatele detectorului antrenat, iar în dreapta sunt exemplele Ground-Truth.



Figura 4.8: Inferență Validare SSD-MobileNetV1 - Imaginea 1

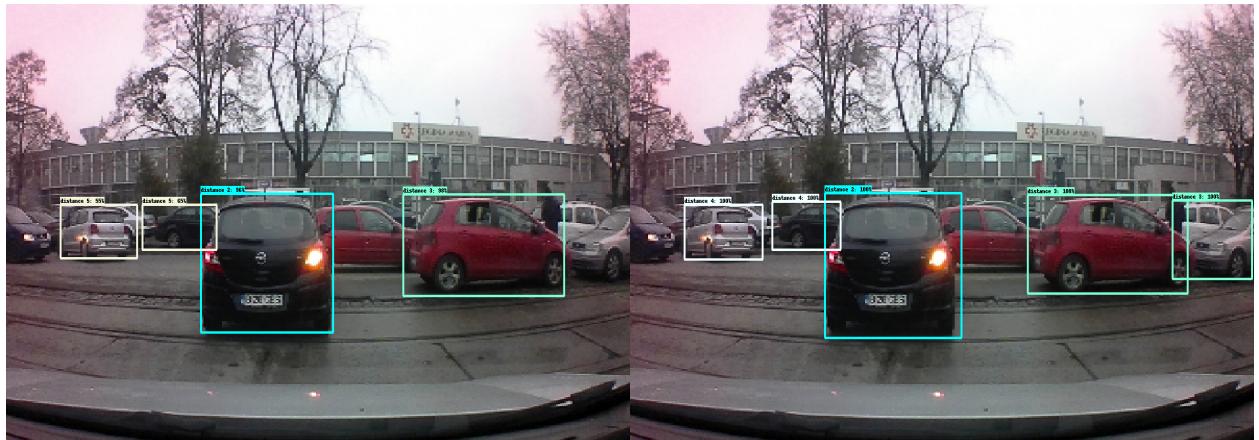


Figura 4.9: Inferență Validare SSD-MobileNetV1 - Imaginea 2

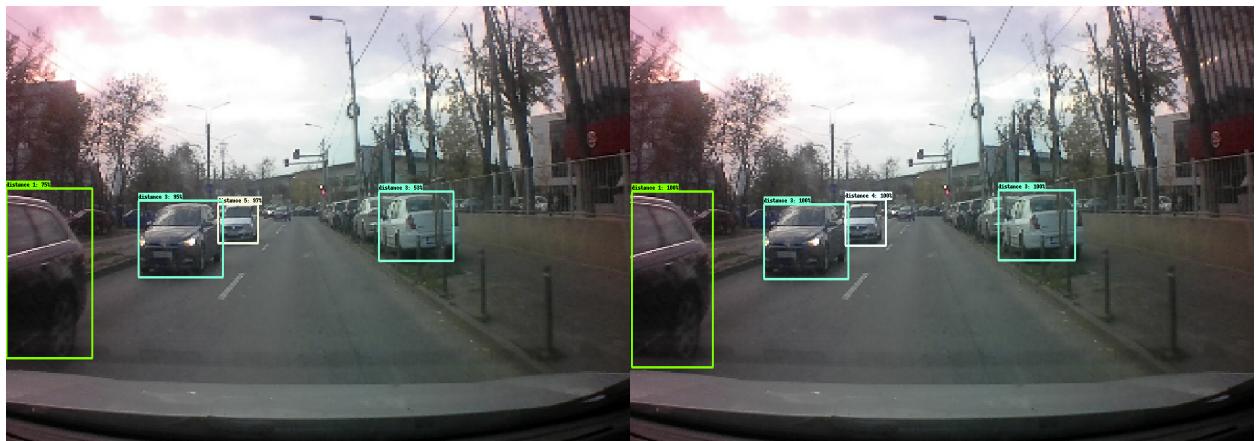


Figura 4.10: Inferență Validare SSD-MobileNetV1 - Imaginea 3

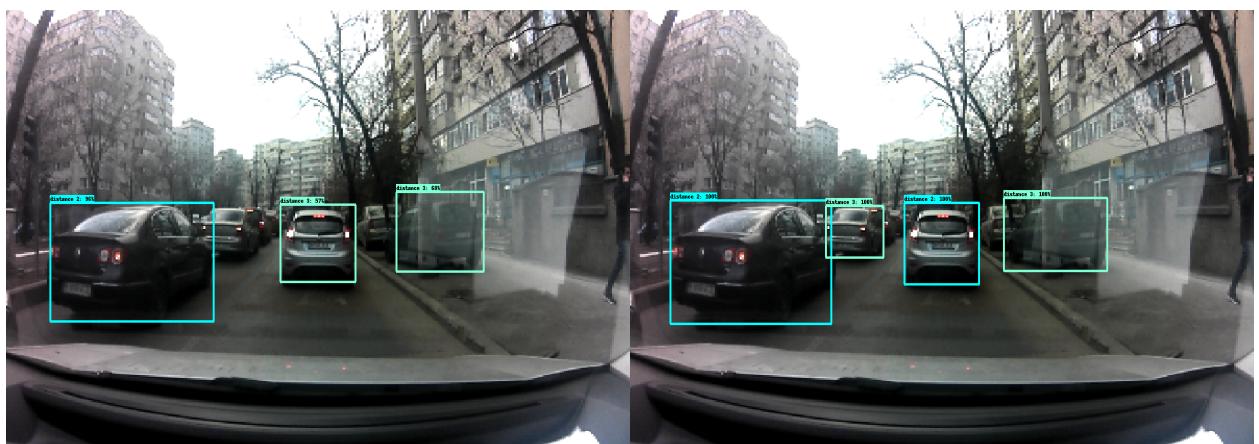


Figura 4.11: Inferență Validare SSD-MobileNetV1 - Imaginea 4

După cum putem observa se pare că algoritmul pentru detectare de vehicule prezintă rezultate bune, chiar și în cazul vehiculelor aflate la distanțe depărtate sau de mare tonaj.

Sesizăm ușoare diferențe pentru detecțiile la clasa Distance 4 cu Distance 5 sau invers, dar, luând în considerare faptul că noțiunea de regresie a fost păstrată, aceste mici erori nu vor prezenta importanță.

Chiar dacă arhitectura SSD-MobileNetV1 prezintă rezultate foarte bune, pentru a ne putea crea o imagine de ansamblu este necesară și analizarea arhitecturii SSD-MobileNetV2, rezultate pe care le vom prezenta în continuare.

4.2.3 Rezultate SSD-MobileNetV2 pentru detectare de vehicule

Fără alte introduceri, vom proceda analog precum în cazul lui SSD-MobileNetV1. Astfel vom prezenta graficele rezultatelor pe setul de validare, urmate de graficele funcțiilor de pierdere.

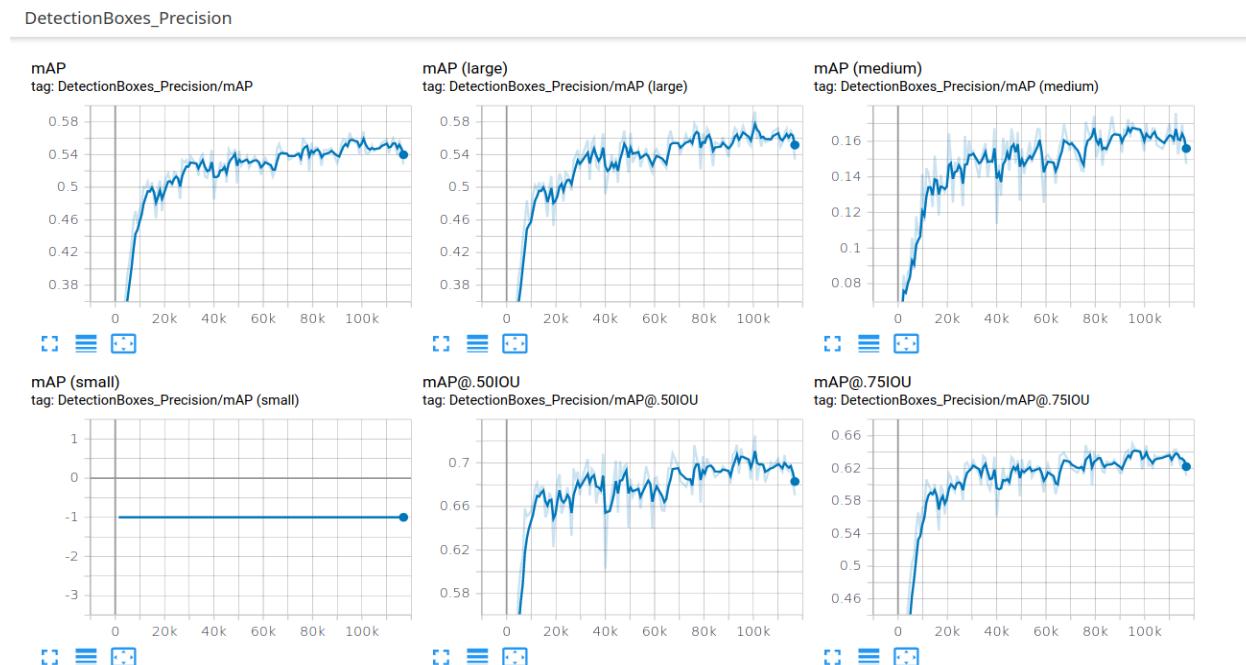


Figura 4.12: Variații de mAP pentru SSD-MobileNetV2

Atât în cazul lui SSD-MobileNetV1, cât și în cazul lui SSD-MobileNetV2 observăm cum $mAP(small) = -1$, acest caz fiind explicitat anterior.

Continuăm cu prezentarea rezultatelor cu evaluarea graficelor funcțiilor de pierdere.

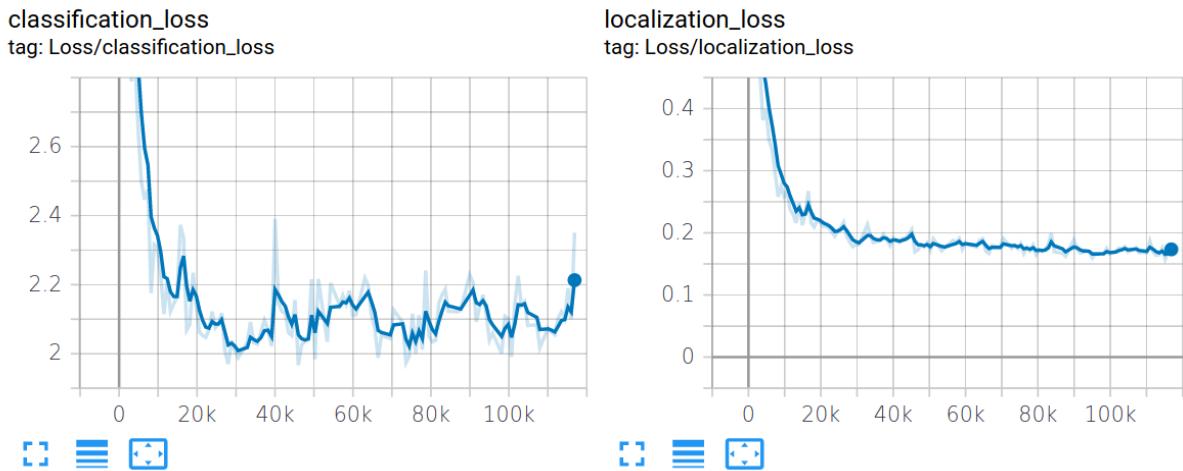


Figura 4.13: Funcțiile de pierdere pentru clasificare și localizare la SSD-MobileNetV2

După cum vom observa din figura următoare 4.14 ce prezintă funcția totală de pierdere, vom avea un punct de minim mai scăzut pentru arhitectura Single-Shot Detector cu MobileNetV2 decât cu MobileNetV1 ca rețea convolutională de bază. Acest factor ar duce intuitiv și la rezultate mai bune la nivel de localizare și detecție.

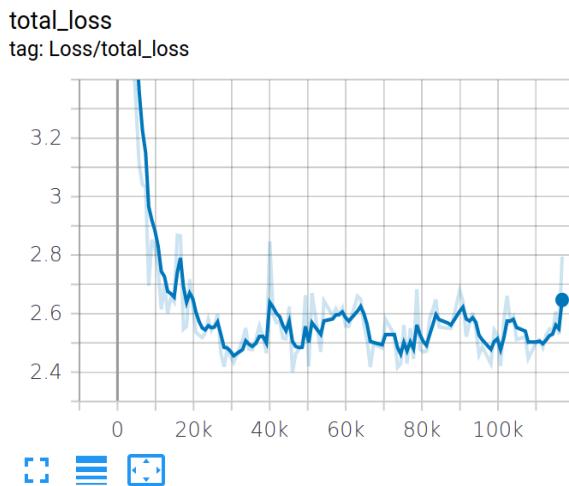


Figura 4.14: Funcțiile de pierdere totală la SSD-MobileNetV2

Astfel, alegând minimul global de pe graficul funcției de pierdere totală, vom obține un model cu o pierdere totală de **2.42**, iar o medie a metricii **mAP** peste diversi coeficienți IoU de aproximativ **57%**. Aceasta fiind o diferență sesizabilă față de scorul de **52%** obținut de SSD-MobileNetV1.

4.2.4 Rezultate Inferență Validare SSD-MobileNetV2

În stânga sunt rezultatele detectorului antrenat, iar în dreapta sunt exemplele Ground-Truth.

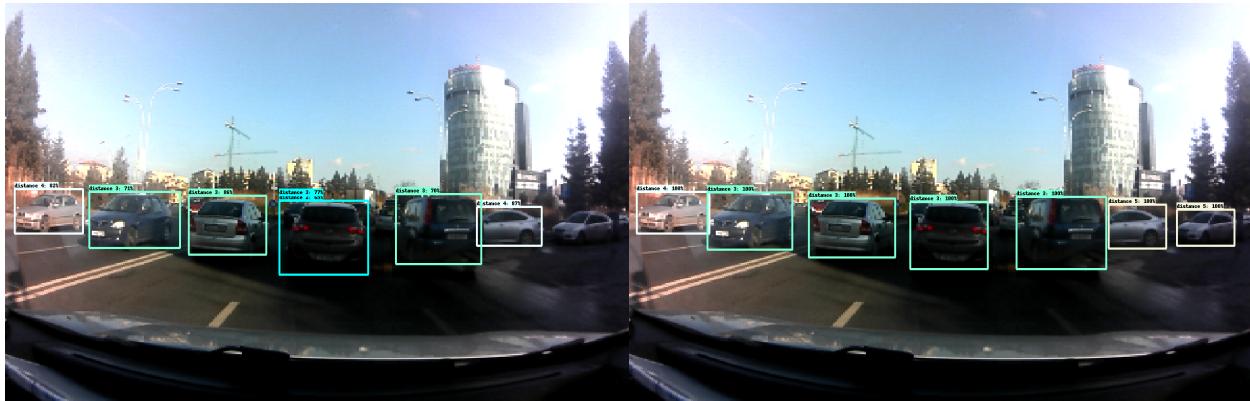


Figura 4.15: Inferență Validare SSD-MobileNetV2 - Imaginea 1



Figura 4.16: Inferență Validare SSD-MobileNet2 - Imaginea 2

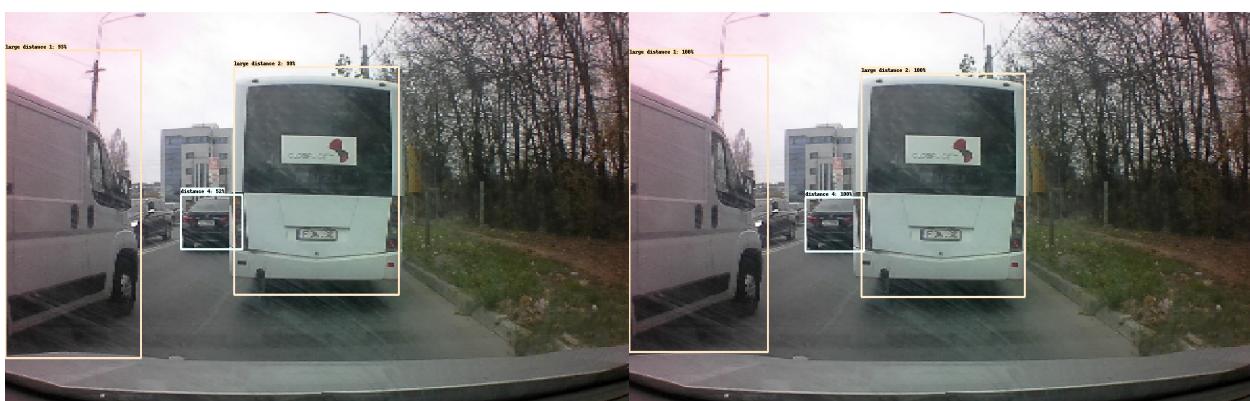


Figura 4.17: Inferență Validare SSD-MobileNetV2 - Imaginea 3

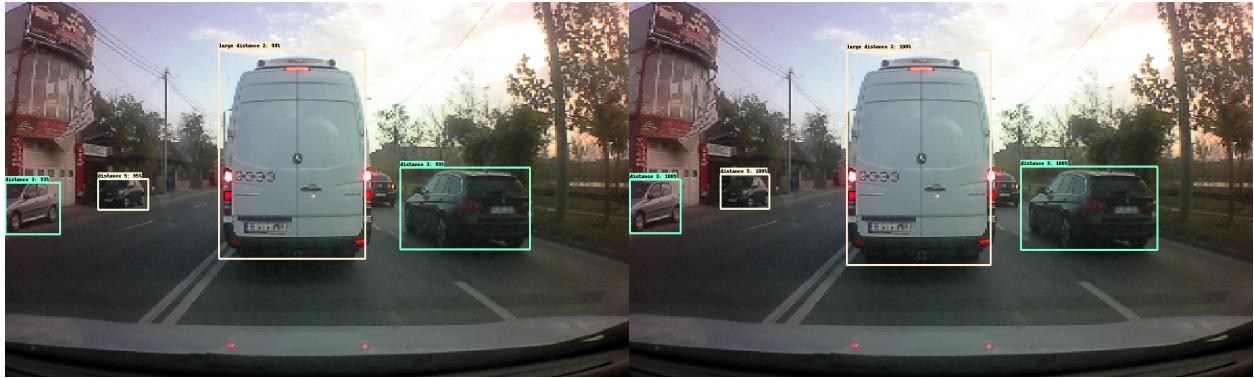


Figura 4.18: Inferență Validare SSD-MobileNetV2 - Imaginea 4

Pe baza acestor rezultate, putem să ne îndreptăm la a doua problemă de învățare automată, și anume detectarea vehiculelor care sunt în fața noastră. Această problemă de clasificare binară va fi referită în capitolul următor ca problema de calibrare a detecțiilor, din motive ce urmează a fi explicate ulterior.

4.3 Problema de calibrare a detecțiilor

Cum spuneam și în capitolul de introducere, este important ca după detecția tuturor vehiculelor din imagine să putem cunoaște care vehicule se află sau nu în fața noastră.

Explicația fiind următoarea: anterior am precizat că noi vom trimite șoferilor un semnal de alarmă (trigger) de Safe Distance Warning atunci când mașina este în mișcare la o anumită viteză și nu păstrează o distanță corespunzătoare. Cum dispozitivul dezvoltat de noi permite amplasare oriunde pe parbrizul mașinii, trebuie să luăm în considerare următorul scenariu:

Ne aflăm pe un drum cu două benzi de circulație, chiar dacă aceste benzi sunt paralele, dispozitivul nu le va percepe astfel. Acesta va forma un punct de vizibilitate maximă (vanishing point) formând un triunghi alungit al cărui vârf tinde la infinit.

Atunci când camera este plasată pe centrul mașinii acest punct de vizibilitate maximă corespunde cu centrul mașinii, iar astfel ce este în fața camerei automat se află și în fața vehiculului nostru.

Dar atunci când camera prezintă o deviație spre șofer, acel triunghi de vizibilitate al camerei va

rezenta și el o deviație spre lateralul mașinii, astfel fiind posibilă detectia vehiculelor de pe banda opusă. Acest fapt fiind inadmisibil deoarece noi am trimite avertismente false șoferului, cum că nu păstrează distanța față de vehiculul din față, în timp ce șoferul merge normal, iar pe cealaltă bandă se deplasează alte vehicule.

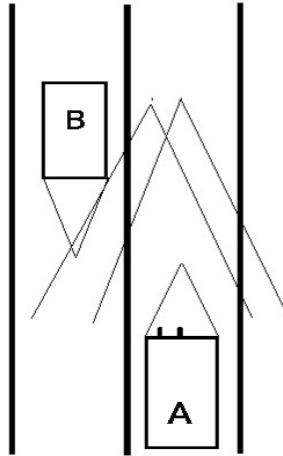


Figura 4.19: Schița descriptivă a problemei de calibrare

Putem sesiza din figura 4.19 că atunci când camera situată în mașina A prezintă o deviație spre șofer, obiectele considerate în fața camerei nu sunt obligatoriu situate și în fața mașinii, lucru cauzat de capacitatea camerei de a surprinde imaginiile la un unghi larg de vizualizare (ultra-wide).

Acesta este motivul pentru care din numărul total de detectii efectuat este necesar să găsim un algoritm care să filtreze păstrând doar detectiile din fața vehicului nostru.

Pentru aceasta am hotărât să folosim o abordare bazată pe machine learning (învățare automată) care să primească drept caracteristici de intrare coordonatele ferestrei de încadrare a detectiei (bounding box) și anume:

- TLX (Top Left X)
- TLY (Top Left Y)
- BRX (Bottom Right X)
- BRY (Bottom Right Y)

Pe lângă toate acestea, vom mai folosi o caracteristică separată, și anume un ”offset” care reprezintă deviația camerei față de centrul mașinii, unde centrul mașinii ar fi 0, partea spre șofer (presupunând că volanul este pe stânga) $-X$, iar partea dinspre pasager să fie $+X$, iar X este distanța în centimetri.

Astfel caracteristicile de intrare pentru algoritmul de calibrare vor fi:

- TLX (Top Left X)
- TLY (Top Left Y)
- BRX (Bottom Right X)
- BRY (Bottom Right Y)
- Offset (Deviatia Camerei)

Astfel am colectat un set de date care conține 32860 de exemple de acest fel, a căror offset și eticheta sunt cunoscute de noi. Din acest set de date 21942 de exemple sunt etichetate ca fiind ”Not In Front” (Label 0) și 10918 exemple etichetate ca fiind ”In Front” (Label 1). Din figura 4.20 putem remarca distribuția de label-uri și din figura 4.21 putem observa distribuția de offset-uri.

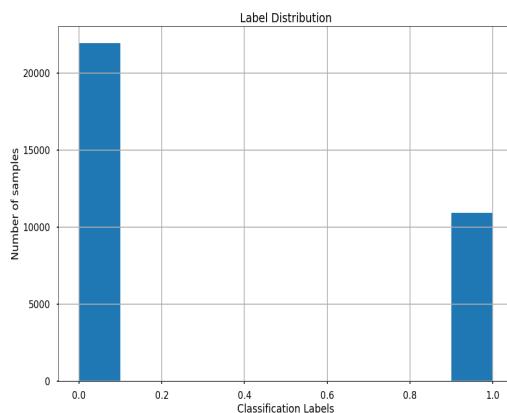


Figura 4.20: Distribuția de etichete pentru problema de calibrare

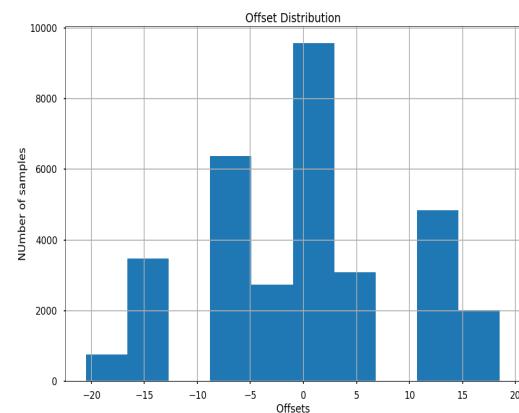


Figura 4.21: Distribuția de offset-uri pentru problema de calibrare

Pentru o mai bună înțelegere a problemei, vom plota centrul fiecărei ferestre de încadrare pentru offset-ul principal, și anume offset 0. Din ce putem vedea este sesizabil un tipar. Iar pentru alte offset-uri rezultatele ar fi la fel, doar ușor deviate în direcția offset-ului.

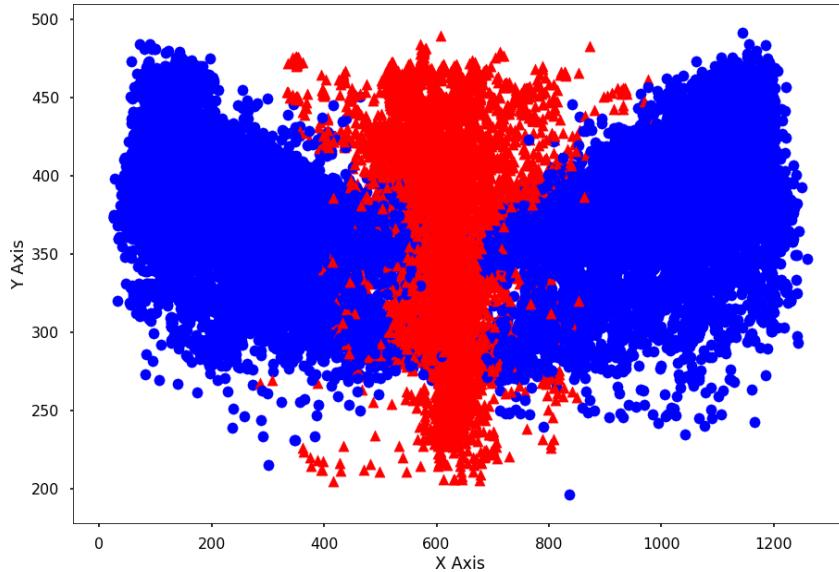


Figura 4.22: Plotarea centrelor ferestrelor pentru detectii la offset 0

Precum în cazul problemei pentru detectarea de obiecte suntem nevoiți să luăm în calcul limitările hardware.

Din acest motiv am hotărât să experimentez diversi algoritmi al căror timp sau număr de operații pentru procesul de inferență să fie redus.

Astfel după încercarea unor algoritmi precum Support Vector Machines, Logistic Regression, k-Nearest Neighbors sau Decision Tree, fiind nemulțumit de rezultatele acestora am hotărât să folosesc o rețea neuronală cu implementarea în Pytorch.

Comparativ cu alte abordări de machine learning, rezultatele rețelei neuronale au fost semnificativ mai bune atât în cazul testelor pe setul de validare, cât și pe teren după găsirea unui threshold de încredere al predicției optimizat.

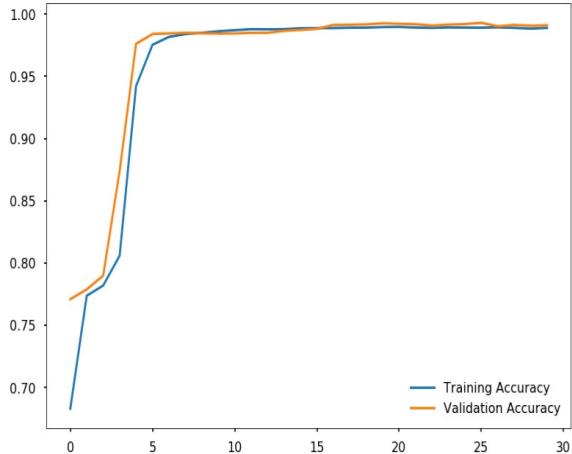


Figura 4.23: Acuratețea pentru rețeaua neuronală de calibrare

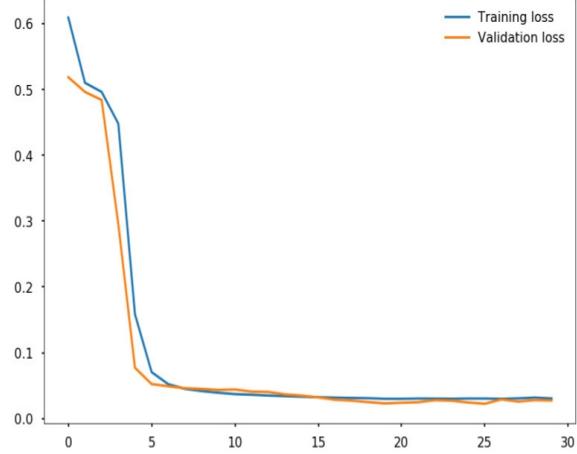


Figura 4.24: Funcțiile de pierdere pentru rețeaua neuronală de calibrare

După cum putem observa, clasificatorul nostru obține o acuratețe mai mare de 99% din cazuri atât pe setul de antrenare, cât și pe setul de validare, dar cum algoritmul nostru o să ruleze la două cadre pe secundă (2 FPS) pentru a lua o decizie pe o imagine care probabil conține mai multe detectii, rulând pe mai multe mașini, în cadrul mai multor flote de vehicule, acel 1% din erorile de predicție ale calibrării ar fi necesar a fi minimizat.

După această filtrare a detectiilor pe baza rețelei neuronale secundare o să mai fie aplicată și o condiție de viteză asupra vehiculului condus de noi, deoarece atunci când stăm pe loc nu putem penaliza sau atenționa șoferul că nu păstrează distanță.

În capitolul următor avem ocazia de a vedea rezultatele finale aplicate în viață reală.

Capitolul 5

Rezultate Finale și Concluzii

Urmează să prezentăm rezultatele finale extrase pentru evenimente din lumea reală.

5.1 Rezultate corecte ale sistemului

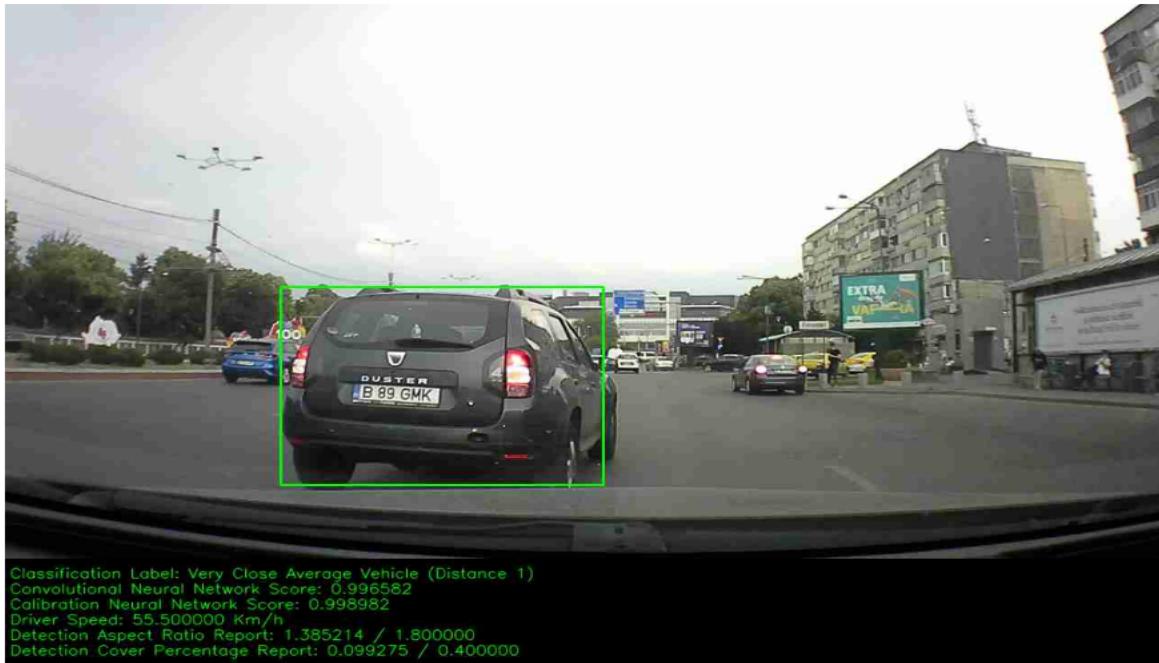


Figura 5.1: Rezultate Finale - Imaginea 1



Figura 5.2: Rezultate Finale - Imaginea 2



Figura 5.3: Rezultate Finale - Imaginea 3



Figura 5.4: Rezultate Finale - Imaginea 4



Figura 5.5: Rezultate Finale - Imaginea 5

Din ce putem observa prin rezultatele de mai sus, sistemul de estimare a distanței are capacitatea de a generaliza indiferent de diversi factori extremi, precum condiții de lumină scăzută, diverse forme și culori ale vehiculelor, reușind să estimeze distanța și în cazurile când detectiile sunt foarte depărtate.

În mod evident, sistemul de estimare a distanței prezintă ușoare erori atât în rețeaua convoluțională pentru detectie, cât și în cazul rețelei secundare pentru calibrare.

5.2 Rezultate eronate ale sistemului de estimare a distanței



Figura 5.6: Rezultate Finale - Imaginea 6 (Eroare de detectie)

După cum putem sesiza în imaginea de mai sus (Figura 5.6) există o eroare de detectie. Motivul pentru aceasta este faptul că prin exemplele de noapte folosite la antrenare, farurile vehiculelor de la distanțe apropiate sunt ușor confundabile cu reflexia luminilor pe asfalt în condiții de umiditate ridicată.

Partea bună este reprezentată de faptul că nivelul de încredere al rețelei convolutionale pentru detecție este unul de 22%. Astfel, printr-o ridicare de threshold această eroare nu ar fi luată în considerare.



Figura 5.7: Rezultate Finale - Imaginea 7 (Eroare de calibrare)

Din fericire, pentru toate aceste tipuri de erori, atât de detectie, cât și de calibrare, există posibilitatea modificării de threshold-uri, augmentării imaginilor pentru clasele care prezintă probleme la detectie, metodologii de antrenare pentru exemple puternic negative în cazul problemei de calibrare. Unele dintre ele fiind deja abordate în practică.

Pe baza celor prezentate mai sus, putem concluziona această lucrare de licență. În cadrul acestoria am reușit să dezvoltăm un sistem pentru estimarea distanței între vehicule, invariabil la diversi factori externi, bazat pe două arhitecturi de rețele neuronale. Aceasta este capabil să ruleze pe un dispozitiv încorporat cu limitări hardware, sistemul fiind specializat pentru echilibru între viteza de inferență și precizia.

Referințe

- [1] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.
- [2] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.
- [3] Wikipedia. Precision and recall — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Precision%20and%20recall&oldid=963320349>, 2020. [Online; accessed 20-June-2020].
- [4] Wikipedia. Jaccard index — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Jaccard%20index&oldid=961245353>, 2020. [Online; accessed 20-June-2020].
- [5] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.
- [6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

- [7] J. Phelawan, P. Kittisut, and Nithiroth Pornsuwancharoen. A new technique for distance measurement of between vehicles to vehicles by plate car using image processing. *Procedia Engineering*, 32:348–353, 12 2012.
- [8] Shah Nawaz. *HOG-SVM Car Detection on an Embedded GPU*. PhD thesis, 11 2015.
- [9] Naoya Ohta, Takeki Ogitsu, and Yuta Kanuki. Simple white line detection algorithm for automatic driving systems. 11 2017.
- [10] Alex Krizhevsky. Convolutional deep belief networks on cifar-10. 05 2012.
- [11] Mark Everingham, Luc Van Gool, C. K. I. Williams, J. Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge, 2010.
- [12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.
- [13] Manolis Loukarakis, José Cano, and Michael O’Boyle. Accelerating deep neural networks on low power heterogeneous architectures. 01 2018.
- [14] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.
- [15] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.
- [16] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.
- [17] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever,

Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

- [18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [19] Tzutalin. Labelimg is a graphical image annotation tool and label object bounding boxes in images. 2015.
- [20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [21] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [22] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.
- [23] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.
- [24] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.