YOVO: You Only Voxelize Once

June 15, 2020

Contents

Chapter 1

Introducere

1.1 Contextul problemei

Reconstituirea digitală a unui obiect reprezintă o problemă propusă de câteva decenii și este activ tratată în domeniul Viziunii Artificiale și al Graficii Computerizate, având ca scop final obținerea unei forme cât mai fidele al obiectelor reale.

Procedeele existente se folosesc de multiple tipuri de date pentru obținerea unei reconstrucții, acestea fiind obținute prin uzul de tehnologii precum: Camere clasice, Camere RGB-D, LiDaR, Raze X, Ultrasunete, RMN, CT etc. Considerând costul aferent fiecărei alternative prin hardware-ul dedicat necesar, se poate deduce că cea mai ieftină soluție ar necesita o simplă cameră RGB, cu viziune monotipică, pentru a crea imagini 2D. Cu toate că celelalte tipuri de date pot conduce la reproduceri volumetrice mai robuste, economisirea resurselor este un aspect ce concretizează interesul existent în soluții ce folosesc doar imagini.

Recuperarea dimensiunii a treia din poze 2D a fost țelul multor studii în ultimii ani. Prima generație de metode a tratat perspectiva geometrică din punct de vedere matematic, observând proiecția 2D al obiectelor tridimensionale, și încercând să creeze un proces reversibil. Soluțiile cele mai bune ale acestei abordări necesită multiple cadre ce capturau diferite unghiuri ale obiectelor, acestea având nevoie de o calibrare meticuloasă. A doua generație de metode a folosit o memorie ce conținea cunoștințe anterioare despre obiecte. Se poate trage o paralelă la abilitatea umana de a percepe tridimensionalitatea unui obiect cu ajutorul cunoștințelor precedente despre alte obiecte asemănătoare celui în cauză. Din acest punct de vedere, problema de reconstrucție al obiectelor 3D devine o problemă de recunoaștere. Considerând în prezent eficiența soluțiilor de Învățare Profundă pentru problemele de recunoaștere, cât și creșterea cuantumului de noi date ce pot fi folosite drept date de antrenare, se justifică tendința comunității științifice de a folosi ramuri ale Învățării Profunde precum Rețele Neuronale Convoluționale pentru obținerea geometriei și structurii 3D al obiectelor.

1.2 Soluția propusa

Aceasta lucrare abordeaza reconstructia 3D a obiectelor din imagini 2D, prin intermediul retelelor neuronale. Deoarece natura acestei probleme face parte din domeniul Viziunii Artificiale, sunt folosite preponderent Retele Neuronale Convolutionale.

Solutia propusa se numeste YOVO: You Only Voxelize Once, intrucat se proceseaza o singura imagine 2D pentru estimarea unui volum voxelizat. Arhitectura acesteia este compusa din 3 module:

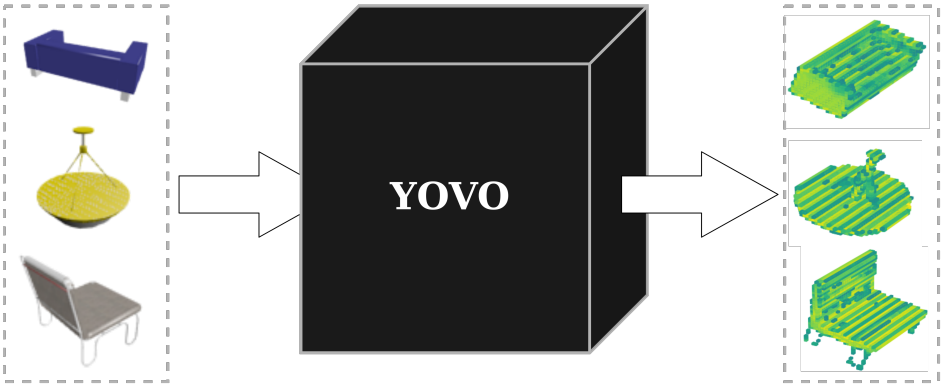


Figure 1.1: Abstractie a procesului de reconstructie 3D

- Autoencoder: format dintr-un Encoder ce extrage diferite trasaturi ale imaginii primite si un Decoder ce interpreteaza trasaturile extrase in volume voxelizate
- Agregator Contextual: realizeaza contopirea multiplelor volume deduse intr-un volum mai robust
- Rafinator: realizeaza cizelarea volumului unificat

Spre deosebire de soluții din aceeași familie precum 3D-R2N2[1], care introduce aspecte recurente in reconstructia volumului prin blocuri LSTM și GRU 3D Convoluționale, sau Pix2Vox[2], soluție care face tranzitia catre un Autoencoder complet convoluțional clasic, aceasta solutie introduce, din punct de vedere arhitectural, nivele adiționale de abstractie la ultimele 3 trepte ale Codificatorului, rezultând în 3 volume voxelizate ce capturează diferite reconstrucții ale obiectului real. Dupa trecerea prin celelalte module ale arhitecturii, rezultatul final este un singur volum. Aditional, adaugarea si integrarea procedeelor precum backbone-ul MobileNetV2, functii de activare Mish, regularizare DropBlock si optimizare Ranger, aduc rezultatele lui YOVO la nivelul SotA-ului actual pe datasetul Data3D–R2N2.

Cu ajutorul bibliotecii kaolin si matplotlib, volumele pot fi vizualizate intr-un mediu 3D interactiv, in care se poate analiza din orice unghi volumul voxelizat reconstruit. De asemenea, YOVO poate fi configurat sa produca videoclipuri de exibitie al volumelor reconstruite, sau sa le exporte in formaturi tipice aplicatilor grafice 3D.

Chapter 2

Fundamente teoretice

2.1 Inteligenta Artificiala si subdomeniile ei

Odată cu creșterea popularității soluțiilor de Inteligență Artificială s-a creat un nivel ridicat de confuzie despre cum se definește și diferențiază aceasta de alte concepte precum Învățare Automată, Învățare Profundă și Viziune Artificială. Se poate observa in figura 2.1 structura subdomeniilor Inteligenței Artificiale.

Inteligența Artificială poate fi interpretată drept încorporarea logicii umane în mașini. Aceasta include și cele mai simple exemple de soluții implementate pe un calculator, cum ar fi sistemele definite de reguli condiționale.

Învatarea Automată reprezintă un subdomeniu al Inteligenței Artificiale, reprezentând abilitatea calculatoarelor de a „învăța" să rezolve o problemă fără a avea explicit programată fiecare instrucțiune. Cu cât sistemul este mai expus la un cuantum mai mare de date, cu atât mai mult algoritmul de învatare automată se auto-reglează.

Învățarea Profundă este la rândul sau un subdomeniu al Învățării Automate și descrie o tehnică de rezolvare a problemelor cu rețele neuronale, structuri inspirate de sistemul cerebral uman. Spre deosebire de celelalte tehnici alternative, învățarea profundă necesită mai multe resurse hardware, în funcție de profunzimea și densitatea rețelelor folosite.

Viziunea Artificială este un domeniu al Informaticii ce are ca scop dezvoltarea abilităților calculatoarelor de a identifica, procesa și interpreta imaginile primite.

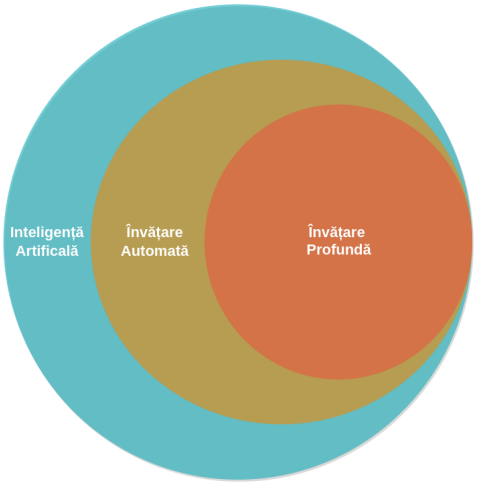


Figure 2.1: Definirea domeniilor Inteligentei Artificiale

2.2 Tipuri de Învățare Automată

Soluțiile învățării automate au nevoie de cantități semnitifcative de date pentru a putea oferi răspunsuri la probleme. Deoarece există multiple tipuri de probleme precum Clasificare, Detecție, Regresie, Reconstrucție etc., formatul în care datele aferente problemei diferă, precum si modul de abordare. Ergo, există 3 tipuri importante de Învățare Automată.

2.2.1 Învățarea supravegheată

„Învățarea supravegheată presupune învățarea unei corelări între un set de variabile de intrare X și o variabilă de ieșire Y, cât și aplicarea acestei mapări pentru a prezice ieșirile pentru date nevăzute”[3]. In alte cuvinte, această tehnică necesită date adnotate drept reper pentru predictiile facute. Pentru a asigura verosimilitatea modelului antrenat este nevoie de un set de date abundent și divers împreună cu adnotările de rigoare. Învățarea supravegheată este folosită preponderent în probleme de clasificare, detecție și regresie.

2.2.2 Învățarea nesupravegheată

Spre deosebire de tipologia anterioară, Învățarea nesupravegheată are scopul de a determina tipare prezente în setul de date, fără a avea acces la etichetări ale setului de date. Principalele tehnici aferente învățării nesupravegheate sunt clusterizarea și asocierea. În domeniul Învățării profunde, una dintre cele mai populare arhitecturi folosite sunt Auto-Codificatoarele, cu scopul de a extrage caracteristicile unei imagini și a le decodifica și readuce la dimensiunile originale. Aceste arhitecturi sunt folosite și în domeniul procesărilor de imagini.

2.2.3 Învățarea prin întărire

În aceasta tipologie este vizată recompensa interpretatorului bazată pe interacțiunea dintre mediul și agentul ce interacționează cu acesta. Învățarea prin întărire este orientată pe rezultate finale. Din aceste motive, este acceptată diversitatea de soluții pe care le aplică agentul, atâta timp cât rezultatul final este mai aroape de cel dorit. Astfel, această metodă de învățare rezolvă problema corelării imediate dintre acțiunile efectuate și rezultatele întârziate ce sunt produse de acestea. Necesitatea de a aștepta pentru a observa rezultatele finale ale acțiunilor efectuate reprezintă unul din punctele definitorii ale acestei forme de învățare automată, trăsătura asemănătoare cu efectele acțiunilor umane în mediul înconjurător.

2.2.4 Învățarea colaborativă

Această formă de învățare este o idee relativ recent adoptată în domeniul inteligenței artificiale, diferențiindu-se față de celelate 3 tipuri de învățare importante prin faptul că rețeaua neuronală este antrenată pe mai multe dispozitive, cu date diferite ce nu pot fi împărtășite intern cu alte dispozitive. Rezultatul este un sistem cu baza de date decentralizata, cu un server ce agreghează și comasează trăsăturile antrenate de dispozitive, după care reinițializează aceleași dispozitive cu o soluție îmbunătățită. Această tehnică facilitează securitatea datelor întrucât nu există o sursă centralizată a acestora.

2.3 Rețele Neuronale

Rețelele Neuronale reprezintă tehnica ce stă la baza Învățării Profunde, fiind definite printr-o structură sub formă de graf (noduri și muchii) ce poate fi asociată cu sinapsele și neuronii creierului uman. Din aceste motive, nodurile sunt referite ca neuroni, iar muchiile ca ponderi sau parametrii. Aspectul adânc al rețelelor neuronale este denotat de prezența multiplelor straturi ascunse. Este de menționat faptul că există o pondere între fiecare neuron din straturi diferite alăturate. O reprezentare simplă poate fi observată în 2.2.

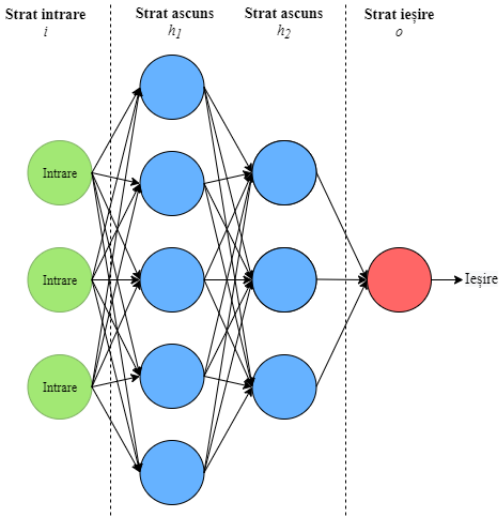


Figure 2.2: Exemplu Retea Neuronala

În mod normal, o rețea neuronală conține un strat de intrare, un număr pozitiv întreg de straturi ascunse și un strat de ieșire. Din punct de vedere matematic, ponderile sunt numere ce transformă datele de intrare în trecerea lor prin rețea, iar neuronii reprezintă funcții de activare. De asemenea, există și parametrul numit bias, cu rolul de a ajusta suma ponderată la intrarea într-un neuron.

Un strat des folosit în Rețelele Neuronale este cel dens, ce transformă datele de intrare al unui strat prin intermediul ponderilor și al funcțiilor de activare, cu formula 2.3.1.

$$\hat{y} = \sigma(Wx + b) \tag{2.3.1}$$

În formula de mai sus, $\hat{y}$ este ieșirea dintr-un neuron, σ reprezintă funcția de activare (neuronul), W sunt ponderile aferente neuronului respectiv, b este bias-ul iar x sunt datele de intrare pentru neuron. Pentru ca o rețea să deducă mărimea erorii față de rezultatul dorit, este folosită o funcție de cost. Un exemplu de funcție de cost des întâlnită este Cross-Entropia Binară, definită ca:

$$E = -\frac{1}{n}\sum_{k=1}^{n}((1 - y_k)ln(1 - \hat{y}_k) + y_k ln(\hat{y}_k)) \tag{2.3.2}$$

În 2.3.2, $\hat{y}$ este predicția rețelei, iar y este răspunsul corect. Această metrică este folosită în probleme de învățare supravegheată precum clasificarea .

Scopul rețelei este de a minimiza funcția de cost pentru a produce rezultate cât mai corecte.

2.3.1 Gradient Descent

Gradient Descent este o metodă de optimizare consacrată în Învățarea Profundă, ce are scopul de a minimiza funcția de cost prin actualizarea ponderilor modelului în direcția celei mai „abrupte" pante.

Considerând 2.3.1, putem dezvolta în:

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n b) \tag{2.3.3}$$

Pentru a actualiza ponderile, avem nevoie de gradientul funcției de eroare, definit prin:

$$\nabla E = (\frac{\delta E}{\delta w_1}, \frac{\delta E}{\delta w_2}, \cdots, \frac{\delta E}{\delta w_n}, \frac{\delta E}{\delta b}) \tag{2.3.4}$$

Într-un final, definim o rată de învățare α. Având toate valorile acestea, se poate efectua un pas de Gradient Descent:

$$w_k^* = w_k - \alpha \frac{\delta E}{\delta w_k} \tag{2.3.5}$$

$$b^* = b - \alpha \frac{\delta E}{\delta b} \tag{2.3.6}$$

unde w_k^* si b^* sunt noile valori ale ponderii w_k și bias-ului b.

2.3.2 Feedforward

În Rețele Neuronale, procesul de Feedforward este folosit pentru a transforma datele de intrare în date de ieșire ale rețelei. Pentru a putea defini mai bine acest concept, fie structura 2.3 cu funcția de activare σ, ponderile $W^{(i)}$pentru stratul i și predicția sistemului $\hat{y}$.

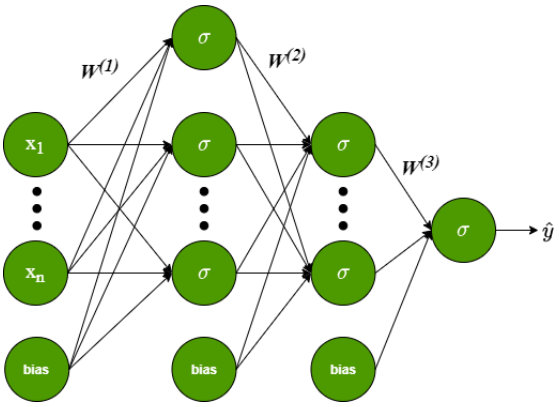


Figure 2.3: Retea Neuronala cu multiple straturi ascunse

Aplicând 2.3.1pentru toate straturile din 2.3, avem formula:

$$\hat{y} = \sigma \bigcirc W^{(3)} \bigcirc \sigma \bigcirc W^{(2)} \bigcirc \sigma \bigcirc W^{(1)}(x) \tag{2.3.7}$$

cu $\bigcirc$ drept operatorul pentru compoziția de funcții.

2.3.3 Backpropagation

Metoda folosită în Rețele Neuronale pentru a calcula gradienții funcției de cost necesari metodelor de optimizare se numește Backpropagation. Aceasta are la bază folosirea Regulii Lanțului pentru a calcula derivatele parțiale δ ale unor funcții compuse. Pentru $A = f(x)$ și $B = g \circ f(x)$, Regula Lanțului definește:

$$\frac{\delta B}{\delta x} = \frac{\delta B}{\delta A}\frac{\delta A}{\delta x} \tag{2.3.8}$$

Prin folosirea recursivă a acestei reguli, Backpropagation reușește „să producă o expresie algebrică pentru gradientul unui scalar, cu respect la fiecare nod din graful computațional produs de acel scalar”[4].

2.3.4 Framework-uri

Un framework pentru Învățarea Profundă este o bibliotecă ce simplifică procesul de creare al unui model prin folosirea unor sintaxe mai inteligibile. Acestea sunt create pentru a interpreta și optimiza interacțiunea dintre utilizator și dispozitivele de calcul precum Plăci Video sau Procesoare. De exemplu, un framework va integra transformările și operațiile cu matrici, va face posibilă paralelizarea procesului computațional sau va ușura vizualizarea și înregistrarea de parametri. Două dintre cele mai populare framework-uri sunt:

1. Tensorflow[5]: oferă instrumente utile precum Tensorboard pentru vizualizarea clară a pipeline-ului unui model, permite RT-serving, un proces de optimizare semi-automată al modelelor prin tehnici precum prunizare, cuantizare, împărtășire de ponderi. Din aceste motive, Tensorflow aduce avantaje în crearea de modele pentru producție, dar are dezavantajul de a prezenta o sintaxă mai dificilă și complexă.

2. Pytorch[6]: bazat pe biblioteca Torch, oferă claritate și ușurință în folosire, având tendințe „Pythonice". De asemenea, este un framework ce simplifică procesul depanării de cod, având avantajul unei sintaxe ușoare, experimentele fiind ușor de realizat. Din aceste motive, comunitatea academică tinde să folosească acest framework la scara largă.

2.4 Rețele Neuronale Convoluționale

„Rețelele Neuronale Convoluționale sunt un tip de învățare profundă pentru a procesa date ce au structura în formă de grilă, precum imaginile, ce este inspirată de organizarea cortexului vizual și este proiectată să învețe în mod automat și adaptabil ierarhii spațiale de caracteristici, de la tipare de nivel mic la cele de nivel mai mare. Rețelele Neuronale Convoluționale sunt un crez matematic ce este compus de trei tipuri de straturi (sau blocuri): convoluție, pooling și strat dens.”[7]

Convoluția este o operație matematică care, în domeniul Viziunii Artificiale și a Învățării profunde este folosită pentru a aplica filtre asupra imaginilor și de a antrena aceste filtre. În urma aplicării unui set de filtre, rezultatul este un ansamblu de hărți de caracteristici(canale). Acest ansamblu se mai numește strat. Din punct de vedere matematic, aceste sunt reprezentate în formă de tenzori, ce reprezintă o formă generalizată a unei matrici. Tenzorii pot fi atat 0-dimensionali (un singur număr), cât și pluridimensionali.

2.4.1 Convoluții Multidimensionale și Terminologii

Un strat conține o multitudine de canale. Kernel-ul este o matrice de ponderi ce se înmulțește cu porțiuni ale stratului de intrare, valorile rezultate fiind însumate într-o singură valoare finală reprezentativă pentru porțiunea respectivă. Un filtru de convoluție reprezintă un ansamblu de kernel-uri stivuite. Pasul unei convoluții reprezintă din câte în câte puncte se aplică filtrele. Folosirea unui pas mai mare de 1 duce la reducerea exponențială a dimensionalității. O reprezentare a diferitelor tipuri de kerneluri este prezentată în figura2.4.

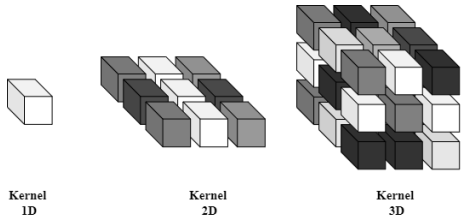


Figure 2.4: Kernel-uri de convolutie

Pentru aplicarea unei convolutii 2D, un kernel este aplicat fiecarui canal, in adancimea stratului, iar rezultatele sunt adunate si agregate intr-un nou canal. Numarul de kerneluri din filtru reprezinta numarul de canale pe care il va rezulta convolutia. Convolutiile 2D pot deplasa filtrele doar pe inaltimea si latimea stratului. Figura2.5 prezinta o convolutie simpla, cu un kernel de 2x2 si 6 canale de iesire. Convolutiile 3D contin kernel-uri 3D, ce se aplica independent in adancimea stratului de intrare. Astfel, convolutiile 3D deplasa filtrele atat pe inaltimea si latimea stratului, cat si pe adancimea acestuia.

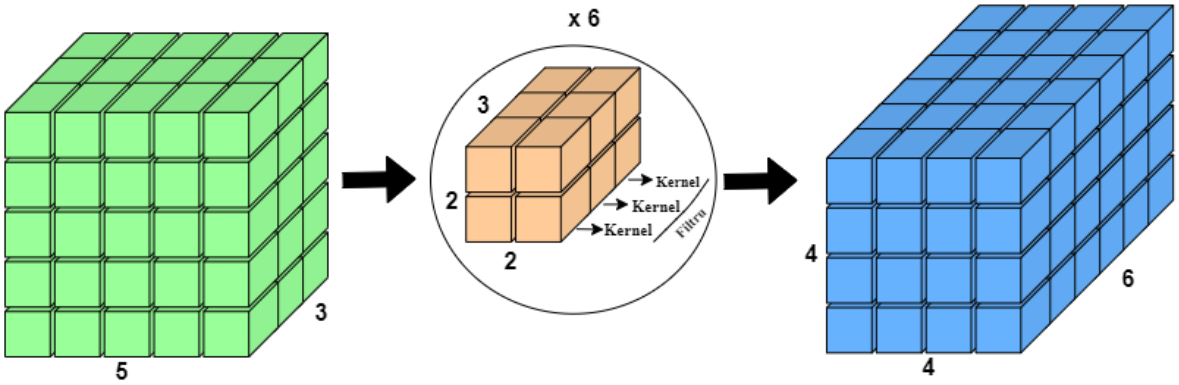


Figure 2.5: Convolutie 2D cu 6 canale de iesire

Operatia opusa convolutiei, din punct de vedere al dimensionalitatii, este convolutia transpusa. Daca o convolutie clasica are ca rezultat un strat cu dimensiuni egale sau mai mici decat cele ale stratului de intrare, convolutia transpusa are scopul de a mari stratul de intrare, prin adaugarea de padding sau prin dilatarea stratului de intrare.

In aceasta lucrare ne vom referi la reducerea sau cresterea dimensionalitatii drept downsampling, respectiv upsampling.

De asemenea, operatia de convolutie se poate aplica cu pas diferit, pentru a reduce dimensionalitatea datelor. Pentru a mari dimensionalitatea, se folosesc convolutii transpuse.

2.4.2 Convolutii separabile

Un alt concept folosit in diverse arhitecturi este eficientizarea operatiei de convolutie prin convolutii separabile[8]. Acestea impart procesul unei convolutii intr-o convolutie depthwise, ce doar reduce dimensiunile stratului de intrare, si alta pointwise, ce prelungeste numarul de canale. Rezultatul este un numar substantial redus de operatii. Un exemplu este prezentat in figura 2.6 drept alternativa la convolutia prezenta in figura 2.5.

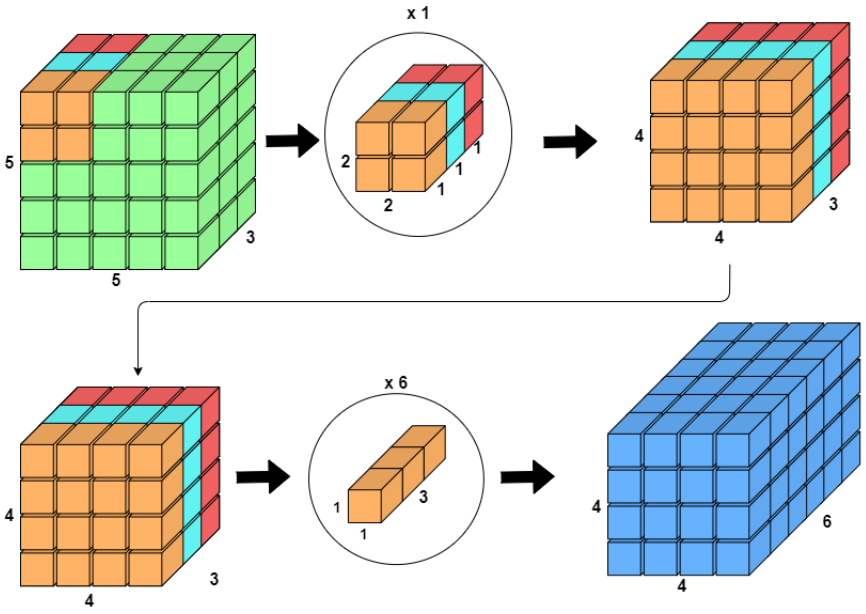


Figure 2.6: Convolutie separabila

In cazul din figura2.5, pe stratul de intrare se aplica filtre de $2 \times 2 \times 3$, de 4×4 ori pentru a avea un singur strat de iesire. Se aplica de 6 ori pasii precedenti, dimensiunile rezultatului final avand $4 \times 4 \times 3$. In total avem $2 * 2 * 3 * 4 * 4 * 6 = 1152$ de inmultiri. Aplicand convolutia separabila prezentata la 2.6, pe stratul de intrare se folosesc 3 filtre de $2 \times 2 \times 1$, de 4×4 ori. Acest pas se numeste convolutie depthwise si efectueaza doar o singura data. Numarul de operatii pentru aceasta tehnica este $3 * 2 * 2 * 1 * 4 * 4 = 192$ de inmultiri. Rezultatului intermediar dupa convolutia depthwise ii este aplicata o convolutie pointwise, cu un filtru de $1 \times 1 \times 3$, de 4×4 ori. Acest pas este repetat de 6 ori. In total, convolutia pointwise prezinta $1 * 1 * 3 * 4 * 4 * 6 = 288$ de inmultiri. Astfel, convolutia separabila are in total $192 + 288 = 480$ de inmultiri fata de cele 1152 ale unei convolutii clasice. Aceasta diferenta devine mai semnificativa la date de intrare mari. Dezavantajul convolutilor separabile este reducerea numarului de ponderi din convolutie, posibil limitand capacitatea de invatare fata de o convolutie normala. Cu toate acestea, eficienta rezultata din convolutiile separabile compenseaza pentru potentialele aspecte negative.

2.4.3 Blocuri Reziduale

Un strat Rezidual clasic, prezentat in [9][10], este format dintr-o convolutie 2D cu kernel 1x1 ce reduce numarul de filtre, o a doua convolutie 2D cu kernel 3x3 ce nu reduce dimensionalitatea, urmat de a 3-a convolutie 2D cu kernel 1x1 ce readuce numarul de filtre la cel initial, elementele iesirii ultimei convolutii fiind adunate cu elementele intrarii in din prima convolutie. Aceasta conexiune dintre Input si Output se numeste skip connection, si se poate aplica doar cand dimensiunile acestora sunt identice. Toate convolutile au ReLU ca functie de activare. Blocul

rezidual este reprezentat in figura 2.7.

Aceste blocuri au fost create pentru a combate problema gradientilor cu valori foarte mici. Deoarece acest block nu adauga un nou nivel de abstractie, ci doar rafineaza informatia, folosirea a mai multor blocuri reziduale pot spori puterea computationala, astfel modelul reusind sa invete mai bine caracteristicile. Aceast concept poate fi vazut ca si cum iesirea din aceasta structura poate fi influentata doar de caracteristicile intermediare, dar nu poate fi anulata. Denumirea „reziduala" provine din faptul iesirea din acest bloc este egala cu intrarea plus informatia reziduala captata.

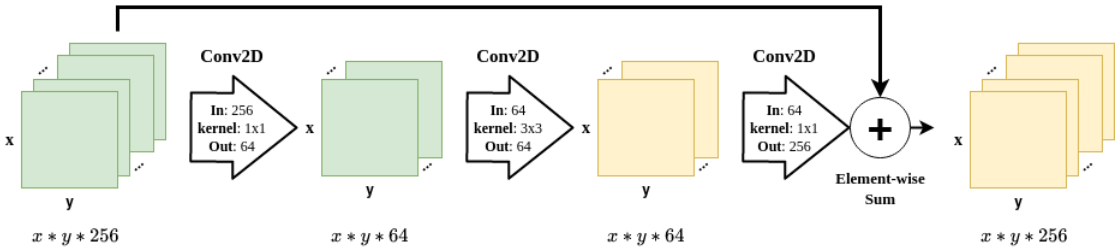


Figure 2.7: Bloc Rezidual

2.4.4 Blocuri Reziduale Inversate cu Linear Bottleneck

Introduse in [11], acestea au pornit de la premiza ca hartile de caracteristici pot fi codificate in straturi cu dimensiune redusa (cu mai putine canale) si ca activarile non-liniare (e.g. ReLU) provoaca pierderi de informatie, chiar daca au abilitatea de reprezentare a complexitatii datelor. In aceasta structura sunt folosite activari liniare $f(x) = x$ (functia identitate) si activari non-liniare ReLU6, ce sunt defapt activari ReLU cu limita superioara 6.

Acest bloc primeste ca un input un tensor cu putine canale si aplica in primul rand o convolutie pointwise ce creste numarul de canale pentru a se pregati urmatoarelor operatii non-liniare, ce necesita aceasta crestere in dimensiunea tensorului. Dupa o activare ReLU6 este aplicata a doua convolutie: una depthwise cu kernel de 3×3, urmata de o alta activare ReLU6, cu rolul de a filtra dimensiunile marite ale tensorului. A treia convolutie este una pointwise ce proiecteaza numarul mare de canale intr-o dimensiune redusa, egala cu cea a datelor de intrare. Intrucat dupa a treia convolutie am revenit la spatiul redus din punct de vedere dimensional, aplicarea unei activari non-liniare ar cauza prea multa pierdere de informatii. In consecinta, se foloseste activarea liniara $f(x) = x$. Intr-un final, daca nu se doreste reducerea dimensiunilor principale ale hartilor de caracteristici (lungimea si latimea), este efectuat un skip connection dintre input si output. In caz contrar, nu este aplicata o asemenea conexiune. Structura acestui bloc este prezentata in figura 2.8.

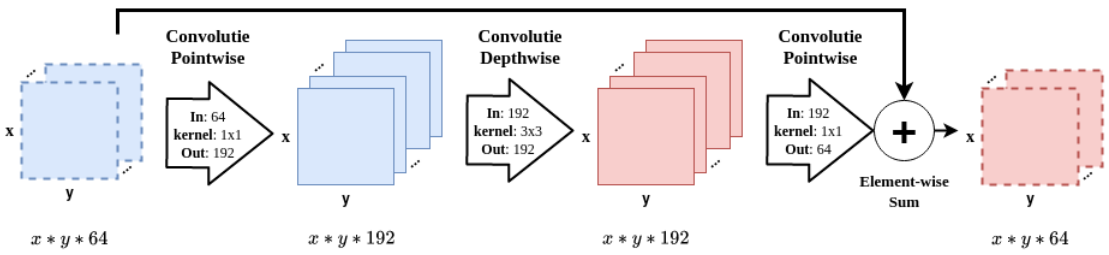


Figure 2.8: Block Rezidual Inversat cu Linear Bottleneck

2.4.5 Alte tipuri de straturi des folosite

Unul dintre cel mai comun strat folosit pentru reducerea dimensionalitatii este stratul de pooling. Acesta aplica la randul lui un filtru de kernel-uri pe datele de intrare. In functie de tipul stratului de pooling, aceste kerneluri pot comasa datele de intrare dupa valorile medii, sau dupa valoriile maximale.

O caracteristica ce poate fi asociata unei multimi de straturi este padding-ul, ce extinde dimensiunile straturilor. Aceasta extindere se poate realiza cu valori nule sau cu media valorilor vecine. Aplicarea padding-ului poate evita reducerea dimensionalitatii la operatii cu kernel.

Un alt strat des folosit in Retelele Neuronale Convolutionale este cel de Batch Normalization, pentru cresterea stabilitatii. Acesta este aplicat dupa functia de activare si functioneaza prin normalizarea valorilor de intrare x. Valorile normalizate $\hat{x}$ sunt obtinute prin scaderea mediei lotului respectiv si impartirea la deviatia standard a lotului. Normalizarea are efectul de a preveni situatile in care o activare produce valori foarte mari sau foarte mici. Pentru a evita potentialele situatii destabilizante ale acestei normalizari, se adauga doi parametrii antrenabili: $\gamma - (gamma)$ pentru deviatia standard si $\beta - (beta)$ pentru medie. Formula simplificata a stratului de Batch Normalization este:

$$y = \gamma * \hat{x} + \beta \tag{2.4.1}$$

Astfel, in cazul in care un optimizator trebuie sa denormalizeze valorile, se vor modifica doar acesti parametri. De asemenea, folosirea acestui strat anuleaza valorile de bias din stratul respectiv, acestea fiind echivalente cu β.

Un alt tip de straturi folosite sunt cele de regularizare. Acestea sunt definite de metoda de regularizare aleasa, si au rolul de a evita cazurile in care o retea memoreaza setul de date si nu poate generaliza pe date noi.

2.4.6 Backbone

Backbone-ul are rolul de a extrage caracteristicile dintr-o imagine si este structurat prin suprapunerea de straturi de convolutie, pooling si alte operatii. Acesta este deobicei folosit pentru transferul de cunostiinte, intrucat un backbone preantrenat sa detecteze caracteristici generale poate imbunatati considerabil viteza de invatare a unei retele. Printre cele mai populare arhitecturi se numara AlexNet, VGG si ResNet.

2.4.7 Arhitecturi folosite in Retele Neuronale Convolutionale

Primele arhitecturi folosite in domeniu aveau structura secventiala. Aparitia solutiilor moderne a adus o multitudine de structuri arhitecturale ce reusesc sa extraga informatia captata in imagini si sa mentina caracteristici atat mai generale, cat si mai concrete.

Una dintre aceste structuri este Autoencoder-ul[12], format dintr-un Encoder, ce reduce dimensionalitatea input-ului si are rolul de a codifica o reprezentare a acestuia. A doua parte a acestei structuri este un Decoder, ce creeaza o reproducere cat mai fidela a input-ului bazata pe rezultatul Encoder-ului. Astfel, caracteristicile extrase isi pastreaza aceeasi dimensionalitate ca si imaginea originala. O variatiune a acestuia este U-Net[13], ce adauga conexiuni „skip” intre nivelele Encoder-ului si al Decoder-ului, pentru a pastra caracteristici pierdute in procesul de reducere.

O alta arhitectura este Feature Pyramid Network[14], ce prezinta 2 structuri piramidale paralele ce extrag si transmit caracteristici. Un aspect important al acestei structuri sunt trans-

miterea caracteristicilor prin conexiuni „skip” intre nivele paralele ale piramidelor, cat si predictiile multi-level ce permit procesarea caracteristicilor la diferite nivele de complexitate.

2.4.8 Metode de Augmentare

Pentru ca o retea neuronala sa aibe o performanta semnificativa este nevoie de un set de date cat mai mare si divers, astfel permitand modelului produs sa generalizeze pe date noi. Cu toate acestea, nu toate seturile de date prezinta varietate, permitand utilizatorului sa aplice metode de augmentare pentru a spori diversitatea. Printre tehnicile de augmentare se numara:

1. Decuparea - se elimina mai multe randuri de pixeli. Decuparea poate fi centrata, laterala sau aletorie.
2. Redimensionarea - modificarea dimensiunilor imaginii afectand marimea obiectelor prezente. Aceasta augmentare poate produce pierdere de informatii intrucat se produc pierderi la comasarea pixelilor.
3. Variatia fundalului - pentru seturile de date cu fundalul monocrom si usor de izolat, se pot folosi game diferite de culori. Aceasta metoda de augmentare este folosita si pentru datele de intrare compozite, ce se formeaza in timpul executiei programului.
4. Bruiaj cromatic - modificarea luminozitatii, contrastului si saturatiei unei imagini
5. Inducerea de zgomot in imagine - este variatiunea aleatorie a culorii pixelilor unei imagini. Acest zgomot poate varia in functie de functia densitatii de probabiltate.
6. Normalizare - o imagine contine informatii ale pixelilor pe cele 3 canale RGB, cu valori intre 0-255. Normalizarea aduce acele valori intr-un interval potrivit retelei in cauza. In domeniul Retelelor Neuronale, normalizarea si standardizarea denota acelasi proces, termenii fiind folositi interschimbabil.
7. Rotatie - rotirea unei imagini cu un numar de grade variabil.
8. Flip - imaginea este intoarsa 180 de grade in jurul axei orizontale sau verticale.
9. Permutarea canalelor de culoare - se interschimba valorile canalelor de culoare RGB (rosu-verde-albastru)

2.5 Aspectele unei retele

Pentru a produce un model capabil de performante satisfacatoare, trebuie definit procesul de antrenare, cel de validare in timpul antrenarii, cat si procesul de testare si evaluare al performantei.

2.5.1 Hiperparametrii

In aplicatiile de Invatare Profunda, exista parametrii simpli, ce contin informatia invatata de retea. Acesti parametrii pot fi antrenabili si sunt reprezentati de ponderi, bias, gamma si beta (Batch Normalization), etc.

De asemenea, exista si hiperparametrii, ce definesc procesul de antrenare si sunt setati inaintea inceperii acestuia. Printre cei mai consacrati hiperparametrii se numara:

- Rata de invatare (lr) - aplicata in tehnica de optimizare aleasa.
- Functiile de Activare - guverneaza cum sunt procesate sau filtrate ponderile.
- Parametrii Optimizatorului - initializeaza si seteaza anumite limite in procesul de optimizare. Un optimizator are scopul de a micsora functia de cost.
- Marimea lotului ($batch\_size$) - Deoarece este ineficient si costisitor sa parcurgem intreg setul de date ca sa efectuam un pas de optimizare, acesta este divizat in loturi, asigurand o oarecare invatare a cazurilor neobisnuite care altfel nu ar fi avut niciun impact comparativ cu restul datelor.
- Numarul de epoci - O parcurgere a intregului set de date este definit ca o epoca. Retelele Neuronale necesita multiple parcurgeri pentru a deveni eficiente.
- Parametrii arhitecturali - Numarul de blocuri structurale folosite in arhitectura sau numarul de straturi ascunse

Procesul de alegere a valorilor optime se numeste Reglarea Hiperparametrilor. In mod normal acestia sunt alesi arbitrar, sau determinati empiric. Cu toate acestea, in ultimii ani au aparut metode de automatizare a acestui procesului de Reglare Hiperparametrica, precum AutoML[15]. Aceste metode de automatizare reprezinta o solutie accesibila pentru antrenarea si lansarea in productie a aplicatilor cu retele neuronale, dar prezinta limitari in domeniul academic, ce are scopul de a depasi solutiile State-of-the-Art.

2.5.2 Functii de Activare

Functia de activare reprezinta o ecuatie matimatica ce determina valorie de iesire. Din punct de vedere matematic, pentru aplicarea algoritmului de Gradient Descent, acestea sunt de preferat sa fie differentiabile pe tot domeniul de valori. Printre cele mai reprezentative functii de activare se numara:

1. Sigmoid2.5.1 - folosita in general pentru clasificarea binara, intrucat rezultatul acestei functii are valori intre 0 si 1. Reprezinta un caz particular al functiei Softmax2.5.2. Pentru ambele functii, comonentele au suma 1.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.5.1}$$

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{2.5.2}$$

2. ReLU2.5.3(Rectified Linear Unit) - introdus in solutiile de invatare profunda de [16], a devenit cea mai populara functie de activare, prin rezolvarea problemei disparitiei gradientului si rapiditatii computationale. Aceasta functie este diferentiabila in toate punctele exceptand $x = 0$, intrucat o derivata are valoarea 1 iar cealalta are valoarea 0. Pentru a evita posibile complicatii, in general este convenit ca derivata in $x = 0$ este cea de pe a doua ramura. Exista variatii de ReLU in care sunt impuse limite superioare (e.g. Relu6).

$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & otherwise \end{cases} \tag{2.5.3}$$

3. Leaky-ReLU2.5.4(Leaky Rectified Linear Unit) - introdus de [17], aduce o solutie la problemele in care activarile ReLU „mor", adica tind spre 0. Acest lucru se realizeaza prin tratarea valorilor negative cu o pondere mica, valori ce in mod normal nu ar fi tratate de ReLU.

$$LeakyReLU(x) = \begin{cases} x, & x > 0 \\ 0.01x, & otherwise \end{cases} \tag{2.5.4}$$

4. ELU2.5.5(Exponential Linear Unit) - introdus de [18] si spre deosebire de activarile precedente, aceasta functie realizeaza mai bine tranzitia de la valori pozitive la valori negative, adaugand un hiperparametru α, cu valoarea de baza 1.

$$ELU(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & otherwise \end{cases} \tag{2.5.5}$$

5. Swish - creata de [19], trateaza intr-o maniera mai buna problemele ce apar in ReLU, diferentiandu-se de celelalte activari prin faptul ca functia este diferentiabila in toate punctele si nu este monotona, avand limita inferioara in punctul de inflexiune al functiei. De asemenea, functia nu este limitata superior si prezinta un parametru β, ce poate fi constant sau antrenabil. Cand β este 0, functia devine liniara, iar cand acesta tinde spre ∞, functia devine ReLU. Valoarea de baza al lui β este 1.

$$Swish(x) = x * Sigmoid(\beta x) \tag{2.5.6}$$

6. Mish - descrisa de [20] si inspirata de [19], aceasta functie incearca sa rezolve aceleasi probleme dar reuseste sa evite mai eficient saturarea functiei in valorile negative, problema existenta in functiile prezentate. De asemenea, este diferentiabila in toate punctele. Fiind relativ recent descoperita, analize comparative sunt realizate si in prezent.

$$Mish(x) = x * tanh(ln(1 + e^x)) \tag{2.5.7}$$

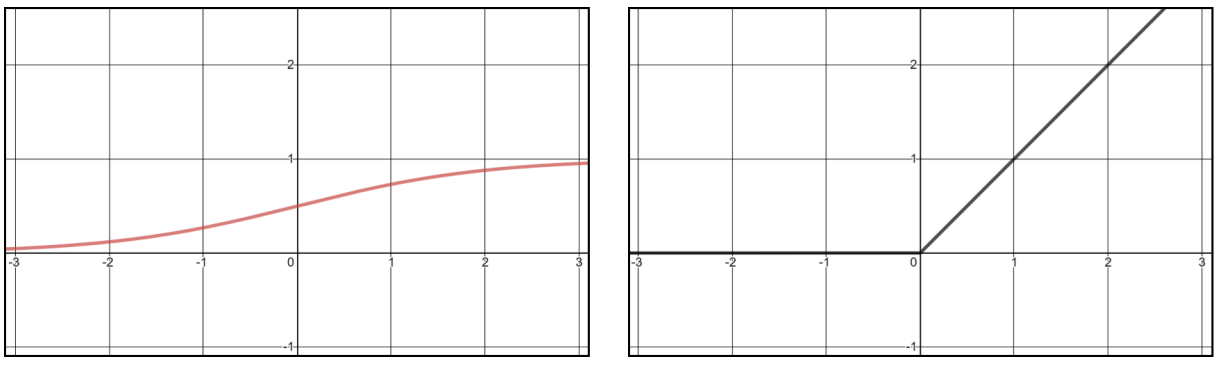


Figure 2.9: Sigmoid (stanga) si ReLU(dreapta)

2.5.3 Functii de Cost

In domeniul Invatarii Profunde, functiile de cost sunt folosite in procesul de optimizare pentru a calcula eroarea modelului, fapt pentru care valoarea acestora este referita drept „eroarea modelului". Cea folosita in aceasta lucrare se numeste Cross-Entropie Binara Voxel-Unitara.

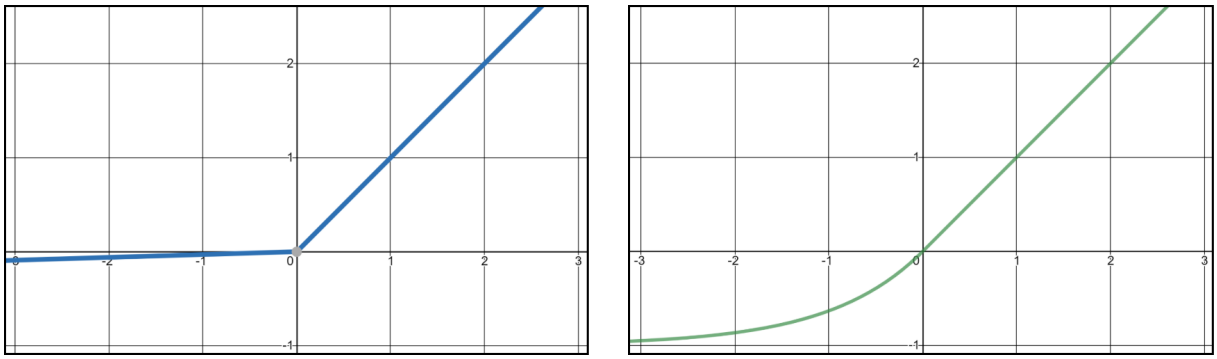


Figure 2.10: Leaky-ReLU (stanga) si ELU(dreapta)

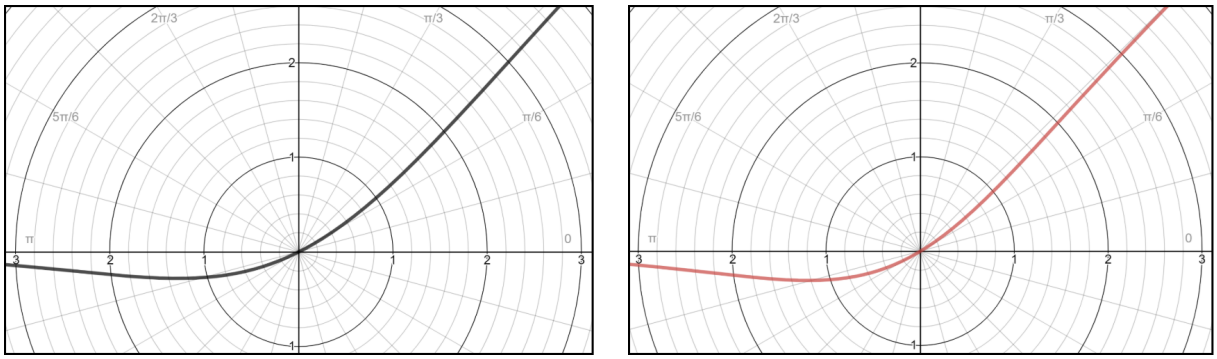


Figure 2.11: Swish (stanga) si Mish(dreapta)

Un voxel reprezinta cea mai minuscula unitate ce poate fi recunoscuta intr-un sistem, echivalentul 3-Dimensional al unui pixel.

Considerand o pozitie in spatiul 3D cu un volum echivalent unui voxel, avem doua evenimente posibile: exista sau nu exista un voxel in pozitia respectiva. Mai departe, definim ocupanta drept probabilitatea unei pozitii de a fi ocupata cu un voxel. Avand un set de valori probabilistice pentru ocupante si un set de evenimente, ambele seturi pentru toate punctele din spatiul 3D existent, Cross-Entropia denota cat este de probabil ca acele evenimente sa se intample bazandu-se pe probabilitatea evenimentelor. Daca este foarte probabil, avem o Cross-Entropie mica. In caz contrar, Cross-Entropia este mare. Formula pentru Costul Cross-Entropic Binar Voxel-Unitar este:

$$L = -\frac{1}{N}\sum_{k=1}^{N}((1 - gt_k)ln(1 - o_k) + gt_k ln(o_k)) \tag{2.5.8}$$

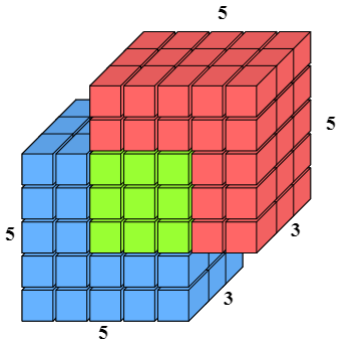
unde N este numarul de voxeli din volumul real, o_k reprezinta ocupanta pentru pozitia k iar gt_keste valoarea reala pentru pozitia k. Din cauza naturii binare a lui gt_k, termenul din interiorul sumei poate fi $ln(1 - o_k)$ sau $ln(o_k)$. In consecinta, putem defini Cross-Entropia din punct de vedere matematic drept negativul sumei logaritmilor naturali ale probabilitatilor aferente evenimentelor reale.

2.5.4 Metrici

In scopul reconstructiei 3D a obiectelor cu voxeli, metrica uzuala este 3D-Intersection-over-Union, sau 3D-IoU. Formula acesteia este:

$$3DIoU = \frac{V_{reconstruit} \cap V_{real}}{V_{reconstruit} \cup V_{real}} \tag{2.5.9}$$

unde $V_{reconstruit}$ si V_{real} este volumul prezis de retea, respectiv volumul real pentru obiect. Un exemplu poate fi observat in figura2.12.



$$3DIoU = \frac{3*3*3}{5*5*3+5*5*3-3*3*3} = \frac{27}{123} \approx 0.22$$

Figure 2.12: Exemplu de 3D-IoU a 2 cuburi de $5 \times 5 \times 3$

2.6 Optimizatori Moderni

In domeniul Invatarii Profunde, optimizatorul este algoritmul ce actualizeaza paramentrii antrenabili ai unei retele neuronale, reducand valoarea functiei de cost, cu scopul de a ajunge in minimul global al functiei. Alegerea optimizatorului si ajustarea hiperparametrilor acestuia este un aspect foarte important, intrucat este dorita trecerea peste punctele de minim local ale functiei, cat si o convergenta cat mai rapida la valoarea dorita.

2.6.1 Cele 3 variatii ale Gradient Descent

Concretizata in domeniul Invatarii Automate de [21], Stochastic Gradiend Descent, sau SGD, este o aproximare stohastica a metodei de optimizare Gradient Descent. Pentru fiecare exemplu din setul de date, SGD calculeaza gradientul erorii si actualizeaza ponderile. Aspectul stohastic provine din faptul ca aceasta metoda estimeaza valoarea gradientului in functie de exemple aleatorii ale setului de date. Batch Gradient Descent se diferentiaza de SGD prin faptul ca eroarea este calculata pentru fiecare exemplu din setul de date, dar ponderile sunt actualizate abia dupa ce toate datele au fost iterate. O alta variatiune a algoritmului de Gradient Descent este Mini-batch Gradient Descent. Deoarece calcularea gradientilor pentru fiecare exemplu poate fi costisitor din punct de vedere computational, dar in acelasi timp memorarea gradientilor pentru intreg setul de date poate fi costisitor din punct de vedere al resurselor de stocare, Mini-batch Gradient Descent imparte setul de date in mai multe loturi, calculeaza eroarea modelului pentru acel lot si actualizeaza ponderile aferente. Aplicarea pasului de Gradiend Descent este decris in 2.3.1.

2.6.2 Momentum

Momentum[22] reprezinta o imbunatatire a algoritmului de Gradient Descent, ce ia in considerare gradientii precedenti si decide nivelul de contributie al acestora cat si al gradientului actual, folosind un parametru β - factorul de descompunere exponențiala. Cu ajutorul acestui

parametru se poate spune ca o medie locala exponential ponderata a gradientilor este calculata pentru actualizarea ponderilor modelului. Algoritmul este prezentat in 2.1.

Algoritm 2.1 Actualizarea ponderilor folosind Gradient Descent cu Momentum

pentru iteratia k:

calculeaza gradientii $\nabla E_k(\omega)$ si $\nabla E_k(b)$ al lotului k
calculeaza mediile locale $\Delta\omega$ si Δb:

$\Delta\omega \leftarrow \beta\Delta\omega + (1-\beta)\nabla E_k(\omega)$
$\Delta b \leftarrow \beta\Delta b + (1-\beta)\nabla E_k(b)$

actualizeaza ponderile ω^* si b^*:

$\omega^* \leftarrow \omega - \alpha\Delta\omega$
$b^* \leftarrow b - \alpha\Delta b$

sfarsit pentru

2.6.3 RMSProp

Acronim pentru Root Mean Square Propagation, RMSProp[24] imparte rata de invatare cu media locala a magnitudinilor gradientilor precedenti pentru ponderea in cauza. Se foloseste termenul γ ca factor de uitare si ϵ drept scalar pentru prevenirea imartirii cu 0, operatiile efectuandu-se element-wise.Algoritmul este prezentat in 2.2.

Algoritm 2.2 Actualizarea ponderilor cu optimizatorul RMSProp

pentru iteratia k:

calculeaza gradientii $\nabla E_k(\omega)$ si $\nabla E_k(b)$ al lotului k
calculeaza mediiile locale ale magnitudinii gradientilor $\Theta\omega$ si Θb:

$\Theta\omega \leftarrow \gamma\Theta\omega + (1-\gamma)(\nabla E_k(\omega))^2$
$\Theta b \leftarrow \gamma\Theta b + (1-\gamma)(\nabla E_k(b))^2$

actualizeaza ponderile ω^* si b^*:

$\omega^* \leftarrow \omega - \alpha\frac{\nabla E_k(\omega)}{\sqrt{\Theta\omega}+\epsilon}$
$b^* \leftarrow b - \alpha\frac{\nabla E_k(b)}{\sqrt{\Theta b}+\epsilon}$

sfarsit pentru

2.6.4 Adam

Adaptive Moment Estimation, denumit si Adam, reprezinta aplicare algoritmilor Momentum si RMSProp pe Gradient Descent. De asemenea, Adam efectueaza corectie de bias, ce are rolul de a atenua erorile in pasii incipienti antrenarii. Se pastreaza sintaxa prezentata mai sus. Algoritmul este prezentat in 2.3.

Algoritm 2.3 Actualizarea ponderilor cu optimizatorul Adam

pentru iteratia k:

calculeaza gradientii $\nabla E_k(\omega)$ si $\nabla E_k(b)$ al lotului k

calculeaza $\Delta\omega, \Delta b, \Theta\omega$ si Θb:

$\Delta\omega \leftarrow \beta\Delta\omega + (1-\beta)\nabla E_k(\omega)$

$\Delta b \leftarrow \beta\Delta b + (1-\beta)\nabla E_k(b)$

$\Theta\omega \leftarrow \gamma\Theta\omega + (1-\gamma)(\nabla E_k(\omega))^2$

$\Theta b \leftarrow \gamma\Theta b + (1-\gamma)(\nabla E_k(b))^2$

efectueaza corectie de bias:

$\Delta\omega^{corect} \leftarrow \frac{\Delta\omega}{1-\beta^k}$

$\Delta b^{corect} \leftarrow \frac{\Delta b}{1-\beta^k}$

$\Theta\omega^{corect} \leftarrow \frac{\Theta\omega}{1-\gamma^k}$

$\Theta b^{corect} \leftarrow \frac{\Theta b}{1-\gamma^k}$

actualizeaza ponderile ω^* si b^*:

$\omega^* \leftarrow \omega - \alpha\frac{\Delta\omega^{corect}}{\sqrt{\Theta\omega^{corect}}+\epsilon}$

$b^* \leftarrow b - \alpha\frac{\Delta b^{corect}}{\sqrt{\Theta b^{corect}}+\epsilon}$

sfarsit pentru

returneaza parametrii ω^*, b^*

2.6.5 Rectified-Adam

O versiune imbunatatita fata de optimizatorul Adam, RAdam[26] aduce un grad de adaptabilitate ratei de invatare α. In retelele neuronale, exista riscul ca optimizatorul sa convearga catre minime locale, demers ce poate aparea inca de la inceputul antrenarii si rezulta intr-o varianta crescuta. De aceea, autorii au descris o perioada de „incalzire" la inceputul antrenarii pentru rate de invatare, in care aceasta are valori scazute.

Calculand valoarea maxima a mediei locale exponential ponderata si corectata ρ_∞ cu formula 2.6.1, valoarea aproximata a mediei locale exponential ponderate si corectate ρ_k la iteratia k cu formula 2.6.2, se defineste termenul de rectificare r_t pentru iteratia k cu formula 2.6.3.

$$\rho_\infty = \frac{2}{1-\beta} - 1 \tag{2.6.1}$$

$$\rho_k = \rho_\infty - \frac{2k\beta^k}{1-\beta^k} \tag{2.6.2}$$

$$r_k = \sqrt{\frac{(\rho_k - 4)(\rho_k - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_k}} \tag{2.6.3}$$

Termenul de rectificare se aplica pasilor de actualizare al ponderilor pana cand $\rho_t \leq 4$ se adeverste. Valoarea 4 a fost aleasa empiric. Este de mentionat cazul in care daca $\rho_\infty \leq 4$ si $\beta \leq 0.6$, R-Adam degenereaza in algoritmul de SGD cu Momentum.

2.6.6 Lookahead

Creat relativ recent, Lookahead[27] se diferentiaza de celelalte optimizatoare prezentate. In primul rand, acesta dispune de doua tipuri de ponderi: ponderi incete ϕ si ponderi rapide θ. Avand un punct de pornire ϕ_t memorat, algoritmul efectueaza k pasi cu o variatie a SGD. Acesti pasi se considera ca se aplica pe ponderi rapide. Dupa cei k pasi, functia de cost ajunge intr-un punct θ_k. Avand ϕ_t , se efectueaza in pas in directia θ_k, cu o pondere aleasa arbitrar α. In alte cuvinte, algoritmul „se uita in fata" aplicant SGD, dupa care efectueaza un pas partial in directia finala. Algoritmul este prezentat in 2.4, iar reprezentarea vizuala in figura 2.13.

Algoritm 2.4 Actualizarea ponderilor cu optimizatorul Lookahead

initializeaza parametrii ϕ_0, k si α
functia de cost L, optimizatorul A
pentru pasii $t = 1, 2, \ldots$:
    sincronizeaza parametrii $\theta_{t,0} \leftarrow \phi_{t-1}$
    pentru pasii $i = 1, 2, \ldots$:
        alege lotul d
        actalizeaza ponderile rapide $\theta_{t,i} \leftarrow$
        $\theta_{t,i-1} + A(L, \theta_{t,i-1}, d)$
    sfarsit pentru
    actualizeaza ponderile incete $\phi_t \leftarrow \phi_{t-1} + \alpha(\theta_{t,k} - \phi_{t-1})$
sfarsit pentru

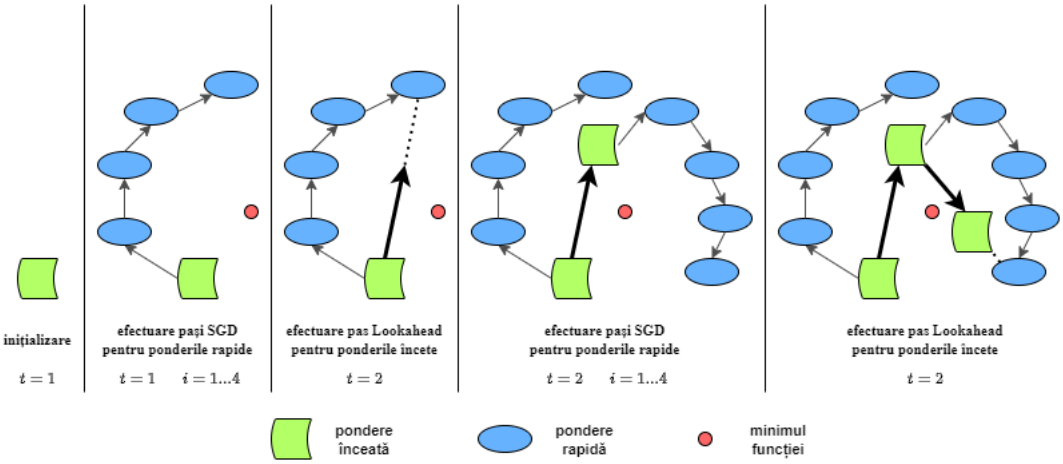


Figure 2.13: Efectuarea a 2 pasi cu optimizatorul Lookahead

2.6.7 Ranger

Ranger[28] este o variatiune a optimizatorului Lookahead ce foloseste Rectified-Adam in loc de SGD. Aceasta combinatie este considerata complementara, intrucat RAdam prezinta imbunatari in starile incipiente ale procesului de antrenare, introducand o „incalzire" a gradientilor, iar Lookahead aduce in acest ansamblu stabilitatea antrenarii in stadiu avansat.

2.7 Problemele Retelelor Neuronale si Solutii

Avand un numar substantial de parametrii si variabile ce trebuiesc ajustate, este aproape imposibil de a prevedea tipurile de probleme ce apar in timpul antrenarii unei retele neuronale.

De la probleme nou descoperite, unde comunitatea academica realizeaza studii sustinute de modele euristice de cercetare, la cele clasic intalnite in experimentele domeniului, unde este urmat un model cantitativ de cercetare, cert este ca inca exista loc pentru imbunatatiri.

2.7.1 Varianță si Bias

In antrenarea unui model, varianta reprezinta cat de mult poate varia o predictie pentru o instanta de intrare nemaivazuta. Daca avem o varianta mare, inseamna ca modelul a invatat prea bine setul de date din timpul antrenarii si nu poate generaliza pentru date noi. Aceasta consecinta se numeste overfitting, si se manifesta printr-o eroare mica in datele de antrenare dar cu o eroare mare in datele de test.

Bias-ul pe de alta parte reprezinta gradul de simplicitate pe care il aplica modelul in predictile sale. In cazul unui bias mare, modelul realizeaza multe presupuneri despre rezultatul dorit, ignorand caracteristicile prezente in setul de date, astfel simplificand prea mult solutia si nereusind sa produca o predictie satisfacatoare. Aceasta consecinta se numeste underfitting, si se manifesta printr-o eroare mare atat pentru datele de antrenare, cat si cele de test.

Se concluzioneaza ca solutia optima prezinta atat o varianta cat si un bias scazut. Deoarece este dificila identificarea cauzei underfitting-ului, solutiile adoptate cauta sa evite complet cazul acesta si propun modele ce sunt predispuse overfitting-ului. Astfel, scopul devine imbunatatirea abilitatii de generalizare a modelului.

Problema de overfitting apare in mod normal tarziu in procesul de antrenare. Pentru a fi detectata si evitata aceasta problema, este o practica buna sa se pastreze un istoric al erorii din timpul antrenarii cat si in timpul testarii. O solutie simpla se numeste early-stopping, ce detecteaza daca modelul incepe sa prezinte overfitting si opreste intreg procesul de antrenare. Deoarece minimele locale existente pot da impresia unui minim global, aceasta solutie nu este optima si trebuie ajustata pentru fiecare problema.

2.7.2 Regularizare

Exista doua solutii pentru combaterea problemei de overfitting: imbogatirea setului de date sau aplicarea metodelor de regularizare. Astfel, regularizarea poate fi definita drept o tehnica de a imbunatati abilitatea unui model de a generaliza.

2.7.2.1 Dropout

Una dintre cele mai folosite metode de regularizare este Dropout[29], reprezentata in figura . Aceasta metoda introduce pentru fiecare neuron din straturile ascunse probabilitatea ca acesta sa fie anulat in timpul unui ciclu de antrenare. La prima vedere, aceast procedeu pare contraintuitiv, intrucat o arhitectura mai complexa creste capacitatea de invatare a retelei. Cu toate acestea, exista cazuri in care la nivelul unui strat unii neuroni domina impactul pe care il au in calcularea predictilor, rezultand ca ceilalti neuroni sa nu poata invata caracteristicile mai subtile. Prin eliminarea neuronilor dominanti in unele cicluri de antrenare se permite dezvoltarea partilor ramase, astfel rezultand intr-un model ce generalizeaza mai bine.

Regularizarea Dropout se foloseste la straturile dense. In cazul imaginilor, informatia este continuta in regiuni, nu in puncte individuale. Aplicarea acestei metode va anula aleator puncte

din imagine, dar nu va elimina caracteristici in totalitate. Batch Normalization prezinta acelasi un efect asemanator de regularizare cu Dropout, fapt pentru care este deseori folosit ca alternativa.

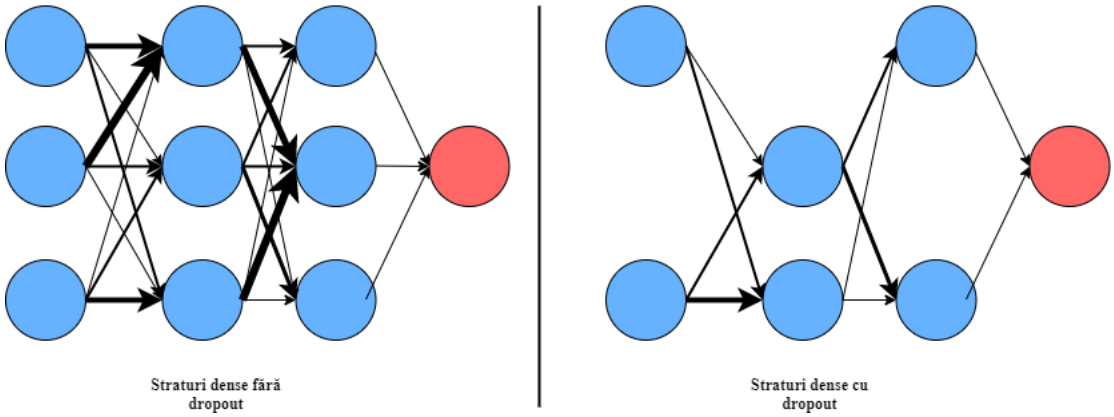


Figure 2.14: Aplicare dropout pe doua straturi dense

2.7.2.2 Regularizare L2

O alta metoda des folosita este regularizarea L2, ce penalizeaza ponderile mari in calculul valorii functiei de cost, adunand suma patratelor ponderilor inmultita cu termenul de regularizare λ. De exemplu, avand ponderile $W = (w_1, w_2, \ldots, w_m)$ si formula de cost Cross-Entriopie Binara 2.3.2, ne rezulta:

$$E = -\frac{1}{n}\sum_{k=1}^{n}((1-y_k)ln(1-\hat{y}_k) + y_k ln(\hat{y}_k)) + \lambda(w_1^2 + w_2^2 + \ldots + w_m^2)$$

2.7.2.3 DropBlock

Creat pentru Retelele Convolutionale, DropBlock[30] anuleaza intregi regiuni dintr-o harta de caracteristici. Aplicarea de Aceasta metoda are doi hiperparametrii:

- $block\_size$ - marimea laturii patratului regiunii anulate
- $drop\_prob$ - probabilitatea ca un punct sa fie ales pentru anulare

Avand o harta de caracteristici, se iau in considerare partile „activate” din aceasta, adica regiunile cu informatie relevanta. Pe punctele continute in aceasta regiune importanta se aplica o distributie Bernoulli cu probabilitatea γ. Deoarece doar punctele cu informatie vor fi alese, se foloseste termenul γ pentru balansa lipsa alegerii punctelor neimportante. Acest termen are formula:

$$\gamma = \frac{drop\_prob}{block\_size^2}\frac{feat\_size^2}{(feat\_size - block\_size + 1)^2} \tag{2.7.1}$$

Tehnica este exememplificata in figura 2.15.

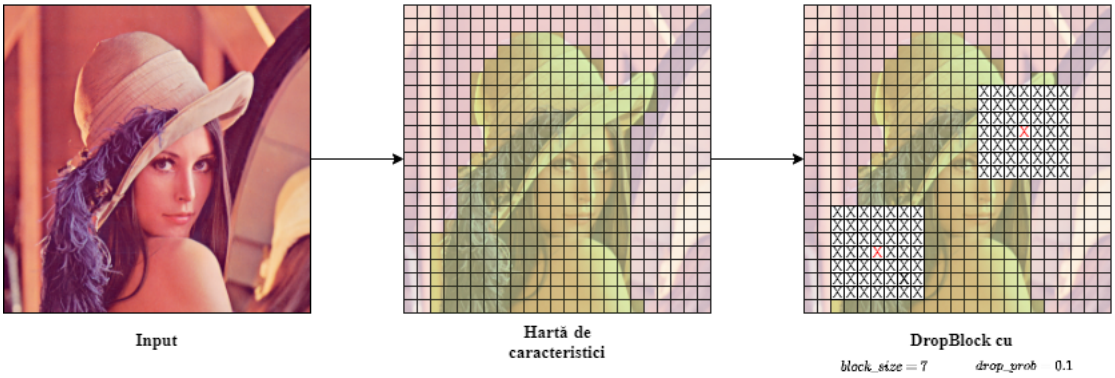


Figure 2.15: Regularizare DropBlock

2.7.3 Problema „dying ReLU”

In domeniul retelelor neuronale, saturatia este un termen ce descrie „tendinta orizontala” a unei functii. Cu cat o functie se aprope de valoarea la care converge, se poate spune ca functia isi pierde din puterea de antrenare pe care o poseda. Din aceste motive, lipsa limitei superioare a unei functii este o calitate dorita, intrucat problema saturatiei este inexistenta. In cazul limitelor inferioare, se considera ca este contraintuitiv sa se aloce resurse pentru valori mari negative, ce sunt irelevante, intrucat reteaua trebuie sa detecteze doar caracteristicile ce determina rezultatul dorit, nu si cele incidental prezente. Existenta unei limite inferioare reduce din problema de overfitting, deci avand un efect regularizator. Este de mentionat faptul ca unele date pot fi procesate ca fiind irelevante devreme in procesul de antrenare, dar sa prezinte relevanta mai tarziu. Acest fapt justifica existenta tratarii valorilor negative in functiile moderne.

Se poate observa de ce functia de activare ReLU este cea mai folosita functie de activare in interiorul retelelor neuronale: aceasta prezinta cel mai rapid timp de executie si nu are limita superioara. Dezavantajul major este faptul ca functia ReLU este complet saturata in domeniul negativ de valori. Acest detaliu duce la problema numita „dying ReLU”. In alte cuvinte, valori ce sunt categorisite negative in stadiile incipiente ale procesului de invatare vor avea valoarea 0. Daca derivata pantei lui ReLU ajunge sa fie 0, la momentul calcularii gradientului cu regula lantului, intreg gradientul se va inmulti cu 0 si va avea valoarea 0, deci nu se va invata absolut nimic. In consecinta, odata ce un neuron cu activare ReLU ajunge la valoarea 0, acesta va ramane asa intotdeauna, intrucat derivata va fi 0 si neuronul nu va putea niciodata sa invete, lucru ce va cauza si limitarea puterii de invatare a neuronilor posteriori. O rata de invatare prea mare poate duce la moartea prematura a neuronilor cu activare ReLU. Empiric, a fost dovedid ca pana la 40% din neuronii cu activare ReLU dintr-o retea pot suferi de aceasta problema, afectand puternic limitele de invatare ale retelei.

O solutie existenta este tratarea valorilor negative, fapt ce face ca derivata sa nu mai fie 0 si in consecinta un neuron cu activare ReLU ce a ajuns la valoarea 0 isi poate reveni in procesul de invatare. Aceasta solutie a fost intai adoptata de functii precum Leaky-ReLU si ELU, si a fost imbunatatia de functii precum Swish si Mish.

Chapter 3

Detalii de Implementare

Problema reconstructiei 3D a obiectelor din poze RGB prezinta aspecte ce nu pot fi rezolvate de metodele clasice existente[31], unde pozitia pixelilor este triangulata. Unul dintre cele mai intalnite aspecte este reconstructia regiunilor ocluzate. Metodele de reconstructie folosesc de 3 tipuri de volume: ansamblu mesh, ansamblu de puncte si ansamblu de voxeli.

In reprezentarea voxelizata exista 3 variatii des folosite:

1. Grila ocupationala binara - un voxel este setat pe valoarea 1 daca face parte dintr-un obiect de interes, altfel este setat pe valoarea 0

2. Grila ocupationala probabilistica - in aceasta reprezentare fiecare voxel are o probabilitate de apartenenta la obiectul de interes

3. Signed Distance Function sau SDF - in fiecare voxel este codata distanta pana la cel mai apropiat punct de pe suprafata. O valoare negativa denota pozitia in interiorul obiectului iar una pozitiva pozitia pe suprafata.

Solutia implementata se numeste YOVO: You Only Voxelize Once 3.1, denumire motivata de faptul ca aceasta trateaza doar reconstructia dintr-o singura imagine a obiectelor, oferind o reconstructie volumetrica voxelizata, atingand rezultate State-of-the-Art pe setul de date Data3D-R2N2. Pentru a trece peste aspectele unde metodele clasice prezinta dificultate, YOVO se foloseste de puterea si eficienta Retelelor Neuronale Convolutionale adanci, reusind sa invete caracteristici ascunse ale obiectului din imaginea de intrare pe baza informatiei existente despre obiecte din aceeasi clasa.

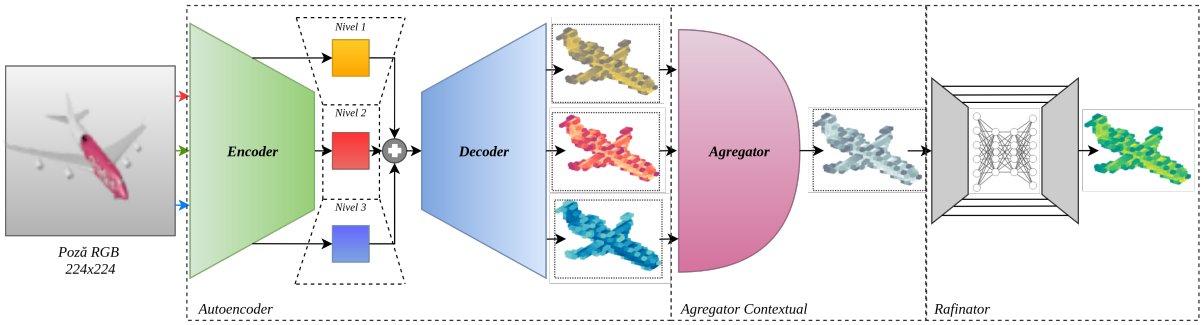


Figure 3.1: Arhitectura YOVO

Domeniul reconstructiei 3D a obiectelor prin invatare profunda este inca in stare incipienta, solutiile existente folosind in mod intensiv resurse in procesul de antrenare. Exista si metode de

reconstructie cu rezolutie mare[32], in care grila de voxeli are dimensiunea 128^3. Aceasta performanta este realizata prin expansiunea retelei, dar prezentand costuri de memorie foarte mari, cresterea acestor costuri fiind cubuice. Din acest motiv, multe dintre solutiile de reconstructie a obiectelor este realizata la o scara mai mica, intr-un spatiu de 32^3. YOVO produce volume in acest spatiu, realizand tranzitia de la o grila ocupationala probabilistica la una binara aplicand un threshold, intreg procesul avand o performanta ce se aproprie de domeniul reconstrutiei in timp real.

Inovatia solutiei prezenta in YOVO este extragerea multi-level eficientizata a caracteristicilor imaginii de intrare, extragand mai exact 3 seturi de caracteristici, fiecare captand aspecte mai mult sau mai putin generale ale obiectului prezent. De asemenea, unul dintre motivele pentru care aceasta solutie atinge noi performante State-of-the-Art este filtrarea informatiei in retea, ansamblul de functii de activare Mish si ELU fiind folosite complementar cu rolul modulelor in care sunt prezente. Aditional, aplicarea metodei de optimizare LookAhead cu Rectified Adam configurata cu hiperparametrii adaptabili, impreuna cu regularizare DropBlock, asigura o oarecare imunitate la overfitting si extinde limitele arhitecturii.

3.1 Arhitectura

Arhitectura retelei YOVO este compusa din 4 componente. Pipeline-ul este urmatorul: se extrag multiple seturi de caracteristici, se reconstruieste fiecare set intr-un volum diferit, aceste volume sunt agregate, volumul rezultat este rafinat de un U-Net tridimensional. Structura detaliata este prezentata in figura 3.2.

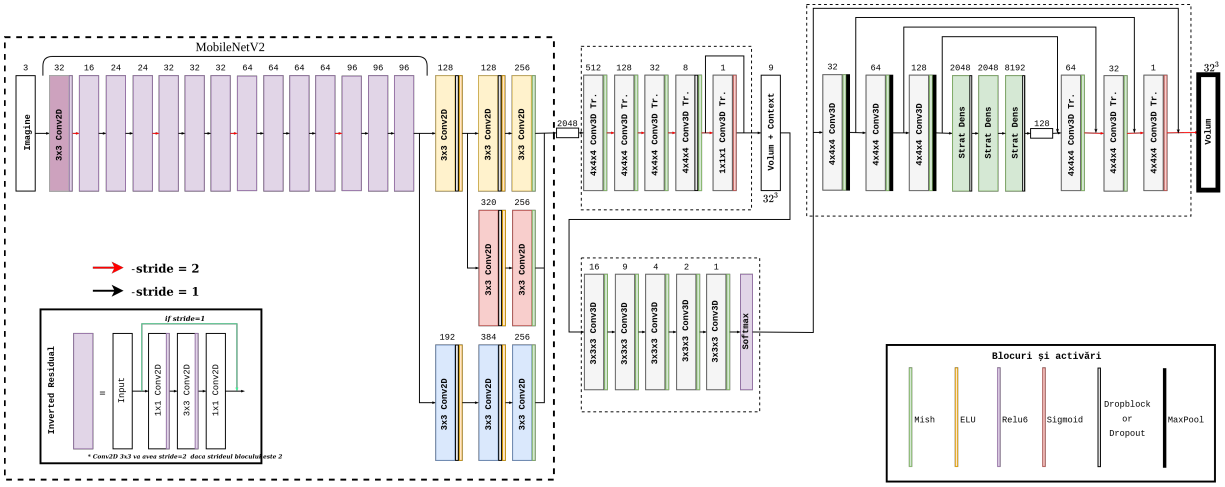


Figure 3.2: Arhitectura detaliata YOVO

3.1.1 Autoencoder

Rolul unui autoencoder este de a transforma un input intr-o reprezentare diferita, un ansamblu 3D. Aceasta structura are doua componente:

1. un Encoder ce interpreteaza datele de intrare si le structureaza multiple seturi de harti de caracteristici

2. un Decoder ce reconstruieste rezultatul dorit pe baza informatiei abstracte prezente in multiplele seturi rezultate din Encoder

3.1.1.1 Encoder

In solutia propusa, caracteristicile sunt intai extrase de un backbone format din primele 14 blocuri 3.3 ale arhitecturii MobileNetV2[11] . Prezenta blocurilor reziduale inversate asigura extragerea de caracteristici complexe intr-un numar mai mic de canale, proces eficientizat de convolutiile separabile. De asemenea, aplicarea straturilor de Batch Normalization au efect regularizator. Setul de caracteristici extras de backbone are dimensiunile $(14, 14, 96)$.

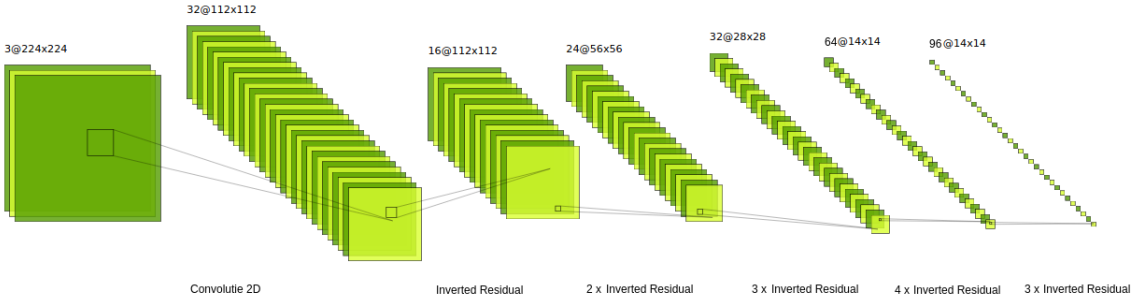


Figure 3.3: Backbone MobileNetV2

In continuare, sunt descrise trei nivele de abstractie la nivelul caracteristicilor cuprinse in spatiul dimensional.

1. Primul nivel cuprinde informatia din spatiul dimensional mare (512 canale) si o contine in 256 canale
2. Al doilea nivel capteaza informatia dintr-un spatiu dimensional mediu (320 de canale) si o codifica in 256 canale
3. Al treilea nivel capteaza caracteristicile generale dintr-un spatiu dimensional mic (128 de canale) si le cuprinde tot in 256 de canale

Toate 3 nivelele produc seturi de caracteristici de dimensiunea $(8, 8, 256)$. Standardul de 256 de canale este ales pentru a se reproduce volume in acelasi grila dimensionala.

Aceasta extragere multi-level este inspirata de versatilitatea arhitecturii FPN, dar adaptata pentru problema reconstruirii volumetrice. Fiecare nivel deriva din primul strat al nivelului precedent, fiind aplicate convolutii ce cresc numarul de canale dar reduc dimensiunile principale (lungime si latime) ale hartilor de caracteristici cu $kernel\_size - 1$, din cauza lipsei aplicarii de padding. Intrucat informatia este extrasa in primele doua convolutii al fiecarui nivel, acestora le sunt aplicate regularizare DropBlock2D si activari ELU. Astfel, metoda DropBlock asigura un nivel de antrenare al caracteristicilor mai putin dominante, iar activarea ELU asigura o filtrare mai buna a caracteristicilor neimportante ale obiectului, acestea rezultand intotdeauna intr-o valoare negativa pentru a evita ambiguitatea creata atunci cand reteaua percepe apartenenta unui obiect la mai multe clase. In urma ultimei convolutii nu este aplicata regularizare DropBlock, dar activarea este setata drept Mish, intrucat in aceasta operatie informatia este codificata in 256 de canale. Arhitectura detaliata a Autencoderului este prezentata in figura 3.4.

3.1.1.2 Decoder

Hartile de caracteristici rezultate din Encoder, ce au structura $(batch\_size, nr\_canale, lungime, latime)$ sunt „aplatizate”, adica reduse la un Tensor 1-Dimensional, apoi sunt restructurate in $(batch\_size, nr\_canale, l$

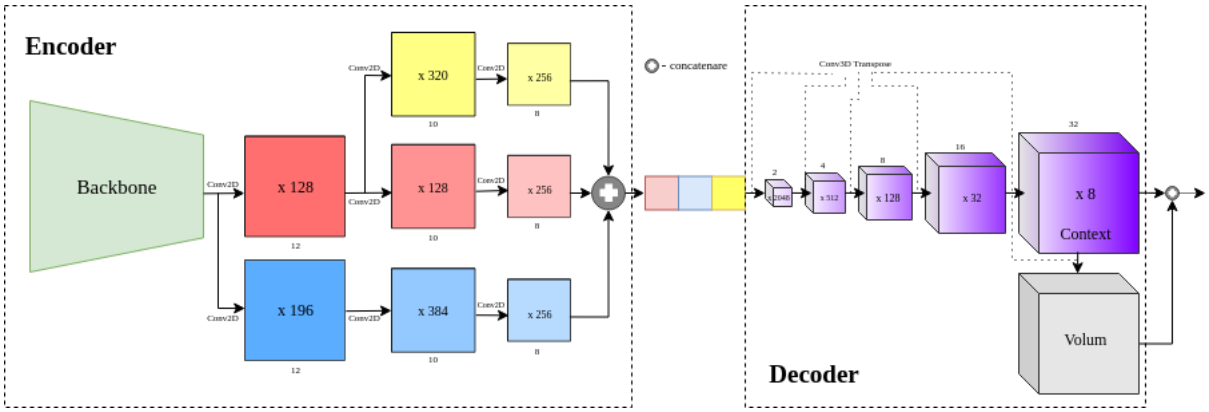


Figure 3.4: Autoencoderul introdus in YOVO

adaugand o noua dimensiune. Astfel, Decoder-ul transforma informatia 2D intr-o interpretare 3D. Pentru a decoda caracteristicile, dimensionalitatea volumetrica $(lungime, latime, inaltime)$ trebuie sa creasca gradual. In consecinta, sunt folosite convolutii 3D transpuse, cu pas de 2. In total sunt folosite 5 straturi convolutionale. Primele 4 convolutii sunt evectuate cu kernel si pas neunitare, aplicand pe acestea Batch Normalization si activari Mish. Ultima convolutie este pointwise si are o activare Sigmoid, ce translateaza intreg ansamblul de caracteristici 3D intr-o grila ocupationala probabilistica. Numerele de canale aplicate pe fiecare convolutie sunt $(512, 128, 32, 8, 1)$. Dupa formarea volumului din ultimul strat, se concateneaza la acesta 8 harti ce caracteristici precedente, ce vor avea rol contextual.

Decoder-ul produce 3 volume voxelizate de 32^3 ale obiectului impreuna cu straturile contextuale ale acestora. Pentru Encoder si Decoder, este folosita functia de cost Cros-Entropie Binara dintre volumul real si cel recreat de intreg Autoencoder-ul.

3.1.2 Agregator contextual

Un context este format dintr-un set de harti de caracteristici extrase din imaginea de intrare, care pe urma a fost interpretat de catre Decoder pana in penultimul strat din acesta. Intrucat extragerea de caracteristici este realizata multi-level de catre Encoder, cele 3 volume rezultate de Decoder sunt reconstruite avand contexte diferite. Fiecare context produce nivele de incredere diferite pentru regiunile volumetrice ale obiectului. De exemplu, un context poate produce un voxeli cu incredere mare pe suprafatele plane, altul poate produce voxeli ce reproduc cu incredere portiunile ascutite marginale.

Agregatorul contextual realizeaza contopirea volumelor in functie de contextul acestora. Pentru a realiza aceasta operatie, se vor crea ponderi pentru fiecare dintre cele 3 volume, mai exact pentru fiecare potential voxel continut in acestea. Aceste ponderi sunt produse de o retea convolutionala 3D, ce interpreteaza fiecare pereche de context cu volum. Ponderile sunt apoi normalizate intre toate 3 contexte cu o functie softmax

‘

$$s_t^{(i,j,k)} = \frac{e^{p_t^{(i,j,k)}}}{e^{p_1^{(i,j,k)}} + e^{p_2^{(i,j,k)}} + e^{p_3^{(i,j,k)}}} \tag{3.1.1}$$

unde s este scorul unui singur voxel (i, j, k) pentru contextul t iar p este ponderea voxelului. Intr-un final, se inmultesc cele 3 volume cu multimea de scoruri aferenta iar volumele rezultate sunt adunate element-wise, rezultand un singur volum intr-o grila ocupationala binara.

Structura acestui modul este prezentata in figura 3.5.

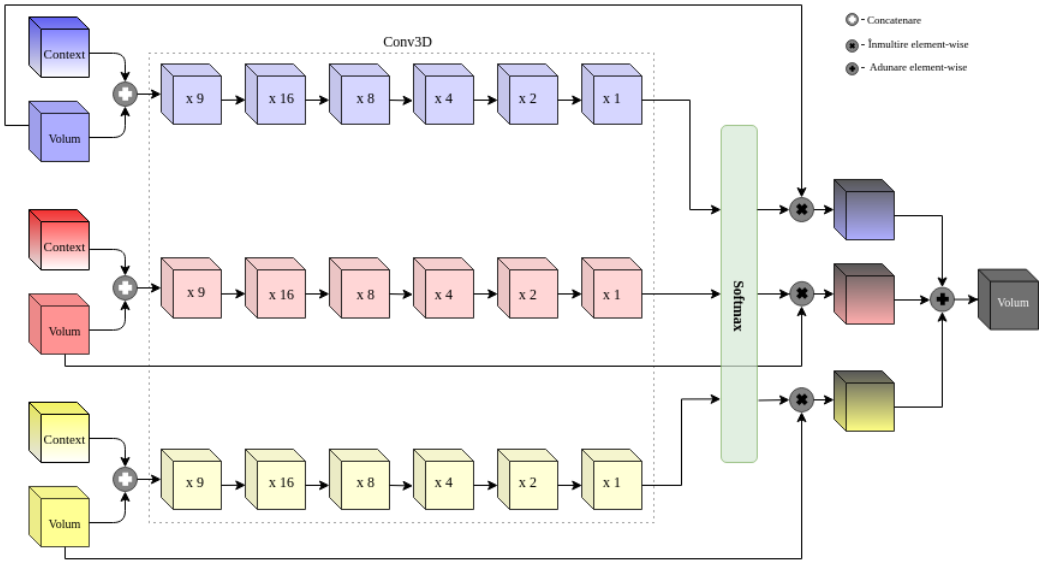


Figure 3.5: Agregatorul prezentat in YOVO

3.1.3 Rafinator

Prezenta sau lipsa unor ansamble mici de voxeli pot fi cauzate din cauza noise-ului . In cazul grilei dimensionale de 32^3, asemenea ansamble pot avea un efect semniticativ in scaderea acuratetei modelului, aspect ce are mai putin impact in grile dimensionale mari. Din aceste motive, scopul acestui modul este de a imbunatati si retusa volumele produse de modulele anterioare. O analogie facuta de [2] face referinta la scopul blocurilor reziduale, in care captarea informatiei reziduale produce rezultate imbnatatite.

Acest modul consista intr-un 3D U-Net - un Autoencoder cu skip connections intre Encoder si Decoder. Encoderul este compus din 3 convolutii 3D cu efect de downsapling, rezultat din kernelul si pasul non-unitar. Trei straturi dense cu dropout prelucreaza informatia, dupa care aceasta trece prin alte 3 straturi de convolutii 3D transpuse. Intre straturile convolutionale ale Encoderului si Decoderului sunt efectuate skip-connections, pentru pastrarea informatiei. Pentru fiecare strat convolutional este aplicat Batch Normalization si activare mish, exceptie facand ultimul strat ce are activare Sigmoid.

Volumul rezultat este compus dintr-un ansamblu probabilistic de voxeli, pe care se aplica in paralel limite de 0.2, 0.3, 0.4 si 0.5, rezultand intr-un ansamblu binar. Pentru a se decide limita optima, se calculeaza acuratetea pentru fiecare in parte si se alege varianta cu cele mai bune rezultate.

3.2 Set de date

Datasetul ShapeNet contine o multitudine de Proiectari 3D asistate de calculator (CAD). Este folosit un subset al acestui dataset, ce contine 43,783 de modele structurate in 13 clase ordonate alfabetic. Clasele sunt: aeroplane(avion), bench(banca), cabinet(dulap), car(masina), chair(scaun), display(monitor), lamp(lampa), speaker(boxa), rifle(arma), sofa(canapea), table(masa), telephone(telefon), watercraft(barca). Denumirile acestora sunt codificate in conformitate cu WordNet.

3.3 Procesul de antrenare

Pentru a incepe procesul de antrenare, o configuratie trebuie stabilita. Retelei YOVO ii se pot seta anumiti parametrii direct din linia de comanda (e.g. denumirea experimentului, numarul de epoci, tipul rularii: testare/antrenare etc.), dar majoritatea dintre acestia sunt configurati intr-un fisier separat. Pentru simplicitatea accesarii valorilor din fisierul de configuratie, este folosita biblioteca easydict. De asemenea, YOVO salveaza progresul modelului la fiecare 10 epoci sau atunci cand performanta pe setul de validare atinge noi valori maxime.

3.3.1 Preprocesarea datelor

Imaginile din datasetul ShapeNet au in medie dimensiunea de 137x137. Acestea sunt cropuite la marimea 128x128 apoi redimensionate la 224x224. Pentru procesul de antrenare sunt aplicate metode de augmentare precum: flip aleator, culoare de fundal, permutare canale RGB, Normalizare, introducere de Noise. In schimb, pentru procesul de testare, doar normalizarea si schimbarea culorii de fundal sunt aplicate.

3.3.2 Hiperparametrizare

Hiperparametrii existenti in YOVO fac referire atat la procesul de antrenare sau optimizare, cat si la celelalte metode folosite.

Sistemul este antrenat pentru 250 de epoci, cu marimea lotului $batch\_size = 32$.

In algoritmul de optimizare Ranger, hiperparametrii au fost adaptati problemei actuale.

- RAdam: $\beta = 0.9$, $\gamma = 0.999$, $\epsilon = 10^{-8}$
- Lookahead: $\alpha = 0.5$, $k = 6$

Pentru Encoder, Decoder si Rafinator avem rata de invatare $lr = 10^{-3}$ iar pentru Agregator avem $lr = 10^{-4}$. De asemenea, dupa 150 de epoci ratele de invatare se inmultesc cu un factor de 0.5.

Tehnicile de regularizare se aplica doar in timpul antrenarii. Pentru DropBlock avem $drop\_prob = 0.05$ si $block\_size = [1, 2]$, iar rata de dropout aplicata straturilor dense ale Rafinatorului este 0.1.

3.3.3 Functia de cost si Metrica

Pentru a calcula eroarea in timpul antrenarii este folosita Cross-Entropia Binara Voxel Unitara 2.5.8, iar metrica folosita pentru calculul acuratetei modelului este 3DIoU2.5.9.

3.3.4 Validare si Testare

O practica des intalnita este impartirea setului de date atat pentru antrenare si testare, cat si pentru validare. Datele de validare sunt folosite pentru a evalua sistemul in timpul antrenarii, in scopul ajustarii hiperparametrilor. Datele de test au rolul de a evalua modelul final, pentru formarea unei analize comparative cu alte modele.

Din setul de date ShapeNet, 30640 de imagini sunt folosite pentru antrenare, 4371 pentru validare si 8762 pentru testare.

3.4 Mediul de antrenare

3.4.1 Resurse Hardware

Antrenarea si rularea retelelor neuronale sunt procese ce necesita o cantitate mare de resurse atat computationale cat si de stocare. Reproducerea volumelor obiectelor din imagini 2D este o solutie cu o arhitectura complexa, ce prezinta un numar mare de ponderi antrenabile.

Din punct de vedere hardware, marimea spatiului de stocare, procesorul, placa video si memoria RAM sunt cele mai importante componente. Pentru problemele de Invatare Profunda nu este important daca setul de date este stocat pe un HDD sau pe un SSD. Procesorul este folosit in principal pentru preprocesarea datelor, numarul de nuclee si abilitatea de hyperthreading fiind esentiale in paralelizarea si accelerarea acestui proces. Marimea memoriei RAM va denota cat de multe date se vor putea aloca intr-o iteratie. Deoarece accesarea memoriei disk este seminificativ de inceata, se incearca incarcarea a cat mai multor date in memoria RAM, pentru a scadea numarul de accesari in memoria disk. Datele sunt transmise din memoria RAM in memoria placii video. Pentru a evita cazuri de limitare, este de preferat ca memoria RAM sa fie mare. Majoritatea operatilor realizate intr-o retea neuronala sunt inmultirile matriciale, operatie pentru care placile video sunt mai eficiente. Experimentele academice sunt dominate de placile NVIDIA, ce folosesc arhitectura software CUDA drept baza la antrenarea retelelor neuronale.

YOVO a fost antrenat timp de 60 de ore pe un sistem compus din: procesor Intel Core i7-6800K 3.40GHz, placa video NVIDIA RTX 2080Ti si memorie RAM 24Gb DDR4. Referitor spatiului de stocare, este nevoie de 6.3Gb pentru setul de date, 1.8Gb pentru modelul antrenat YOVO si 45Gb pentru salvarea modelelor intermediare in timpul antrenarii. In total, se recomanda cel putin 60Gb spatiu de stocare.

3.4.2 Resurse Software

YOVO a fost scris in Python3, un limbaj de programare interpretat, de nivel inalt, cu accent pe simplicitate si intelegere usoara a codului. Versiunea softwareului de interactiune cu placa video NVIDIA este CUDA 10.2. Frameworkul open-source folosit pentru manevrarea softwareului CUDA si pentru codarea modulelor specifice retelelor neuronale este PyTorch 1.5. Acesta aduce simplicitate procesul de experimentareh si prezinta atat o interfata in python cat si una in C++. Pentru a oferi versatilitate in experimentarea diferitelor solutii din invatarea profunda, este creat un mediu de antrenare virtual cu ajutorul softwareului Anaconda, in care se instaleaza urmatoarele biblioteci:

1. argsparse - parsarea parametrilor din linia de comanda
2. easydict - accesarea membrilor unui dictionar ca si atribute
3. matplotlib - salvarea si vizualizarea imaginilor si a videoclipurilor de exibitie
4. numpy - folosit preponderent pentru inmultirea matricilor multidimensionale mari
5. opencv-python - modificarea si procesarea de imagini
6. pprint - afisarea ordonata a configuratiei
7. scipy - citirea si manevrarea fisierelor tipice reprezentarii 3D

8. echoAI - contine implementarea functiei de activare Mish
9. pyglet - pachet necesar bibliotecii kaolin
10. kaolin - biblioteca pentru pentru reprezentare interactiva a obiectelor 3D
11. dropblock - implementarea regularizarii dropblock 2D si 3D
12. tensorboardX - salvarea si vizualizarea statisticilor procesului de antrenare

3.4.3 Vizualizare

YOVO poate fi configurat sa afiseze volumele reconstruite si cele reale, sa le salveze intr-un format specific aplicatiilor grafice 3D, sa le prezinte in 2 tipuri de medii interactive sau sa creeze un videoclip de exibitie in 360 de grade pentru acestea. O optiune pentru vizualizare interactiva este prin folosirea bibliotecii kaolin, ce prezinta volumele simplistic dar le poate converti si intr-un ansamblu mesh. Alternativa este vizualizarea interactiva folosind matplotlib, in care se poate observa grafic increderea pentru fiecare voxel din volumul reconstruit.

Chapter 4

Experimente si Rezultate

nr parametrii

specs

Accuracy measures the percentage correctness of the prediction i.e. correct−classestotal−classes while

Loss actually tracks the inverse-confidence (for want of a better word) of the prediction. A high Loss score indicates that, even when the model is making good predictions, it is less sure of the predictions it is making...and vice-versa.

So...

High Validation Accuracy + High Loss Score vs High Training Accuracy + Low Loss Score suggest that the model may be over-fitting on the training data.

TANH

Bibliography

[1] Christopher B. Choy and Danfei Xu and JunYoung Gwak and Kevin Chen and Silvio Savarese (2016). 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object ReconstructionCoRR, abs/1604.00449.

[2] Haozhe Xie and Hongxun Yao and Xiaoshuai Sun and Shangchen Zhou and Shengping Zhang and Xiaojun Tong (2019). Pix2Vox: Context-aware 3D Reconstruction from Single and Multi-view ImagesCoRR, abs/1901.11153.

[3] Cunningham, Padraig & Cord, Matthieu & Delany, Sarah. (2008). Supervised Learning. 10.1007/978-3-540-75171-7_2.

[4] Goodfellow, Bengio & Courville (2016, 6.5 Back-Propagation and Other Differentiation Algorithms, pp. 200–220)

[5] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/

[6] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NeurIPS Autodiff Workshop (2017), https://pytorch.org/

[7] Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629 (2018)

[8] Chollet, Francois. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. 1800-1807. 10.1109/CVPR.2017.195.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, andJian Sun. Deep residual learning for image recog-nition.CoRR, abs/1512.03385, 2015.

[10] Saining Xie, Ross B. Girshick, Piotr Doll´ar,Zhuowen Tu, and Kaiming He.Aggregatedresidual transformations for deep neural networks.CoRR, abs/1611.05431, 2016.

[11] Mark Sandler and Andrew G. Howard and Menglong Zhu and Andrey Zhmoginov and Liang-Chieh Chen (2018). Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and SegmentationCoRR, abs/1801.04381.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations. MIT Press, Cambridge, MA, USA, 318–362.

[13] Ronneberger, Olaf & Fischer, Philipp & Brox, Thomas. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv:1505.04597v1

[14] Lin, Tsung-Yi et al. "Feature Pyramid Networks for Object Detection." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017): 936-944.

[15] Marcel Wever, F.M., Hüllermeier, E.: ML-Plan for unlimited-length machine learning pipelines. In: Garnett, R., Vanschoren, F.H.J., Brazdil, P., Caruana, R., Giraud-Carrier, C., Guyon, I., Kégl, B. (eds.) ICML workshop on Automated Machine Learning (AutoML workshop 2018) (2018)

[16] Glorot, X., Bordes, A. & Bengio, Y.. (2011). Deep Sparse Rectifier Neural Networks. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, in PMLR 15:315-323

[17] Maas, Andrew L, Hannun, Awni Y, and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In Proc. ICML, volume 30, 2013.

[18] Clevert, Djork-Arné & Unterthiner, Thomas & Hochreiter, Sepp. (2016). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).

[19] Prajit Ramachandran, Barret Zoph, & Quoc V. Le. (2017). Searching for Activation Functions.

[20] Diganta Misra. Mish: A self regularized nonmonotonic neural activation function. arXiv preprint arXiv:1908.08681, 2019.

[21] Herbert E. Robbins (2007). A Stochastic Approximation MethodAnnals of Mathematical Statistics, 22, 400-407.

[22] David E. Rumelhart, Geoffrey E. Hinton, & Ronald J. Williams (1986). Learning representations by back-propagating errorsNature, 323, 533-536.

[23] Andrew Yan-Tak Ng. „Deep Learning Specialization". https://www.deeplearning.ai/ (2017)

[24] G. E. Hinton. "Lecture 6e RMSProp: Divide the gradient by a running average of its recent magnitude". https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[25] Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

[26] Liu, Liyuan & Jiang, Haoming & He, Pengcheng & Chen, Weizhu & Liu, Xiaodong & Gao, Jianfeng & Han, Jiawei. (2019). On the Variance of the Adaptive Learning Rate and Beyond.

[27] Michael R. Zhang and James Lucas and Geoffrey E. Hinton and Jimmy Ba (2019). Lookahead Optimizer: k steps forward, 1 step backCoRR, abs/1907.08610.

[28] Less Wright (2019). New Deep Learning Optimizer, Ranger: Synergistic combination of RAdam + LookAhead for the best of both. https://medium.com/@lessw/new-deep-learning-optimizer-ranger-synergistic-combination-of-radam-lookahead-for-the-best-of-2dc83f79a48d

[29] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," arXiv preprint arXiv:1207.0580, 2012.

[30] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. DropBlock: A regularization method for convolutional networks. In Advances in Neural Information Processing Systems (NIPS), pages 10727–10737, 2018.

[31] R. Hartley and A. Zisserman,Multiple view geometry in computervision. Cambridge university press, 2003.

[32] J. Wu, Y. Wang, T. Xue, X. Sun, B. Freeman, and J. Tenenbaum,"MarrNet: 3D shape reconstruction via 2.5D sketches," inNIPS,2017, pp. 540–550.

[33] X. Han, H. Laga and M. Bennamoun, "Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era," in IEEE Transactions on Pattern Analysis and Machine Intelligence, doi: 10.1109/TPAMI.2019.2954885.