INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Informatika Ingeniaritzako Gradua

Konputagailuen Ingeniaritza

Gradu Amaierako Lana

Trenetako ikusmen artifizialeko sistemen hardware bidezko azelerazioa

Egilea

Jokin Irastorza Zabalegi

2021

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Informatika Ingeniaritzako Gradua

Konputagailuen Ingeniaritza

Gradu Amaierako Lana

Trenetako ikusmen artifizialeko sistemen hardware bidezko azelerazioa

---

Egilea

Jokin Irastorza Zabalegi

Zuzendariak

Jose A. Pascual Saiz (EHU/UPV)

Mikel Labayen Esnaola (CAF Signalling)

Laburpena

Trenen automatizazioan beharrezkoa da ingurunearen informazioa hautemateko gai izango den sistema bat edukitzea: PER edo *PERception* modulua. Sentsoreetako datuak irakurri, prozesatu eta erabakiak hartzen dituen sistemari igortzeaz arduratzen den modulu horrek Polaris izena hartzen du CAF Signalling enpresan, eta ikusmen artifizialean oinarritzen du bere jarduna. Informazio-fluxu hori denbora errealeko aplikazioa da, gertaera eman denetik sistemak horren berri izan arte igarotako denbora kritikoa baita. Ziklo hori murrizteko, edo ziklo berean kalkulu gehiago egiteko (sentsore gehiagoren datuak prozesatzeko, adibidez), prozesuaren atalak optimizatu behar dira.

Proiektu honen helburua Polaris-eko objektuen detekzioko zatia optimizatzea da. Horretarako, erabiltzen diren *hardware* azeleragailuei eta une honetan PER sistemak darabilen Nvidia Jetson Xavier AGX gailuaren ezaugarriei buruzko ikerketa lan bat egin da, hasteko. Ateratako ondorioetan ikusi da gailu honetan egokiena Nvidiaren sare neuronalak optimizatzeko TensorRT liburutegia erabiltzea dela. Beraz, ondoren, CNN sareak optimizatu eta inferentzia gauzatzeko prozesua lan honetarako bereziki sortutako Casio azpimoduluan bildu da, Polaris-en integratu ahal izateko. Etorkizunean bestelako modeloetara hedatzea aurreikusi arren, momentuz YOLOv4 sarea optimizatu da, TensorRT-k eskaintzen duena beharrezko metodo eta CUDA *kernel*ekin osatuz. Prozesu horri esker, detekzioetan antzeko kalitatea mantenduz errendimendua nabarmen hobetu da: inferentzia-denbora edo latentzia erdira murrizu da; beste hitzetan, emaria edo *throughput*a bikoiztea lortu da.

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

1. KAPITULUA

Sarrera

Hasierako eskuzko sistema primitiboak atzean utziz, trenetako seinalizazio-sistemak automatizazio prozesu etengabea bizi izan du azken hamarkadetan. Trenbideko nahiz treneko azpiegitura elektroniko konplexuei esker segurtasuneko funtzionalitateak automatizatu ziren lehenik: *Automatic Train Protection* (ATP) sistemek hartu zuten segurtasuna bermatzearen ardura. ATP horiek ezarritako mugen barruan, oraindik ere gidariaren esku zegoen trenen kontrol operazionala, *Automatic Train Operations* (ATO) sistemen garaia iritsi zen arte. ATO-ek trenetik zuzenean edo/eta ATP-tik jasotako informazioarekin gidatzen dute trena energia-kontsumoa, erosotasuna etab. zainduz, ohiko egoeretan. Hala ere, oraindik gidariaren esku dago akatsak zuzentzeko kontrola hartzea, trena gelditu/abiatzea, ateak ireki/ixtea edo ezusteko egoerei erantzutea.

Trenen industrian GoA4 moduan ezagutzen den erabateko automatizaziora beharrezkoa da ingurunearen nolabaiteko PERzeptzioa izatea sistemak, trena gidatzea ekintza dinamikoa baita. Paper hori betetzen du PER edo hautemate-moduluak: "kanpoko" sentsoreen bidez (kamerak, LiDAR-a...) inguruaren informazioa jaso, prozesatu eta dagokion sistemari bidaltzen dio. PER sistemaren funtzionalitateen artean seinaleak detektatzea, trenbideko oztopoak antzematea eta objektuetara dauden distantziekin kokapena fintzea daude. Kontuan izan behar da kritikoa dela gertaera bat jasotzen denetik ATO edo ATP sistemak bere informazioa jaso arteko latentzia, informazioa eskuratzea denbora errealeko aplikazio bihurtuz. Lan horiek betetzeko erabilitako tekniken artean ikasketa sakonekoak daude, azken aldian trenen industriara ere iritsi baita adimen artifizialaren iraultza.

Seinalizazioaz arduratzen den CAF taldeko CAF Signalling enpresan, PER sistemak Po-

laris izeneko moduluan biltzen ditu ikusmen artifizialarekin lotutako atazak: objektuen detekzioa, distantzien kalkulua, angeluen estimazioa...

Testuinguru horretan, proiektua garatzerakoan, ikerkuntza eta inplementazioa uztartu dira, enpresak ezarritako erronkari aurre egiteko. Alde batetik, sare neuronalen azeleragailuak eta azpiegitura moduan erabili den plaka ikertu dira, PER taldearen etorkizunerako asmoetarako lagungarri izan daitezen. Bestetik, Polaris-en objektuen detekzioko zatia azpimodulu propio batean kapsulatu da, eta baita egun darabilten modeloa optimizatu ere, Casio azpimodulua sortuz.

Lehenengo, testuingura hobeto ulertzeko, trenbideetako seinalizazioari eta bere automatizazioari buruzko informazioa kontsultatu da. Era berean, lana aurrera eraman ahal izateko, sare neuronalei eta ikasketa sakonari (*Deep Learning*) buruz ikasi behar izan da. Formakuntza prozesu horien lekukotza izan asmo dute 4. eta 5. atalek, hurrenez hurren (A eta B eranskinekin batera).

Taldeak une honetan darabilen Nvidia Jetson Xavier AGX plaka aztertzea ere proiektuaren parte da, analisi hori etorkizunean enpresari lagungarri izan dakiokeelakoan, eskaintzen dituen aukera guztiak ustiatzen saiatzeko eta beste plataformekin konparatu ahal izateko. Bere arkitektura orokorraren eta osagai nagusien laburpena egin da 6. atalean. Ikerketako eskakizunak betetzeko, inferentziaren azeleraziorako erabiltzen diren *hardware* azpiegituren laburpen orokorra, berriz, 4. kapituluaren amaieran txertatu da.

7. kapituluan inplementazioaren nondik norakoak azaldu dira: Nvidiak sareak optimizatzeko eskaintzen duen TensorRT liburutegiaren erabilera eta sareen bihurketak; YOLOv4 sarearentzat inplementatu diren CUDA *kernel* espezifikoak; Casio moduluaren diseinua; eta lehen prototipotik azken proposamenera bitarte jarraitu diren pausuak. Jarraitutako bidearekin, YOLOV4 sarearentzat lortutakoa gainerako sare neuronal konboluzionaletara orokortzea erraztea bilatu da soluzioa diseinatzerakoan.

Proposatutako bertsioarekin lortutako emaitzak jatorrizko sarea OpenCV tresnarekin exekutatuz (egungo funtzionamendua) lortutakoekin konparatu dira 8. atalean. Besteak beste, bertsio optimizatuaren emaitzak behar bezain onak direla ziurtatu da, eta lortutako errendimendua konparatu: zikloaren iraupena edo latentzia, eta emaria (frame segunduko edo FPS).

Bukatzeko, proiektuaren irismenetik kanpo gelditu diren etorkizunerako lan-ildo berriak proposatu dira, proiektuaren ondorio orokorren aurretik.

2. KAPITULUA

Proiektuaren helburuak

Proiektu honen helburu nagusia irudietan **denbora errealeko objektuen detekzioa** egin ahal izateko erabiltzen diren sare neuronal konboluzionalen exekuzioa eskuragarri dagoen hardwareak eskaintzen dituen aukeratara egokitzea eta **optimizatzea** da; hots, YOLOv4 sareari Nvidia etxearen Jetson Xavier AGX plakarako hobekuntza espezifikoak egitea. Zehazki, inferentzia-prozesuaren emaria (**FPS**) **handitzea** edo latentzia (**ms**) **txikitzea** bilatzen da. Aldi berean, GPUaren erabilera (%) murriztu nahi da, bestelako atazentzat libre uzteko.

Ezertan hasi aurretik, lehen pausua irudien detekziorako sare neuronaletan erabili ohi diren *software*- eta *hardware*-azeleraziorako teknika eta liburutegiak **ikertzea** eta dokumentatzea izango da. Horrez gain, azpimarra berezia jarriko da Xavier AGX plataformaren ***hardware*a aztertzea**n. Sistemaren osagaien eta euren arteko komunikazioen behemailako analisia egingo da, eskaintzen dituen baliabideak modu sakonean ezagutzeko. Epe laburrean egingo diren optimizazioak analisi hau baino goi-mailakoagoak izatea aurreikusi arren, enpresaren aldetik premiazko eskakizuna izan da, epe ertainean plataformarekin lanean jarraitzeko.

Lehen bi lanketa horien ondorioetatik abiatuta, YOLOv4 **sarea optimizatu**ko da plakarako, errendimenduarekin lotutako helburuak betetzen saiatzeko. Aplikazioaren denbora errealeko izaera kontuan hartuz, lortutako hobekuntza garrantzitsua izango da, zeharkako bi helbururi ere erantzungo baitie: Polaris sistemaren zikloa laburtzeari edo/eta ziklo berean kalkulu gehiago egin ahal izateari. Puntu honetara iritsitakoan, ezagutza- eta denbora-mugak kontuan hartuz, enpresarekin eta unibertsitateko tutorearekin adostuko

dira proiektuaren baitan inplementatu beharreko hobekuntzak.

Egindako optimizazio zehatzaz gain, sarearen bertsio ezberdinen erabilera eta etorkizuneko optimizazioak erraztuko dituen **aplikazioa diseinatu** eta garatu beharko da, azpimodulu independente bat **inplementatu**z.

Garapenarekin amaitutakoan, egindako aldaketek **errendimendua**n duten eragina **ebaluatu** eta konparatu beharko da: hasierako bertsioaren eta bertsio berriaren/berrien artean, konfigurazio-aukeren artean...

Inplementatu gabe gelditu diren (edo etorkizunean gehiago landu daitezkeen) hobekuntzen proposamena ere idatzi beharko da, enpresak aurrera begira bide horien berri izan dezan.

Azkenik, egindakoa enpresako hautemate-sisteman **integratu** beharko da, bertsio hobetua inferentzia-modulu berria izan dadin. Egindako probetan integrazioa egokia bada, eta aukerarik egon bada, garatutako programa trenean probatzea da asmoa.

Laburbilduz, hauek dira proiektuaren helburu teknikorik esanguratsuenak:

1. CNN sareen *hardware* bidezko azelerazioa ikertzea.
2. Xavier AGX plakaren arkitekturaren eta bere osagaien ezaugarriak ikertzea.
3. YOLOv4 sarea Xavier AGX plakarako optimizatzea.
4. Aurreko puntuko prozesua bestelakosareen optimizaziorako orokortzea.
5. Polaris-eko objektuen detekzioa eta beharrezko optimizazioak elkartuko dituen azpimodulua inplementatzea.
6. Inferentziaren errendimendua neurtzea.
7. Sarearen bertsio optimizatua PER sistemaren baitan integratzea eta probatzea.
8. Exekuzioaren optimizazioaren gaian etorkizunean jorratzeko bideak aztertu eta zerrendatzea.

Aipatutako helburu teknikoez gain, proiektuak bere baitan biltzen ditu garapen pertsonal eta profesionalarekin, proiektuen kudeaketarekin edo taldean lan egitearekin lotutako zeharkako helburuak ere.

3. KAPITULUA

Plangintza

Kapitulu honetan proiektuaren gainerako faseekin hasi aurretik egindako planifikaziolana (atazen deskonposizioa eta deskribapena, denboren aurreikuspena, arrisku-plana eta lanerako metodologia) laburtu, eta errealitatean gertatu denarekin konparatuko da (errealitatean behar izan den denbora eta izandako desbiderapenen analisia).

3.1 EDT diagrama

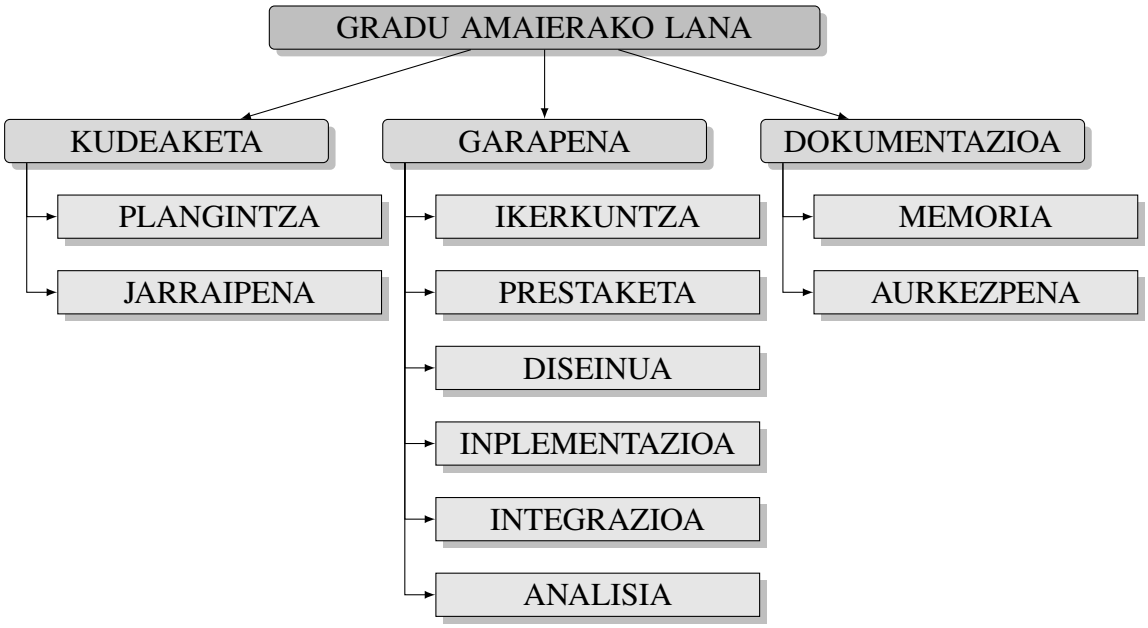
Proiektuan zehar egin beharreko atazak LDE ("Lanen Deskonposaketa Egitura") diagrama batean bildu dira, lan-paketeka sailkatuta (3.1. irudia).

3.2 Atazen deskribapena

3.2.1 Kudeaketa

- **Plangintza:**

  Lehen egunetan proiektuaren oinarrizko plangintza osatuko da; besteak beste, helburu orokorrak identifikatzeko, egin beharreko lanak zerrendatzeko, GrAL-ari dagozkion orduak lan horien artean nola banatuko diren erabakitzeko, arriskuen eragina minimizatzeko estrategiak garatzeko...



3.1 Irudia: Proiektuaren LDE diagrama

Plan horren gainean, lan-metodologia eta azpi-atazen banaketa zehatzagoa enpresarekin adostuko da. Proiektuak iraun bitartean garrantzitsua izango da taldeko arduradunarekin adostutako epe laburreko plangintzak lantzen joatea, eta horretarako, bi astean behin PER talde-mailan egiten den *Sprint Planning* bilera baliatuko da.

- **Jarraipena:**

  Enpresako dinamikaren baitan, goizeroko bilera laburrak (*Daily Meetings*) eta bi astean behingo *Sprint Review* bilerak baliatuko dira nagusiki egindakoaren jarraipen zehatza egiteko. Lehenengoak baliagarriak dira eguneroko *update* laburren bidez elkarren berri izateko, eta beharrezkoa denean elkarri arazo puntualak konpontzen laguntzen saiatzeko; bigarren motako bileretan, berriz, azpi-ataza bakoitzak azken bi asteetan izandako bilakaera aztertu ohi da, aurrez aipatutako *Sprint Planning*ean hurrengo periodoa planifikatu aurretik. Epe finkorik gabe, taldeko arduradunarekin buruz buruko bilerak ere aurreikusten dira.

  Unibertsitateko zuzendariari ere maiz helaraziko zaio proiektuaren nondik norakoen berri. Hartu-eman hauetan ziurtatu beharko da lanak Fakultateak ezarritako eskakizunak betetzen dituela, hartutako bidea hasierako helburuekin bat datorrela eta abar.

3.2.2 Garapena

- **Ikerkuntza:**

  Garapeneko lehen fasea helburuekin lotura zuzena duten hainbat gairen inguruko informazioa eskuratu eta ikertzea izango da: plakaren ezaugarri zehatzak, Nvidiaren *software*-a, optimizazio-teknikak... Beharrezko formakuntza ere ataza honetan sartuko da.

- **Diseinua:**

  Enpresarentzat garrantzi berezia du programaren *software* mailako diseinuak, eta bereziki zaindu beharko da han markatutako gidalerroak jarraitzea.

  Diseinuko atalean ikerkuntza-fasean aurkitutako eta inplementatzea erabakitako soluzioak nola programatuko diren erabaki beharko da, kontuan hartuz elkarren artean konparatzea eta etorkizuneko aldaketan sartzea ahal beste erraztuko duen marko bat eraiki nahi dela.

  Gainera, eboluzio osoaren osteko azken soluzioa, bereziki, orain arteko kodearekin era koherentean, eta gainerako moduluek eskatzen duten interfazea errespetatuz diseinatu beharko da.

  Diseinua helburuanitza izango da, beraz: alde batetik, programak soluzioak elkarren artean konparatu, konputazio-denborak kalkulatu, eta bestelako analisiak erraztu beharko ditu inplementazioak; bestetik, *Polaris* aplikazioaren inferentziaren zatiaz arduratu beharko du.

- **Prestaketa:**

  Inplementazioarekin hasi aurretik, beharrezkoak edota lagungarriak diren tresnak instalatu eta lanerako ingurunea egokitu beharko da.

  Atal honen konplexutasuna garaia iristean baloratu beharko da, plakaren egoeraren eta lehen atalean ikusitako beharren arabera. Gainerako taldekideek ere plaka erabiliko dutenez, unea iristean gerta daiteke *software* dependentzia batzuk ase beharra, bertsio-aldaketak egin beharra... suertatzea.

- **Inplementazioa:**

  Diseinatutako soluzioa inplementatu egin beharko da: sarearen optimizazio zehatzak, CNN-en optimizaziorako osagai orokorrak, guztiak bilduko dituen moduluko klase-hierarkia...

Inplementazio guztia `C++` lengoaia erabiliz egingo da, esku arteko tresnek horretarako aukera eskaintzen dutenean, behitzat. Ondorioz, beharrezko konpilazioko fitxategiak ere sortu beharko dira.

- **Analisia:**

  Lortutako emaitzak aztertu eta konparatu beharko dira, optimizazioaren bidez lortutako balizko errendimenduaren hobekuntza neurtzeko.

  Lehenik sare berrien emaitzak jatorrizkoarekin alderatuz koherenteak direla ziurtatu beharko da - detekzioak zehatzak direla, alegia.

  Helburu nagusia emaria (*throughput*, FPStan) handitzea eta latentzia (ms-tan) murriztea da, ahal bada, aldi berean GPUaren erabilera (%-tan) murriztuz. Ondorioz, balio horiek aztertu beharko dira bereziki.

  Inferentziaren exekuzio-denbora osoa zatikatzea aurreikusten da, *profiling* teknikak erabiliz. Horrela, hobetzeko tarterik handiena eskaintzen duten atalak identifikatuz, izan proiektu honetan bertan hobetzen saiatzeko, edo izan etorkizunerako lan-lerro moduan.

- **Integrazioa:**

  Planteatutako aukeretatik hautatutako soluzioa CAF Signalling-ek garatutako *Polaris* moduluan integratu beharko da, bere arduradunek espezifikatutako eskakizunak eta interfazea betez.

  Beharrezko proben ostean, aldaketak trenera eraman eta ingurune erreal batean erabili ahal izatea espero da.

  Atal honetan giltzarria izango da gainerako taldekideekin elkarlana; izan ere, orain arteko modulu independente izatetik soluzio orokorrago baten azpi-modulu izatera pasako da proiektuan egindakoa.

3.2.3 Dokumentazioa

- **Memoria:**

  Hilabetetan egindako lana txosten batean islatzea izango da pakete honetako zeregina. Unibertsitateko zuzendariarekin adostuko da memoriak izan beharreko edukien zerrenda zehatza, eta bi asteroko bilera espezifikoak aurreikusten dira proiektuaren azken bi hilabeteetan, gertuko jarraipena egin ahal izateko.

Ataza	**Aurreikuspena** (h)	**Dedikazioa** (h)
Kudeaketa	**50**	**50**
Plangintza	30	15
Jarraipena	20	35
Garapena	**180**	**187**
Ikerkuntza	30	20
Prestaketa	5	5
Diseinua	20	20
Inplementazioa	100	125
Integrazioa	10	2
Analisia	15	15
Dokumentazioa	**80**	**84**
Memoria	65	75
Aurkezpena	15	9
GUZTIRA	310	**321**

3.1 Taula: Ataza bakoitzaren denboren estimazioa eta dedikazio erreala

Dokumentua sareko *Overleaf* LaTeXeditorean idatziko da, zuzendariarekin partekatuta egon dadin. Enpresako arduradunari ere sarbidea emango zaio, bere *feedback*a ere jasotzen joateko, lana aberasteko helburuarekin.

- **Aurkezpena:**

  Defentsaren aurreko azken asteak aurkezpena prestatzeko baliatzea aurreikusten da, hein handi batean. Euskarri grafikoa prestatu eta egindakoa dagokion denboran nola laburtu erabaki beharko da.

3.3 Dedikazioaren estimazioa eta desbiderapenak

3.1. taulan ataza bakoitzarentzat proiektuaren hasieran aurreikusitako denborak eta errealitatean behar izandakoak bildu eta konparatu dira.

Zeregin bakoitzari dagokion denbora azpi-atazetan nola banatu hasieran aurreikustea ezinezkotzat jo zen plangintza egiterakoan, kasu askotan aurreko emaitzekiko dependentziak agertuko zirelako; aitzitik, enpresako gainerako zereginekin batera plangintzatu dira, bi astean behin.

3.4 Arrisku plana

Ia lau hilabeteko iraupena duen proiektu batean kanpotik eragin dezaketen faktoreak ugariak izan daitezke. Arrisku nagusiak identifikatu eta horien eragin negatiboa mugatzeko erabiliko diren estrategiak definituko dira atal honetan.

Alde batetik, proiektuari eskaini beharreko denborak planifikatzerakoan bi izan dira aurkitutako arriskuak: dimentsio honetako lanak antolatzeko esperientzia falta; eta proiektuaren zati askok duten ikerketa izaera. Lehenengoa faktore garrantzitsua izan daiteke atazak (eta ataza bakoitzean eman beharreko denbora) antolatzerakoan akatsak egiteko. Bigarrenarengatik, berriz, ataza batzutan egin beharrekoa aurrekoetan lortutako ondorioen menpe egongo da, eta horrek asko zaildu lezake aurreikuspenak egitea.

Egoera honen aurrean, plangintza osatzerakoan zuzendariei laguntza eskatzeaz gain, badaezpada ere atazen iraupena gorantz estimatu da: kasu guztietan aurreikusi baino ordu gehiago planifikatu dira atazarentzat, espero gabeko luzapenek eta atzerapenek lanaren bilakaera egokia ez oztopatzeko - edo ahalik eta gutxien oztopatzeko, behintzat.

Bestalde, informazioaren balizko galerak ekiditeko hartutako neurriak enpresako arriskuplanaren baitan integratu dira: programatutako kodearen bertsio-kontrola GitLab zerbitzari korporatiboan egingo da, eta Jetson Xavier gailuaren babes-kopiak egingo dira periodikoki, PER taldeko gainerako gailuenekin batera. Salbuespen modura, txostenaren ardura pertsonala izango da, eta proposamena *Overleaf* sareko editorea erabiltzea da, proiektuaren kopia lokalak egiten joateaz gain.

3.5 Lan-metodologia

Lanarekin hasterakoan jarraitu beharreko metodologia eta baldintzak zehaztu dira enpresarekin.

COVID-19aren eraginez, lan presentziala bi txandatan antolatua dago, segurtasun-neurriak mantentzea errazteko. PER taldeak astelehen eta asteazkenetan joateko aukera du (baita bi ostiraletik behin ere); gainerako egunetan, etxetik lan egin beharko da normalean.

Ordutegiari dagokionez, malgutasuna erakutsi du enpresak, aste bakoitzean lanetik kanpoko gertaeren arabera egokitzeko aukera eskainiz (unibertsitateko klaseak, tutoretzak, erosotasuna...). Eskolen ordutegia eta txanden antolamendua kontuan hartuz, bertatik bertara egindako lana hobesteko hautua egin da. Horregatik, astelehen eta asteazkenetan ordu

gehiago sartuko dira, gainerako egunetan lasaiago ibiltzeko. Horrek bide emango du, besteak beste, astearte edo ostegunetan fakultatera joateko, bai klaseetara, eta bai tutorearekin biltzera.

Lanerako modu erdi-presentziala dela eta, normalean bilerak *Microsoft Teams* aplikazioaren bidez egingo dira. Azken urtean asko errotu da enpresan bere erabilera, eta talde barruko zein kanpoko gainerako komunikazioetarako tresnarik erabiliena bihurtu da, posta elektronikoaren kaltetan.

Egunerokoan *software* garapenerako *Scrum* metodologia arina erabiltzen da ATO taldean. Horren baitan, bi asteko *sprint*ak planteatzen dira, bakoitza *Sprint Planning* bilerarekin hasi eta *Sprint Retrospective* eta *Sprint Review* bilerekin amaituz. Epe horretarako banakako zein taldeko helburuak finkatzen dira, eta helburu horietara bidean egin beharreko atazen epe laburreko planifikazioa egiten da (ale xehekoa). Lanketa hori errazteko *Redmine* proiektuen kudeaketarako *software*aren bertsio modifikatu bat erabiltzen da. Proiektu hau ere dinamika horren baitan kokatuko da, eta funtzionamendu hori hartuko da eredutzat.

Proiektuaren aurretik hiru hilabeteko praktikaldia egin zen enpresan, eta horri esker, eguneroko jardunera ohituagoa egoteaz gain, beharrezko baliabideak proiektua hasterako prest izatea dakar: kredentzialak martxan izatea, VPN sarbidea gaitua izatea, tresnak ezagutzea...

3.6 Desbiderapenen analisia

Azkenean espero baino luzexeagoa izan da proiektua: 321 ordu inguru behar izan dira. Proiektuaren tamaina eta ingurune aldakorra direla eta, zaila zen proiektuaren hasieran egindako planteamendua zehatz betetzea. Nahiz eta ordu gutxi batzuk gehiago behar izan diren, horrek ez du arazorik suposatu ez enpresarekin, ez unibertsitatearekin.

Kudeaketan esperotako denbora behar izan da guztira, baina Plangintza eta Jarraipena atazen arteko oreka aldatu da: uste baino azkarragoa izan zen hasierako plangintza, eta astez aste ere, denbora gehiago hartu dute bileretako jarraipeneko gaiek planifikaziokoek baino.

Ikerkuntzan jorratu beharreko gaiei buruzko informazioa bildu eta dokumentatu da. Aurreikusi baino lan azkarragoa izan da, bi arrazoi nagusirengatik: aurkitutakoa bere aldetik idatzi ordez, memoriako dagozkien kapituluak baliatu dira ikerketaren lekukotza moduan;

eta formakuntza ataza honek biltzen zuen arren, ikasi beharrekoen zati bat proiektua hasi aurretik landu zen.

Prestaketan eta Diseinuan esperotakoa behar izan da. Gailuan instalaziorik gehienak eginak zeuden, baina tresna gehigarriak lortu, konfiguratu eta instalatu behar izan dira, dagoeneko eskura zeudenetako batzuk modifikatzeaz gain. Diseinuan bide erdian utzi den integritate-liburutegien erabilera prestatzean zati handia kendu zuen; azken proposamenaren diseinua, berriz, espero baino arinagoa izan zen, azken bertsioa ere erdi-prototipoa baita azkenean.

Inplementazioa luzatu egin da, bitarteko pausu asko eman direlako. Egindako aldaketak probatzeko motorrak berreraiki arte itxaron beharrak prozesua nabarmen moteldu du. Liburutegi eta gai arrotz bateko inplementazioak egoki aplikatzea ez da lan erraza izan, eta egindako zatiak maiz desegin behar izan dira. Bestalde, akatsak zuzentzeak, dependentziak ebazteak... ere denbora hartu dute, eta baita kodearen txukunketa eta errefaktorizazioek ere.

Polaris moduluarekin integratzeko, adostutako interfazea bete da, eta zeinbait proba egin dira. Hala ere, moduluan integratzeko lanik handiena bere arduradunari dagokio, egituraren eta kodearen ezagutza zabalagoa behar baita. Horregatik, lan horretan laguntzea izan da zeregin nagusia. Trenean probatzeko denbora bat ere aurreikusia zegoen, eta ez da erabili.

Analisian datuak eskuratzea nahiko prozesu arina izan da, konparatzeko bi bertsio soilik erabili direlako azkenean. Dena dela, datu horiek prozesatu eta lantzen gehiago pasa izanak, horretarako beharrezko tresnak erabiltzen asmatuz, irabazitako denbora konpentsatu du.

Dokumentazioa ere luzatu egin da; hein handi batean, ikerketan aurkitutakoa ataza honen baitan idatzi delako. Gainerakoan, beharrezko irudiak sortzeak ere nabarmen luzatu du atazaren denbora. Aurkezpenaren euskarria diseinatzea azkarra izan da, memorian erabilitako irudietako batzuk berrerabili baitira.

4. KAPITULUA

Trenen automatizazioa

Lan honen helburuak era zehatzagoan ulertzeko, garrantzitsua da trenen automatizazioak tren-industrian duen pisua ezagutzea, eta baita automatizazio hori lortzeko bidean irudien prozesamenduaren beharraren arrazoiak identifikatzea ere. Izan ere, objektuen detekzioaren inferentziaren optimizazioa eta azelerazioa aurkeztu dena baino problema orokorragoa den arren, bilatzen den aplikazio-kasu konkretu batek eraman du CAF Signalling proiektua proposatzera.

4.1 Trenen automatizazioaren bilakaera

Gainerako garraiobideetan bezala, tren eta trenbideetan ere beharrezkoak dira trafikoa era egokian kudeatzeko sistemak. Horiei esker, azpiegitura ibilgailuen artean modu seguruan partekatzea lor daiteke, egunerokoan bistakoa den moduan: auto ugarik errepide bakarra partekatzen dute, trenbide berean egunean zehar hainbat tren igarotzen dira...

Historikoki trenbidea blokeo-atal, kantoi edo *block section* deiturikoetan banatzen da, uneoro sekzio bakoitzean egon daitekeen tren kopurua mugatuz. Muga hori inplementatzeko mekanismoak izan dira seinalizazioaren historiaren lehen pausuak: seinaleen eskuzko kudeaketa baztertuz blokeo automatikorako *interlocking*- edo katigamendu-sistemak garatu ziren, zirkuitu elektriko eta elektronikoak erabiliz, giza-akatsen eragina murrizteko [1]. Sistema finko horietan, sekzio bakoitzak gutxienez trenbidetik igarotzen den trenik azkarrenak gelditzeko behar duen distantzia bezain luzea izan behar du [2].

Trenbideko azpiegitura automatizatuta ere, trenaren aldetik segurtasuna bermatzea gidariaren ardura zen oraindik ere. Segurtasun-funtzioen kontrola hartzeko sortu ziren 60. hamarkadan trenen babes automatikorako *Automatic Train Protection* (ATP) sistemak [2], trenaren eta burdinbidearen arteko komunikazio automatikoa ezarriz [1].

ATPak eskainitako segurtasun-funtzionalitateen gainean *Automatic Train Operations* (ATO) edo trenen eragiketa automatiko funtzionalitateak erabili daitezke, trena gidatzea bera hein handi batean makinen esku uzteko.

Trenaren eta trenbidearen arteko komunikazioa puntuala izatetik etengabekora igarotzea ekarri zuen *Communications Based Train Control* (CBTC) teknologiak, uneoro trenen datuak era zentralizatuan ezagutzeko eta trenei informazio eguneratua edo aginduak bidaltzeko aukera eskainiz.

Gaur egun ikerketa-arlo aktiboa da trenen automatizazioa. Mundu osoan aurki daitezke adibideak, horien artean [3] Errusian tren autonomoarekin egindako proba pilotua eta [4] Txinan ikusmen artifiziala darabilten tranbiekin egindako saiakerak.

4.1.1 Ontziratutako kontrol-elementuak

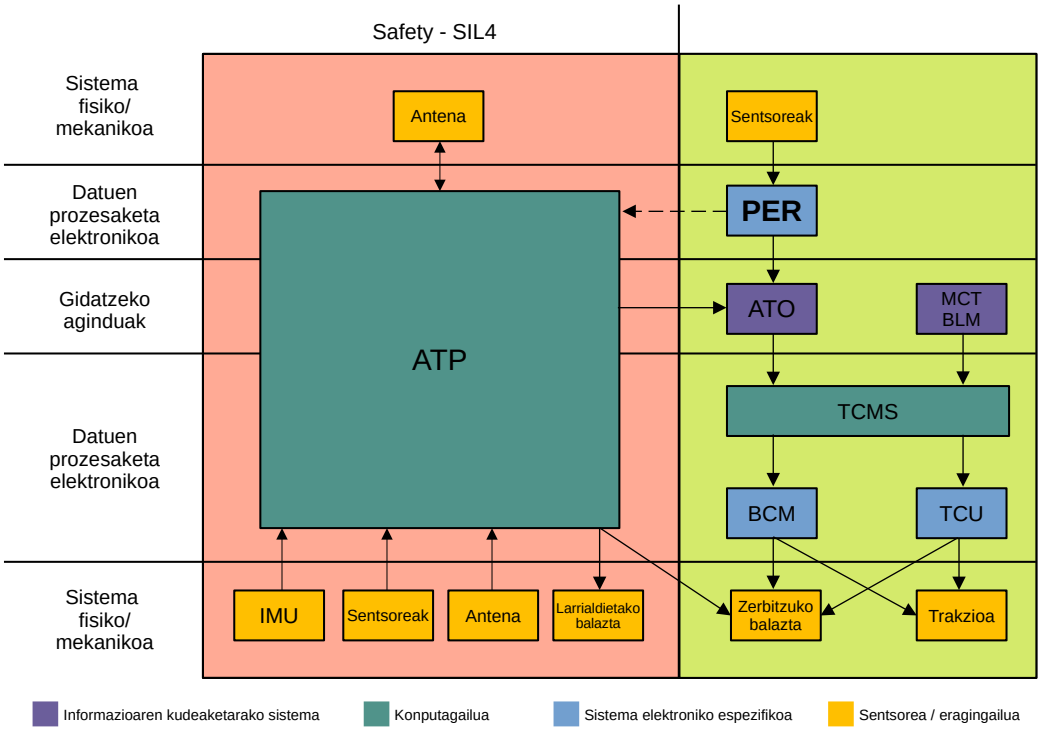
ATP zein ATO sistemak *Trackside* eta *OnBoard* elementuez osatuak egon ohi dira; trenbidekoak eta ontziratutakoak, hurrenez hurren [5] [6].

Automatizazioaren mailatik mailara eta inplementaziotik inplementaziora aldeak egon badauden arren, normalean 4.1 irudian erakutsitako egitura hartu ohi da eredutzat *OnBoard* zatian (diagrama [1] lanetik hartutakoaren itzulpen moldatua da).

Bertan ikus daitekeenez, segurtasun-funtzionalitateez ATPa arduratzen da, treneko sentsoreen (abiadura-sentsorea..) eta *tracksideko* gailuen informazioa jasoz. Funtzionamendu segurua bermatzeko, informazioa emateaz gain balazten gainean eragin dezake zuzenean, abiadura-mugak errespetatzen direla ziurtatzeko. Eragileen banaketa horrek gainerako sistemek segurtasun-betekizunak bete beharrik ez izatea (edo estandar malguagoak betetzea) ahalbideratzen du.

Ondorengoak dira sistemaren osagairik esanguratsuenak:

- **ATP:** *Automatic Train Protection* edo trenen babes automatikoa. Trenbidetik jasotako eta aurrez kargatutako informazioa uztartuz trenaren segurtasuneko mugak ezartzeaz arduratzen da (abiadura-mugak ezartzen ditu, kantoi okupatuetan sartzea eragozten du...).



4.1 Irudia: Trenen kontrol automatizaturako elementuak

- **ATO:** *Automatic Train Opration* edo trenen eragiketa automatikoa. Trena gidatzerakoan gidaria ordezkatzeaz arduratzen da; aldeak alde, beste ibilgailuetan "pilotu automatiko" moduan ezagutzen denaren baliokidea da. Gainerako moduluetatik jasotako informazioa interpretatuz erabakiak hartu eta trenari jakinarazten dizkio.

- **TCU:** *Traction Control Unit* edo trakzioaren kontrol unitatea. Treneko trakzio-sistemaren gainean eragiten du zuzenean.

- **BCU:** *Brake Control Unit* edo balazten kontrol unitatea. Ohiko frenoaz gain, trakzioa ere kontrola dezake, balaztatzearekin laguntzeko.

- **TCMS:** *Train Control and Monitoring System* edo trenaren kontrol- eta monitorizazio-sistema. Aurreko bi kontrol-gailuen eta gidariaren (gizakia zein automatikoa) artean ezarri ohi den abstrakzio-kapa da. Jasotako aginduak interpretatzeaz (era efizienteagoan egikaritzeko) eta zuzenean trenean eragiten duten sistemak monitorizatzeaz arduratzen da.

- **MCT/BLM:** *Master Controller Traction / Brake Lever Manipulator* edo trakzio-

kontrol nagusia eta balazta-palankaren manipulatzailea. Gidarearekiko interfazea osatzen duten gailuen kontroladoreak dira.

- **PER:** *Perception* edo hautematea. Inguruneari buruzko informazioa era independentean jaso (sentsore eta kameren bidez), eta informazio hori prozesatzetik lortutako ondorioak gainerako moduluei igortzen dizkie. Adibideko konfigurazioan ATO eta ATP sistemak soilik elikatzen dituen arren, bestelako erabileretan gidaria laguntzeko ere balio dezake.

Osagaietatik proiektuarekin eta enpresako taldearekin loturarik handiena duten ATP eta ATO sistemei buruzko informazio osagarria A eranskinean irakur daiteke.

4.2 Hautemate modulua

Trenbidetik jasotako edo aurrez ezaguna den informazioaz gain, gidatze autonomorako garrantzitsua da sistemak ingurunearen nolabaiteko zuzeneko pertzepzioa izatea. Horretaz arduratzen da, hain zuzen ere, *PERception* **PER** modulua. Trenean muntatutako sentsoreen bidez kanpoan gertatzen ari dena hauteman eta interpretatzeko gai izan behar du PER azpi-sistemak, gainerako elementuak informazio esanguratsuz hornitzeko. Orokorrean, aurreikusi ezin diren oztopoak detektatzea eta ATPak falta duen informazioa osatzea da sistema honen eginkizuna. Modulu honek oraindik giza gidariaren esku dauden funtzionalitate aurreratuak inplementatzen lagundu dezake, *trackside* azpiegiturarik eskatu gabe, kostua gutxituz eta trenen trenbideekiko bateragarritasuna handituz.

Testuinguruaren eta erabilera-kasuaren arabera alda daiteke erdietsi beharreko datuen izaera: oztopoak, semaforoak eta seinaleak, objektuetara dauden distantziak... Informazio hori eskuratzeko trenaren berezko sistematik kanpoko sentsoreak erabili ohi dira: kamerak, *LiDAR*-a... Azpi-atal honetan moduluaren erabilerarik orokorrenak zerrendatu eta azalduko dira.

4.2.1 Seinaleen detekzioa

Kameretatik jasotako irudiak erabiliz trenbideko seinalizazioaren parte diren elementuak detektatu eta bereizi daitezke (semaforoak, bide-ertzeko seinaleak, gelditze-puntuak etab.). Jasotako irudiei horretarako bereziki diseinatutako algoritmoak aplikatzen zaizkie; sare-neuronal konboluzionalak, oro har [1, 7].

PER moduluak funtzionalitate hau inplementatzen badu, egokia izan daiteke ATP sistema zaharkituak dituzten lineetan tren autonomoak erabili ahal izateko [1].

4.2.2 Oztopoen detekzioa

Trenak aurreikusi gabeko gertaeretatik babesteko, garrantzitsua da aurrez babeserako mekanismoez gain (abiadura-mugak, semaforoak...) ustekabeko gertaerez, trenbidean agertutako oztopoez, ohartzeko gai den sistema bat edukitzea.

Seinaleen detekziorako erabilitako metodo berak moldatzeaz aparte (trenbidea hutsik dagoen edo ez aztertzen duen sare neuronal bat, adibidez [8]), LiDAR [9, 10] eta RADAR [11] tresnak oso erabilgarriak izan daitezke, trenaren aurrean dauden objektuetara dagoen distantzia era zehatzagoan eta fidagarriagoan ematen baitute, baita ikusgarritasuna mugatua denean ere (baldintza meteorologiko txarretan edo gauez, adibidez). Dena dela, nahiz eta gailu horiek fidagarriagoak izan, eta distantzia handiagoak estali, kameretan oinarritutako soluzioak baino garestiagoak eta mantentzeko konplikatuagoak izan ohi dira, erosketa-bolumen txikietan, batez ere.

4.2.3 Distantzien kalkulua

Interesdun objektuetara dagoen distantzia *stereo* kamarak (zehatz lerrokatutako bi kamara, eskuinekoa eta ezkerrekoa), LiDAR-a edo RADAR-a erabiliz kalkulatu daiteke. Kameren kasuan, *stereo matching*eko algoritmoak gai dira kalibrazioko datuak eta ezkerreko eta eskuineko irudien arteko pixelen aldea erabiliz distantziak (sakontasun-mapak) ateratzeko [12]. Estrategiaren arabera, kalkulu hori irudi osoan edo detektatutako objektuen azaleren gainean egin daiteke.

4.2.4 Odometria bisuala

Sentsoreen informazioarekin denboran zeharreko posizio-aldaketak estimatzea da odometria, eta trenbide-industrian erabiltzen diren soluzioek ez dute erabateko zehaztasunik:

[1] Moduluaren funtzionalitate bat teknikoki egingarria izateak ez du produkziorako prest dagoenik esan nahi. Trenen industrian sistema automatikoek araudi eta balioztatze-prozesu zorrotzak bete behar dituzte, eta ekosistema estandarizatutik (trenaren berezko sentsoreak, ATP, ATO...) betekizunetara egokitutako kanpoko sentsoreen eta tekniken erabilerara jauzia emateko asko falta da oraindik.

ohiko odometro edo takometroek (gurpilen birak kontatzen dituztenak), satelite bidezko kokapenak edo inertzia-neurgailuek errorea metatzen dute [13]. Odometria bisualak errore hori gutxitzen lagundu dezake, denboran zehar irudiek jasandako aldea neurtuz posizioaren bilakaera (eta ondorioz, abiadura) estimatuz. Horretarako teknika geometrikoak (irudietatik ertzak edo erpinak bezalako ezaugarriak erauziz, eta ezaugarrien jarraipena eginez) eta ikasketa sakoneko teknikak (era independentean, ikasketa gainbegiratua edo ikasketa ez-gainbegiratuaren bidez; edo sentsoreekin edo teknika geometrikoekin konbinatuz) erabil daitezke [7].

4.3 Automatizazioa CAF Signalling-en

CAF Signalling, CAF taldeko filial teknologikoa da, eta trenen seinalizazioarekin lotutako produktuak merkaturatzen ditu, bai *tracksidek*o azpiegitura, eta bai *onboard* unitateak. Produktu horien artean 1. eta 2. mailako ETCS ATPak [5] eta ETCS sistemen gainean funtzionatzen duen GoA2 ATOa [6, 14] daude.

Martxan dauden proiektuetan hautemate-moduluaren hainbat funtzionalitate ikertzen ari da enpresa: gelditze-puntuak detektatuz GPStik jasotako kokapen-informazioa fintzea; trenbidetik jasotako informazioa kontrastatzeko datuak eskaintzea; ahal den neurrian *tracksidek*o informazioa ordezkatzea, tren autonomoa leku gehiagotan eta modu merkeagoan inplementatu ahal izateko...

4.3.1 Polaris modulua

CAF Signalling-eko PER sisteman konputagailu bidezko ikusmenaren exekuzioaz **Polaris** izeneko modulua arduratzen da: azpi-moduluen arteko exekuzio-fluxua kontrolatzeaz, une bakoitzean dagokionari deituz, eta horiek informazioz elikatuz; egoera-makina inplementatzeaz; erroreak tratatzeaz eta *log*a idazteaz; eta oraindik azpimodulu propiorik ez duten funtzionalitateak inplementatzeaz. Datorren zerrendan exekuzio-fluxua eta erabiltzen dituen azpi-modulu nagusiak ageri dira.

1. Sareko interfaze batetik kabinetako kameren irudi prozesatuak irakurtzen ditu (ezkerreko eta eskuineko kameren irudiak elkarrekin eta bertikalki rektifikatuta).

2. Ezkerreko irudia (eskuinekoa izan zitekeen) **inferentzia-modulua**ri ematen dio,

irudiko objektuen zerrenda jasoz bueltan (objektuen kokapena, tamaina eta kategoriak).

3. Irudiak eta objektuen informazioa distantzien kalkulurako azpi-modulu bati ematen dio. **SterepDepth** denean, detekzioek definitutako eremuan ezkerreko eta eskuineko irudiko pixelen alde horizontala erabiltzen du kalkulurako; **MonoDepth** bada, berriz, detekzioen azalera eta kategoria bakoitzerako ezagunak diren tamainak.

4. **Tracking** algoritmoak ibiltzen ditu, aurreko *frame*tako eta oraingo informazioa erkatuz objektuen eboluzioa kalkulatzeko/aurreikusteko, detekzioen aldakortasuna egonkortzeko.

5. Aurreko azpi-moduluen informazio guztia **ApplicabilityFilter** algoritmo batek iragazten du, aurrez definitutako irizpideen arabera, informazio baliogarriarekin soilik gelditzeko.

6. Hautatutako detekzioak **IfAtoPer** interfazearen bidez serializatu eta sare lokaletik ATOari igortzen zaizkio.

Une honetan 200 milisegunduko zikloa du Polarisek [2]; hau da, segunduko bost iruditako informazioa bidaltzeko gai da. Osagaiak azeleratzeari esker bi helburutako bat lor daiteke, aurrera begira ematen diren pausuen arabera: zikloa murriztea, eta informazioa maizago helaraztea; edo zikloa mantenduz osagai gehiago txertatu ahal izatea.

Azalpen horietan bistakoa da hasiera batean behintzat Polaris-eko azpi-moduluak elkarren menpekoak direla; era sekuentzialean exekutatu beharrekoak, beraz. Azeleraziorako lehen pausua azpi-moduluetako bakoitzaren exekuzio-denborak (zikloak) murriztea da, eta gaur-gaurkoz horietatik motelena optimizatzean zentratzen da lan hau: inferentzia-modulua.

Testuinguru horretan, ondorengo exekuzio-interfazea bete beharko du modulu berriak:

```
DetectionList infer(cv::Mat &image, float threshold);
```

`DetectionList` zerrenda-mota Polaris-en definitzen da, eta detekzio bakoitzeko bere koordenatuak, tamaina, konfidantza, distantzia(k), aplikabilitateari buruzko informazioa... gordetzen ditu. Objektuak hasieratzeaz eta kokapenari eta motari buruzko informazioa idazteaz inferentzia-modulua arduratzen da; gainerako eremuak, berriz, bakoitzari dagokion moduluan betetzen dira.

[2] Ziklo hau irudia jasotzen duenetik ATOra detekzioak igorri artekoa da. Ez du kontuan hartzen kabinetako kameretako irudiak aurreprozesatzeko denbora (beste gailu batzuetan egiten da).

5. KAPITULUA

Objektuen detekzioa

Irudietan objektuen detekzioa gauzatu ahal izateko oinarri teorikoak ezarriko dira kapitulu honetan. Horretarako, sare neuronalak, bereziki irudiak prozesatzeko erabiltzen diren sare konboluzionalak eta haien konputazioa azkartzeko metodoak aurkeztuko dira. Jarraian, PER taldean une honetan erabiltzen ari diren eta, ondorioz, lan honetan erabili den YOLOv4 arkitektura espezifikoa azalduko da. Azkenik, inferentzia azeleratzeko erabiltzen diren *hardware* azeleragailurik ohikoenak zerrendatu dira.

Objektuen detekzioaren problema ikusmen artifizalaren arloko oinarrizko problemetako bat da, sailkapenarekin eta irudien segmentazioarekin batera [15]: kategoria-multzo finitu bat emanik, irudi batean mota horietako zenbat objektu dauden, non dauden eta zein klasetakoak diren ebatzi behar da. Datu meatzaritzako teknika tradizionaletatik (irudien eremuetan SVG sailkatzaileak aplikatzea, HOG histogramak...) sare neuronalen bidezko ikasketa sakonera jauzia eman da azken urteotan [16], 2012an aurkeztu zen AlexNet[1] sare iraultzaileak irekitako bidea jarraituz.

5.1 Sare neuronalak

Sare neuronal (artifizial) edo ***(Artificial) Neural Network*** garun biologikoetan inspiratutako programazio-paradigma da [19]. Oinarrian, neuronen funtzionamendua eta euren arteko loturen funtzionamendua kopiatzen da, adimen artifizialeko gainerako tekniken

[1] Alex Krizhevsky sortzailearen izena hartzen duen CNN sareen aitzindaria[17]. 2012ko *ImageNet* [18] lehiaketa irabazi zuen, txapeldunordearen errorea nabarmen hobetuz.

helburu berarekin[2]: ohiko algoritmo-ereduen bidez ebazteko zailak diren problemen soluzioak programatzea.

[21] artikuluak dioenez, sare neuronalak 1944an Chicagoko Unibertsitateko (AEB) Warren McCullogh eta Walter Pitts-ek proposatu zituzten lehen aldiz, eta ordutik konputazio-zientzien alorrean izan duten garrantzia eta ospeak gorabehera nabarmenak izan ditu: 1969an bigarren plano batera pasa ostean, 80ko hamarkadan berriz ere ikerketa-lerro garrantzitsu bihurtu ziren, milenio berriaren lehen urteetan indarra galtzeraino (2000-2010, ggb.). Egoera guztiz irauli da azken urteetan, konputazio-ahalmenaren igoera orokortuari eta *hardware*aren merkatzeari esker [7].

Datozen azpiataletan[3] [19] liburuko informazioan oinarritutako azalpen labur bat dator, sare neuronalen oinarrian dauden neuronei eta sare-antolaketari buruz. Proiektuarekin lotura zuzenik ez duen arren, sareen entrenamendu-prozesua ezagutzea garrantzitsua izan da sare neuronalak hobeto ulertzeko. Horregatik, B eranskinean entrenamenduari buruzko informazio gehigarria txertatu da.

5.1.1 Neurona artifizialak

Neurona artifizialak dira sareen oinarrizko osagaiak. n sarrera dituen neurona batean, sarrera horietako bakoitzak w_i **pisua** izango du esleitua, eta neuronaren irteera sarreren pisuekiko batura ponderatuaren araberakoa izango da. Balio horri ***bias*** edo, literalki, joera bat esleitzen zaio, neuronaren portaera sarrerekiko modu independentean doitu ahal izateko. Irteera kalkulatzeko, z batura osoari f **aktibazio-funtzioa** aplikatzen zaio (5.1 ekuazioa). Diseinuaren arabera, aktibaziorako hainbat funtzio lineal eta ez linealen artean aukeratu daiteke. Neuronen sarrera-irteerak zenbaki oso edo errealak izan daitezke, kasuaren arabera.

$$y = f(z) = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{5.1}$$

[2]Ohikoa da terminologia lausoa agertzea literaturan, *Artificial Intelligence*, *Machine Learning*, *Neural Networks* eta *Deep Learning* kontzeptuen artean (Adimen Artifizial, Ikasketa Automatiko, Sare Neuronal eta Ikasketa Sakon, hurrenez hurren). Lan honetan [20] erreferentziako interpretazio bera erabili da: $DL \subset NN \subset ML \subset AI$.

[3]5.1.1, eta 5.1.2 azpiatalak.

Aktibazio-funtzioak

Aktibazio-funtziorik oinarrizkoena funtzio bitar mailakatua da: irteerak 1 balioa hartuko du, z aurrez definitutako *threshold* balio bat baino handiagoa bada; 0, bestela. Begibistakoa da horren arazo nagusia: zaila da pisu baten (edo *bias*aren) aldaketak irteeran izango duen eragina neurtzea; are gehiago, neurona bakarraren ordez sare oso bat doitu nahi denean. Aldaketa handiekin irteera berdin mantendu daiteke, z oso positiboa edo oso negatiboa denean; edo aldaketa txiki batekin guztiz alderantizkatu, 0 izatetik 1 izatera, z 0 baliotik gertu dagoenean.

Arazo hori larria da sareak entrenatzerakoan. Entrenamenduaren azpiatalean azalduko denez, ikasketa gainbegiratuan, sareak entrenatzeko irteeraren eta esperotakoaren arteko diferentzia gutxitzeko aldaketa txikiak egiten dira pisu eta *bias*etan, eta hobekuntza lortzeko beharrezkoa da aldaketa horien eragina kontrolatu eta neurtzea.

Horregatik, normalean aktibazio-funtzio aurreratuagoak erabiltzen dira, sarreraren baitako aldaketa leunagoa dutenak. Horien artean ezagunenak *Sigmoid* edo $\sigma(x)$, *Hyperbolic Tangent* edo $tanh(x)$, *Rectified Linear Unit* edo $ReLU(x)$, *LeakyReLU* edo $LReLU(x)$ eta *Softplus* edo $soft(x)$ dira (informazio gehiago [22] artikuluan).

5.1.2 Sareen egitura

Biologian gertatzen den moduan, neurona soil bat bere horretan ezin da elementu "adimentsu"moduan hartu; horregatik, problemak ebatzi ahal izateko, neuronen irteerak besteen sarrerekin kateatuz sare-egiturak eratzen dira. Egitura horietan neuronak ***layer*** edo geruzatan antolatzen dira, normalean, geruza bakoitzari funtzio espezifikoa esleituz.

Urte luzetako ikerketaren ondorioz, problema zehatzei soluzioak emateko hainbat eta hainbat sare-familia agertu dira, bakoitza bere ezaugarri espezifikoekin; irudiekin lan egiteko erabiltzen diren **sare neuronal konboluzionalak**, esaterako.

Hala ere, proposamen guztietan hiru motatako geruzak bereizi ohi dira, sarean duten kokapenaren arabera:

- **Sarrerako geruza** edo *Input Layer*: sareren errealitatearekiko sarrerako interfazearen funtzioa betetzen du. Ebatzi nahi den problemaren informazioa (datuak) jasotzen ditu, eta beharrezkoa denean, hurrengo geruzentzat aurreprozesatzen.

- **Irteerako geruza** edo *Output Layer*: sareak egindako kalkuluen emaitzak informazio ulergarri bihurtzen ditu; irteerako interfazea da, beraz. Kasu honetan ere, informazioaren formatua problematik problemara aldatzen da.

- **Geruza ezkutua** edo *Hidden Layer*: sarrerako eta irteerako geruzen arteko gainerako kapa guztiak dira. Sareak jasotako datuetatik informazio esanguratsua erauztea da haien lana.

Sareak, berriz, geruzen arteko loturen norantzaren arabera ***Feed Forward*** eta *Recurrent* eratakoak izan daitezke: lehen kasuan informazio-fluxua norabide bakarrekoa da, eta geruza baten sarrera ez dago inoiz bere irteeraren menpe; bigarrenean, ordea, tarteko geruza bat erabiliz kapa baten irteera berriz ere bere sarrerarekin lotu daiteke. Analisi teoriko honetan azaldutakoa *Feed Forward* erakoei aplikatzen zaie.

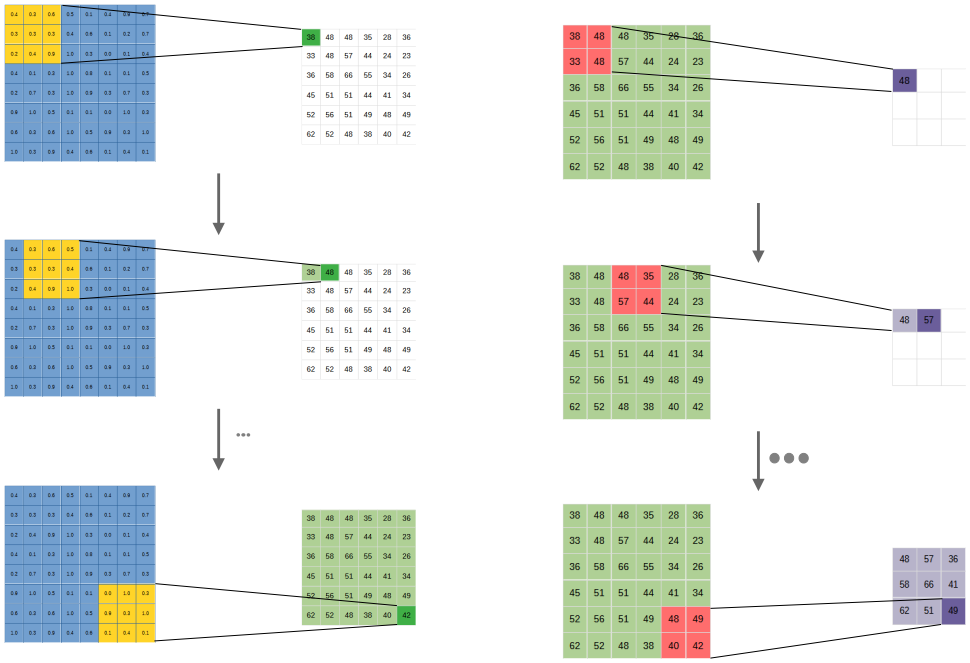
5.2 Sare neuronal konboluzionalak

Sare konboluzionalen (*Convolutional Neural Network*, CNN) helburua sarrerako irudiko ezaugarriak [4] automatikoki erauztea da, horretarako geruza konboluzional edo *Convolutional Layer* deituriko kapa bereziak erabiliz.

Geruza bateko neurona guztiak hurrengoko guztiekin lotu ordez (*Fully Connected* sareetan gertatzen den bezala), ***Convolutional Layer*** geruzetako neurona bakoitza sarrerako datuen azpi-multzo batez arduratzen da - bere *Local Perceptive Field* eremuaz. Eremu guztiek pisu eta *bias* bera partekatzen dute, sarrerako irudiaren eremu bakoitzetik ezaugarri edo *feature* bera erauziz. Pisu eta *bias* multzoari iragazki edo *Filter* deritzo; eragiketaren irteerari, berriz, ezaugarri-mapa edo ***Feature Map***. Iragazkia irudi osoan zehar desplazatzen da, *stride* balioa adina pixel pausu bakoitzean, eta sarrerako datuen eremuaren eta iragazkiaren arteko matrize-biderketak ematen du ezaugarri-mapako elementuari dagokion balioa. Eredu hau orokortuz, normalean geruza konboluzional bakarrak ezaugarri-mapa ugari erauzi ohi ditu.

Hainbat *Convolutional Layer* kateatzen dituzten egitura konplexuetan, geruzen artean ***Pooling Layer*** deituriko kapa sinplifikatzaileak txertatzen dira. Ezaugarrien mapako eremu bateko datuak balio bakarrean laburbiltzen dira, maximoa (*Max Pooling*) edo batez-

[4] Ezaugarriak ertzak, erpinak edo bestelako formak izan daitezke. *Deep Learning* tekniken arazoetako bat da zenbat eta sakonagoa, orduan eta zailagoa dela gertatzen ari dena mundu errealeko baliokide sinpleen bidez azaltzea. Oro har, ezaugarria irudiko datuen patroi ezkutu moduan uler daiteke.



(a) Sarrerako datuen konboluzio-eragiketa **(b)** Ezaugarri-maparen *pooling* eragiketa

5.1 Irudia: Sare konboluzionaletako iragazki baten oinarrizko eragiketak

bestekoa (*Average Pooling*) aukeratuz. Normalean eragiketa honetako eremuak elkarren artean disjuntuak izaten dira.

Convolution eta *Pooling* eragiketek abantaila ugari eskaintzen dituzte; horien artean:

- Irudien ezaugarriak era automatikoan ondorioztatzen dira, irudien izaerari buruzko aurretiazko ezagutzaren beharrik gabe. Horrek sare-arkitektura berak testuinguru eta egoera ezberdinetara egokitzea errazten du.
- Doitu beharreko parametroen kopurua murrizten da, pisuak eta *bias*ak partekatzeari esker.
- *Pooling* eragiketak, informazioa sinplifikatzearekin, *overfitting* arazoa ekiditen laguntzen du.

5.1 irudian konboluzioaren eta *pooling*-aren erabileraren adibide bat ikus daiteke, geruza bateko iragazki bat. 5.1a irudiak 8×8 tamainako sarrerako matrizeari (urdina) 3×3

iragazki bat (horia) aplikatzen zaio, $stride = 1$ izanik. 5.1b-n prozesu horretatik lortutako 6×6 (filtroak 6 mugimendu egin ditzake ertzera iritsi aurretik) *feature map*ari (berdea) 2×2 *pooling* eragiketa aplikatzen zaio (gorria), 3×3 tamainako matrizeak lortuz (morea) [5]. F *feature-map* matrizeko i, j elementuaren balioa kalkulatzeko A sarrerako matrizearen dagokion zatiaren eta W iragazkiaren arteko elementuz elementuko biderketaren elementuen baturari f aktibazio-funtzioa aplikatzen zaio, 5.2 ekuazioak erakusten duen moduan.

$$F_{i,j} = f\left(b + \sum_l \sum_k w_{l,k} a_{i+l,j+k}\right) \tag{5.2}$$

Problema bat ebatziko duten sare neuronal konboluzionalen diseinuan, egitura konplikatzen duten elementuak agertzen dira: koloretako irudiek hiru kanal izan ohi dituzte; geruza bakarrean ezaugarri ugari erauz daitezke; konboluzioaren irteeratik problemak eskatzen duen *output* espezifikoa lortu behar da[6]... Gainera, bi estrategia jarraitzen dira detekzioaren emaitzak hobetzeko: bi etapatako detektagailuek irudiko eremu interesgarriak identifikatzen dituzte lehenik, sailkatzaileen bidez proposamen horiek prozesatzeko; etapa bakarrekoek, berriz, irudi osoa aztertzen dute. Lehen motakoek emaitza hobeak lortzen dituzten arren, bigarrenak azkarragoak dira, eta erabilgarriagoak denbora errealeko testuinguruetan.

Objektuen detekzioan erabiltzen diren sare konboluzionalek hiru atal izan ohi dituzte: ***backbone*** edo bizkarrezurrak ezaugarriak erauzten ditu; ***neck*** edo lepoak ezaugarri horiek nahastu eta egituratzen ditu, informazio semantikoa (nolakoa) eta espaziala (non) konbinatuz; eta ***head*** edo buruak informazio horrekin muga-laukiak eta klase-probabilitateak kalkulatzen ditu.

Eragiketa berri hauek entrenamenduaren hasierarako finkatu beharreko hiperparametro berriak dakartzate arazora, geruza-kopuruaz gain: kapa bakoitzeko iragazki-kopurua eta haien tamaina eta *stride* balioa; *pooling* eragiketa-mota (*max* edo *average*)...

[23, 24] artikuluek azken urteotan CNN sareen alorrean gainditu diren mugarriak azaltzen dituzte, gaur egungo teknikak eta proposamenak aztertuz. *Feedforward* motako sare konboluzionalik esanguratsuenen artean daude aitzindari izan zen 2012ko AlexNet [17], 2014ko GoogLeNet [25], 2015eko *Visual Geometry Detector* edo VGG [26], 2016ko *Single Shot Detector* edo SSD [27] eta datorren atalean sakonduko den YOLO familia. *Recu-*

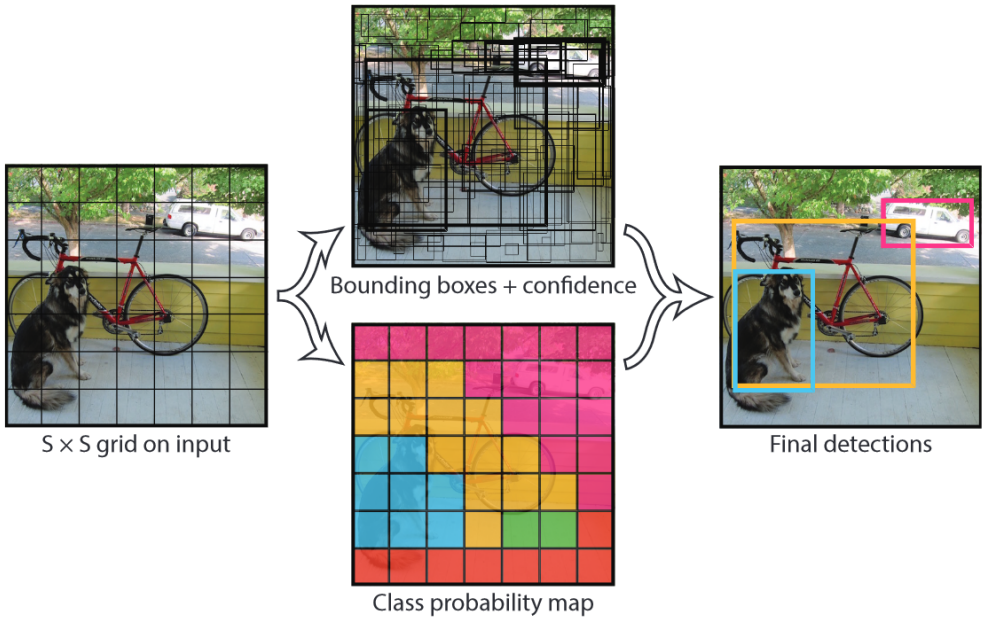
[5] 5.1 irudiko adibidea modu honetan definitu da: $\forall i, j \in \{0,...,7\}$ $0.0 \leq A_{i,j} \leq 1.0$; $\forall l, k \in \{0,...,5\}$ $w_{lk} = 10$; $b = 1$; $f(x) = x$; *pooling* eragiketa *max-pooling*.

[6] Elkarrekin kateatutako *convolutional* eta *pooling* geruzen segidari *fully connected* motako azpi-sare bat eransten zaio irteeran, informazioa erabilgarri bihurtzeko.

rrent erakoetan, berriz, *Recurrent-CNN* edo R-CNN [28] eta bere eratorpenak [29] aipatu ohi dira.

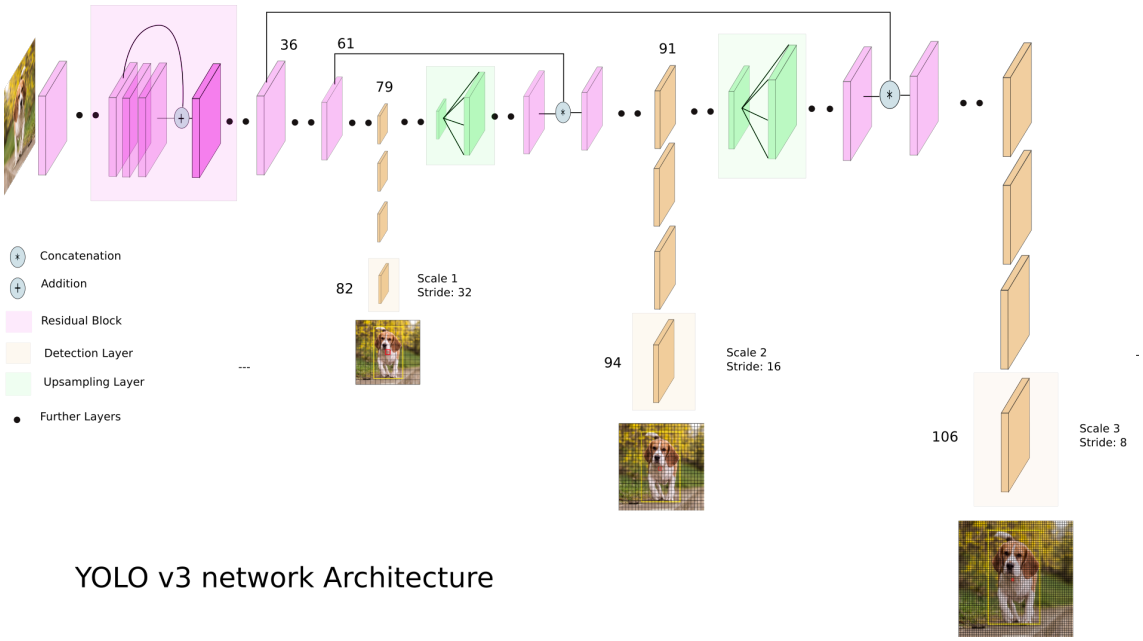
5.3 YOLO arkitektura

2016. urtean objektuak detektatzera zuzendutako YOLO izeneko sare neuronal konboluzionala aurkeztu zuten [30]: ***You Only Look Once***. Etapa bakarreko detektagailu honek inferentzia bakarrean kokapena deskribatzeko muga-laukiak edo *bounding box*ak eta klase-probabilitateak kalkulatzen ditu, arazoa erregresio-problema moduan planteatuz. Irudia $S \times S$ tamainako sareta batean banatzen da, eta gelaxketako bakoitza B kutxa iragartzeaz arduratzen da. Kutxa bakoitzak objektu izateko duen probabilitatea ($P(obj)$) eta objektua bada klaseetako bakoitzekoa izateko duen probabilitate baldintzatua ($P(C_i|obj)$) kalkulatzen dira; detekzio bat c_i klaseko objektu izateko probabilitatea, beraz, $P(obj) \cdot P(c_i|obj)$ da. Horretarako 24 geruza konboluzional (batzuen artean *maxpool* kapekin) eta bi *fully connected* (irteeran) erabiltzen ditu, konboluzioan *Leaky ReLU* aktibazioarekin. 5.2 [30] irudiak YOLOv1 sarearen detekziorako prozesuaren interpretazio logikoa erakusten du: objektu izateko probabilitatea *threshold* batetik gora duten muga-laukiak aukeratu, eta bakoitzari probabilitate-maparen arabera dagokion klasea esleitzen zaio.



5.2 Irudia: YOLOv1 sarearen detekzio-prozesua

Ondorengo urteetan arkitekturaren hedapen eta hobekuntzak agertu ziren, lehen bertsioa



5.3 Irudia: YOLOv3 sarearen arkitektura

sortu zuen taldeko Joseph Redmond eta Ali Farhadi-ren eskutik. **YOLOv2** edo YOLO9000 [31] sareak azken *fully connected* geruzen bidez muga-laukiak detektatu ordez, entrenamendu garaian definitutako *anchor boxes* edo aingurak erabiltzen ditu mugak ondorioztatzeko, bestelako hobekuntzen artean [7]. 2018an **YOLOv3** bertsioa aurkeztu zuten [33], batez ere eskala ezberdinetako detekzioak ahalbideratuz: irteerako hiru geruza daude egitura honetan, bakoitzak eskala bateko detekzioa egiteko, piramide-egitura jarraituz (ikus. 5.3 [33] irudia). Bereizmenik handieneko geruzak objekturik txikienei buruzko informazioa emango du; bereizmenik txikienekoak, berriz, objekturik handienei buruzkoa. Bestalde, klaserik egokiena aukeratzeko *softmax* funtzioa erabili ordez, erregresio linealeko teknikak erabiltzen ditu, klase bakoitzaren konfidantza era independentean kalkulatzeko, *multi-label detection* egiteko aukera eskainiz (objektu bera klase baten eta bere azpi-klasearena izan daiteke aldi berean, seinalea eta semaforo gorria, adibidez).

2020an YOLOv3 sarean oinarritutako **YOLOv4** aurkeztu zuen beste ikerketa-talde batek [34]. YOLOv3 sarearen *backbone*a (Darknet-53[8], moldatua) *Spatial Pyramid Pooling*

[7] *Batch normalization* erabiltzea, kokapenak $[0, 1]$ tartean bornatzea, eskala anitzeko irudiekin entrenatzea... [31, 32].

[8] YOLOv2, v3 eta v4-k *Darknet* familiako bizkarrezurrak eta izen bereko *framework*ak erabiltzen dituzte. [35] GitHub-eko biltegian kodea ikus daiteke.

[9] eta *Path Aggregation Network* [10] bezalako teknikekin konbinatuz hedatzen du (*neck*), sareari berari aldaketa txikiak egiteaz gain (*mish* [38] aktibazio-funtzioa erabiltzea, adibidez). Amaieran YOLOv3-ren *head* bera ibiltzen du. Literaturan aurkitzen diren *benchmark*etan oso emaitza onak lortzen ditu, zehaztasunaren eta abiaduraren arteko oreka lortuz, eta proiektua egiterako unean denbora errealeko objektuen detekziorako *State of the Art* kontsideratzen da [11].

5.4 Azelerazio-teknikak

Sareak hazten joan diren heinean, gero eta konplexuagoak dira egin beharreko eragiketak, bai entrenamenduan, bai inferentzian. Matrize- eta bektore-eragiketen segida handiak dira funtsean sareak, eta horrek *Single Instruction Multiple Data* edo SIMD motako paralelismoa oso erabilgarri bihurtzen du horiek azeleratzeko: eragiketa berak behin eta berriz errepikatu behar dira datu ezberdinen gainean.

Hainbat modutan lortu ohi da paralelizazio hori: CPUan bertan exekuzioa optimizatuz, edo/eta azeleragailuak erabiliz. Azken multzo horretan sartzen dira GPUak, inferentziarako bereziki garatutako gailuak eta diseinu espezifikoak egikaritzen dituzten FPGAk. Kasu hauetan, problema orokorrak ebatzi ahal izateko, ko-prozesagailu moduan funtzionatzen dute, CPUan hasieratzen eta kontrolatzen den exekuzio-fluxuaren zati jakinak (helburuarekin bat datozenak) gailuan exekutatuz.

5.4.1 CPU

Azken urteotan hainbat saiakera egin dira sare neuronalen inferentzia CPUetan bertan azeleratzeko, SIMD bektore-eragiketen bidez. 2011n Googleko lantalde batek [40] artikuluan CPU bidezko azeleraziorako oinarrizko teknikak bildu zituen ahots-ezagutzako sare

[9] Tamaina anitzetako sarreretatik tamaina finkoko ezaugarriak erauzteko gai den ikusmen artifizialeko teknika, CNN sareetarako 2015ean proposatu zena [36]. Ale larriko eta xeheko *pooling* eragiketak konbinatu daitezke geruza bakarrean.

[10] Iragazkien konplexutasuna handitu ahala ezaugarri zehatzagoak erauzi daitezke, baina bereizmen espaziala txikitzen da (zailagoa da objektu handiei buruzko informazioa lortzea). Geruza bakoitzak funtzio bat du, eta PAN bakoitzaren informazioa sarean zehar era efizienteagoan hedatzeko erabiltzen da; *shortcut* edo bide-zidorren bidez, kapa guztiak zeharkatu beharrean [37].

[11] COCO edo *Common Objects in COntext* Microsoftek sortutako objektuen datu-baseen multzoa da, objektuek detekzioko teknikak garatu eta ebaluatzeko erabltzen dena. Ohar hau idazterakoan *Real-Time Object Detection* atazan 15. zehaztasunik handiena eta 2. inferentzia-denborarik baxuena dauka 608x608 sarrerako YOLOv4 modeloak (bere *tiny* bertsioa soilik da azkarragoa, zehaztasun gutxiagorekin) [39].

bat adibide moduan hartuta, x86 arkitekturako Intel eta AMD prozesagailuekin: datuen kokapenaren optimizazioa, multzokatzea (*batching*), SSE (*Streaming SIMD Extensions*) bektore-aginduen erabilera... 2019ko uztailean Amazon enpresako ikertzaileek NeoCPU optimizaziorako liburutegi irekia aurkeztu zuten, CNN sareetako inferentzia-prozesu osoa CPU-etarako optimizatzeko [41]. 2021ean Standord unibertsitatean *sparse*[12] motako sareak CPUrako optimizatzeko SparseDNN optimizatzaile propioa aurkeztu dute, orain arteko gainerako *frameworkek* lortutako emaitzak hobetuz [44].

Sare neuronalen CPUko exekuzioa hobetzeko esfortzu berezia egin du Intel etxeak. Maila orokorrago batean, OpenVINO *Toolkit* Intel *hardware*etarako sare neuronalen eragiketak optimizatzeko tresna-multzoa da [45]. CPUetan zentratuz, *Deep Learning Boost* izeneko paketean bateratu ditu hainbat tresna [46]: AVX-512 bektore-eragiketen familia; VNNI (*Vector Neural Network Instructions*) eragiketa-multzo berria; *bFloat16* datu-mota...

5.4.2 GPU

GPU edo *Graphics Processing Unit* ko-prozesagailuak irudi/grafikoak mugitzeko sortu ziren, baina problemaren izaera paralelizagarriak eta hori soluzionatzeko diseinuak (hari asko, memoriaren banda-zabalera handia, konputazio-ahalmen handia...) oso egokiak bihurtu ditu bestelako eskakizun altuko kalkuluak ere egiteko, *General Processing* GPU edo GPGPU paradigmari bide emanez [47].

Helburu orokorreko GPU programazioan Nvidia-k eskuratu du merkatuko lidertza[13], eta bere gailuek emaitza onak lortu diruzte ertzeko konputazioan ere [50]. Nvidiaren GPUetan programatzeko CUDA plataformak liburutegi espezifikoak, arazketarako tresnak, `C/C++` konpilatzailea etab. eskaintzen ditu [51]. GPU bidezko azeleraziorako *de facto* estandarra bihurtu da, nahiz eta gainerako markekin bateraezina izan. Horrez gain, cuDNN libreriak abstrakzioa eskaintzen du, sare neuronaletan ohikoak diren eragiketak inplementatuz. Horietan oinarrituta, maila altuagoko optimizazio-liburutegi bat ere inplementatu du Nvidiak: *Tensor Real Time* edo TensorRT.

Nvidiaren unibertsotik kanpo, alternatiba OpenCL (*Open Computing Language*) da, beste ekoizleen GPUetan programatzeko ere balio baitu, eramangarritasuna eta bateragarritasu-

[12] Neurona guztiak ez dira gainerako guztiekin konektatzen, eta inferentzia-/entrenamendu-prozesu batean ez dute denek parte hartzen, edo ez dira denak beharrezkoak. *Sparse Netowrks* sareak ezaugarri horiek baliatzen saiatzen dira kalkuluak sinplifikatzeko, parametroak arbuiatuz. Informazio gehiagorako, ikus. [42, 43].

[13] Adibidez, 2020ko azaroko top500 zerrendako azeleragailuen/ko-prozesagailen gehiengoa Nvidiak eraiki zuen [48, 49].

na hobetuz. [52] artikuluak OpenCL-ren programazio-eredua azaltzen du: terminologia aldatzen den arren, CUDA-renaren oso antzekoa da. Ereduz aldatzerakoan jasaten den errendimenduaren galera da desabantailetako bat, eta horren azterketa egiten da [53] lanean.

5.4.3 ASIC

Azken urteotan sare neuronalen inferentziara bideratutako ASIC edo *Application Specific Integrated Circuit* produktuak agertu dira. ASIC zirkuitu hauek helburu zehatz bat betetzeko diseinatzen dira, eta erabilitako elektronika guztia (transistore guztiak) helburu horretara bideratzeak errendimendu- edo efizientzia-abantaila eskaintzen die, helburu orokorreko gailuekin alderatuz. Zirkuitu horien CNN sareetarako diseinatzeko azken urteetan egin den ikerketa zientifikoa biltzen du [54] artikuluak, erabilitako teknikak eta aurkikuntzak sailkatu eta azalduz.

Azken aldian maiz entzuten dira gailu eramangarrietako NPU edo *Neural Processing Units* gailuei buruzko berriak, gero eta ohikoagoa baita *smartphone*-ek adimen artifizialera bideratutako kontsumo baxuko ASIC txikiak izatea, prozesagailu nagusiari laguntzeko [55, 56]. *Cloud computing* zerbitzuetan, Google-ren TPU [57] eta Amazon-en Inferentia [58] gailuak aipagarriak dira, enpresa horiek euren sareko plataformetan azeleragailu moduan eskaintzen baitituzte. Ertzeko konputazioan, ostera, ASIC soluzioen adibide garbia proiektuan erabili den Jetson Xavier AGX plakako bi NVDLA gailuak dira (6.4 atalean azaltzen dira).

5.4.4 FPGA

Field Programmable Gateway Array edo FPGAk fabrikazio-prozesuaren ostean konfiguratu/diseinatu/programatu daitezkeen gailuak dira, euren bizi-zikloak zehar zirkuitu integratu ezberdinak inplementatu ditzaketenak. Malgutasun horrek ASIC soluzioetan erabilitako diseinu espezifikoak eraikitzeko alternatiba egoki bihurtzen ditu produkzio bolumena baxua edo ertaina denerako (ASIC gailuak fabrikatzea errentagarria ez denerako), prototipoak eraiki nahi direnerako (aldaketak, zuzenketak... egin ahal izateko, produkziora igaro aurretik) eta problema aldakorrei erantzun behar zaienerako (superkonputagailuen koprozesadore moduan erabiltzeko, adibidez).

[59] liburuko 2. atalean FPGA gailuen arkitekturari buruzko informazio orokorra eskaintzen da, eta bertako informazioa laburtu da datozen puntuetan.

- Bertsio ugari dauden arren, normalean osagai komunak izaten dituzte: funtzio logikoak inplementatzeko erabiltzen diren *Configurable Logic Block* edo CLBak; blokeen arteko komunikazioa gauzatzeko lotura-sare aldakorrak; eta txipetik kanpoko konexioetarako S/I blokeak.

- Azpiegitura elektronikoaren erabilera efizientea egiteko, ale xeheko programazioaren (transistore-mailakoa) eta ale larrikoaren (prozesagailuak) arteko oreka bilatuko duten teknikak erabiltzen dira. Saltzaile komertzialen artean, *Look Up Table* (LUT) edo egia-taulak dira erabilienak CLBak eraikitzeko, LUT eta D edo JK biegonkor (*flip-flop*) batez osatutako hainbat BLE (*Basic Logic Element*) konbinatuz bloke logiko bakoitzean, nodo barruko lotura-sare lokalen bidez.

- CLBak elkarren artean konektatzeko erabilitako komunikazio-sareak gailuaren azalera elektronikoaren %80-90 har dezake, eta FPGA osoaren malgutasunak dependentzia zorrotza du lotura-sarearenarekin. Bi topologia-familia nagusi bereizten dira: *Island-Style Routing Architecture* eta *Hierachical Routing Architecture*. Lehenengoan nodoak (CLBak) 2D maila batean antolatzen dira, eta elkarren arteko loturak *Switch Box* eta *Connection Box* deituriko bloke programagarrien bidez definitzen. Bigarrenean, diseinuetako konexioek jarraitu ohi dituzten lokaltasun-patroiak ustiatzea da helburua, nodoak *cluster* edo multzotan banatuz, eta azken hauek multzo handiagotan, zuhaitz-egitura bat osatuz (zuhaitzaren maila bakoitzaren barruko komunikazioak azkarragoak izango dira).

- Aplikazioak FPGAtan inplementatzeko *Computer Aided Design* (CAD) fluxua jarraitzen da. *Hardware Description Language* edo HDL lengoaien (VHDL, Verilog...) bidez deskribatutako zirkuituak interpretatu eta sintetizatzeko gai diren tresnak eskaintzen dituzte fabrikanteek, kasuan kasuko azpiegiturara hobeto egokitzen den behe mailako diseinuaren (bloke- eta lotura-mapak) *bitstream* edo bit segida sortuz, eta horrekin gailuaren memoria-bitei balioak esleituz (azken hauek zehazten dute FPGAren funtzionamendu logikoa).

FPGA gailuen aplikazioetan aurrerapen handiak eman dira iragan laburrean, eta asko ikertu da sare neuronalen inferentzia nola bizkortu gailu horiek erabiliz. [60, 61, 62] artikuluek CNN sareen FPGA gaineko inferentziaren egoera aztertu eta ikertu dute, eta erreferentzia baliagarriak dira gaian gehiago sakontzeko. [63] lanean, berriz, bereziki FPGAtan exekutatua izateko *framework* bat diseinatu eta proposatzen dute.

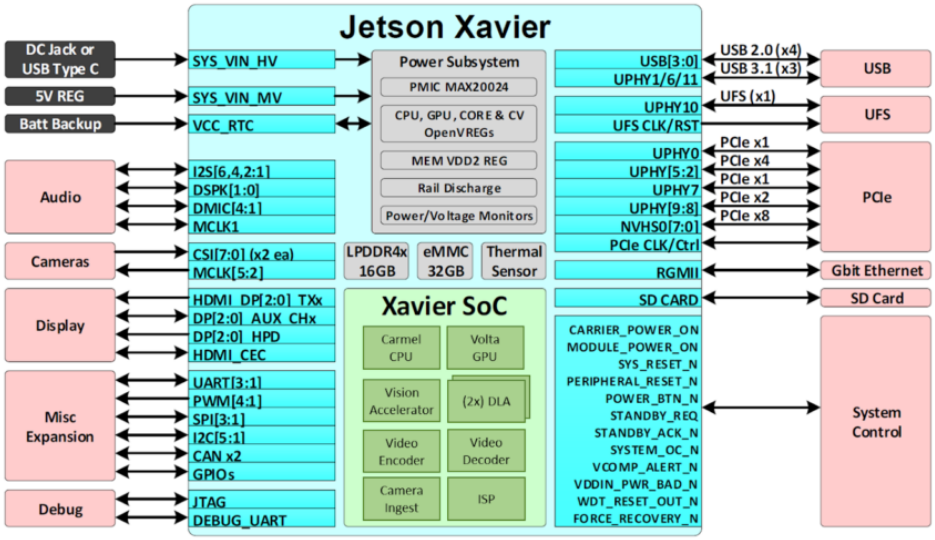
6. KAPITULUA

Hardware azpiegitura

Ibilgailu autonomoen testuinguruan sentsoreen bidez jasotako datu guztiak prozesatzea lan astuna izan daiteke, eta konexio- nahiz latentzia-arazoek prozesaketarako *edge devices* deiturikoak erabili beharra dakarte, hau da, informazioaren denbora errealeko lanketa ibilgailuan bertan egiteko beharra. Kapitulu honetan CAF Signalling-eko PER taldeak lan horretarako aukeratu zuen **Nvidia Jetson Xavier AGX** plaka aurkeztuko da, eta baita bere ezaugarririk behinenak azalduko ere: arkitektura orokorra, CPUa, GPUa, bestelako azeleragailuak... eta Nvidiak eskaintzen duen *software*-pila espezifikoa.

Esku arteko problemaren arabera, informazioaren kudeaketa modu ezberdinetan egin daiteke [64]. Latentzia-muga malgua duen informazioa prozesatzeko *cloud computing* erabili ohi da, datu-zentru zentralizatu ahaltsuetan informazio-sorta handiak prozesatuz (izan *datacenter* pribatuetan, izan hainbat konpainiak eskaintzen dituen zerbitzuetan). Emaitzak eskuratzeko denbora gutxiago dagoenean, ohikoa da konputazioa bezerotik gertuago kokatzea. Bigarren mota horretako sistemak sistema banatuak dira, eta paradigma hori muturrera eramatea da ***edge computing***, normalean denbora-errealeko atazei aplikatu ohi zaiena: datu-sarrerak fisikoki leku berean kokatutako konputagailuetan aztertzen dira (fabrika bateko makinen informazioa, ibilgailu bateko sentsoreen datuak...). Erabilpen kasu berean konputazio eredu ezberdinak konbina daitezke, azpi-problema batzuk sarean eta besteak ertzean ebatziz.

Nvidia etxearen Jetson familia *edge computing* paradigmarako diseinatutako plakek osatzen dute, bereziki adimen artifizialean oinarritutako produktuak garatu eta merkaturatzeko [65]. Katalogoko gailuetatik Xavier AGX da ahaltsuena, eta modelo hori da esku



6.1 Irudia: Xavier AGX plakaren egitura orokorra

arteko ikusmen artifizialeko azpi-sistemaren garuna.

6.1 Arkitektura

Xavier AGX plataforma SoM (*System on Module*) plaka bat da, modulu bakarrean prozesu-unitateak, memoria nagusia, biltegiratzea, S/I interfazeak eta energia-azpisistema elkartuz (6.1 irudia [66]). Modulua *Developer Kit* deituriko formatuan ere saltzen da (lan honetan hori erabili da), disipadore termikoarekin, beharrezko kableekin... erabilera errazteko.

Moduluaren bihotzean dagoen SoC (*System on Chip*) txipak sistemak erabiltzen dituen prozesu-unitateak biltzen ditu: ARMv8.2 arkitekturan oinarritutako Carmel **CPU**a; Nvidia etxearen Volta arkitekturan oinarritutako **GPU**a; NVDLA motako ***deep learning* azeleragailuak** (2); **ikusmen-azeleragailua**; **bideo kodifikatzailea** eta **deskodifikatzailea**... Osagaiak txip bakarrean elkartzeak beraien arteko komunikazioak azkartzen ditu; esate baterako, CPUak eta GPUak **memoria partekatua** erabiltzen dute.

Bestalde, Sarrera/Irteera azpisistema oso bat eskaintzen du funtzionalitateak hedatu ahal izateko. Ohiko periferikoen interfazeez gain (HDMI eta DP portuak, USB portuak, SD txartelen irakurgailua...) sistema txertatuetan ohikoak diren komunikazio-bideak ere baditu: UART, I2C eta CAN interfazeak; 40 GPIO *pin*... Memoriari dagokionez, proiektura-

ko erabilitako bertsioak 2133MHz-tara doan 32GB-eko LPDDR4x memoria nagusia du, 256biteko datu-busarekin eta 137GB/s-ko banda-zabalerarekin.

6.2 Carmel CPUa

Nvidiak ARMv8 arkitektura jarraituz eraiki duen 64 biteko zortzi nukleoko prozesagailu propioa darabil plakak: Carmel izeneko 12nm-ko bina *core*ko lau *cluster* ditu, eta nukleo bakoitza 2,26GHz-ko maiztasunera funtzionatzeko gai da (6.2a irudia) [66].

Ohikoa denez, memoria-atzipenen kostua murrizteko 3 mailako *cache*-hierarkia bat darabil prozesagailuak. *Core* bakoitzak L1 mailako bere *cache* memoria banatuak ditu, bana aginduentzat (128KB) eta datuentzat (64KB). *Cluster* bakoitzak, berriz, bere 2MB-ko L2 *cache* bateratua du. Azken maila, berriz, prozesagailu osoarentzat partekatua da: 4MB-ko *cache* memoria. Hierarkia osoa *set-associative* edo multzoka elkargarria da: 4-*way* L1, eta 16-*way* L2 eta L3.

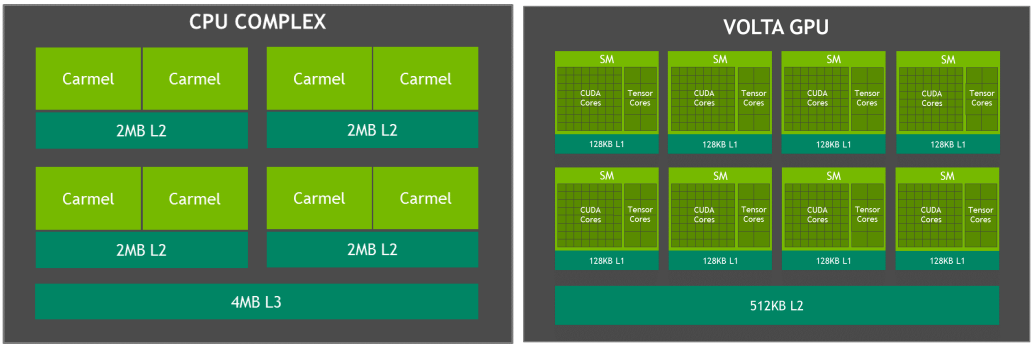
Nukleo guztiak *System Coherency Fabric* (SCF) deritzon geruza batek kontrolatzen ditu, *Heterogeneous Multi-Processing* (HMP) ingurune bat sortuz [67]: une bakoitzeko konputazio-beharraren arabera *core* batzuk edo besteak era koherentean piztu/itzaliko dira, energetikoki efizienteagoa den *multicore* exekuzio moldagarri bat lortuz [1] . Gainera, SCF arduratzen da nukleoen eta memoria-kontrolagailuaren (*Memory Controller Fabric*) arteko zubi-lana egiteaz, eta baita memorian mapeatutako S/I gailuekiko bitartekari izateaz ere (MMIO, *Memory Mapped I/O*).

6.2.1 ARMv8.2

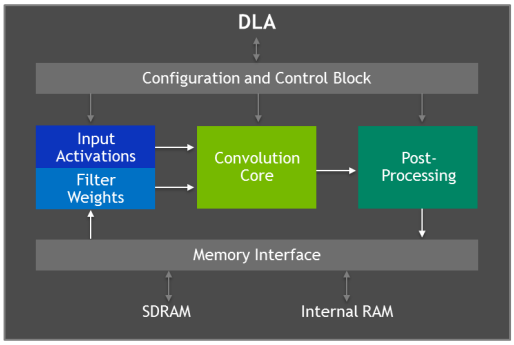
ARM mikroprozesagailuak diseinatzeko arkitektura abstraktu bat da, gailuek jarraitu beharreko gutxieneko espezifikazio funtzionalak ezartzen dituena [69]: *Instruction Set* (IS) edo agindu-multzoa, erregistroen antolaketa, salbuespenen/etenen kudeaketa, memoria-eredua...

Reduced Instruction Set Computers (RISC) motako arkitektura denez, eragiketa aritmetiko-logiko guztiak erregistroetan egiten dira, eta ez memoriako datuetan. Datuak memoriatik

[1] [68] artikuluan gailu mugikorretako HMP eredua azaltzen da: arkitektura ezberdinetako prozesagailuak uztartzen dira, une bakoitzeko energia- eta ahalmen-behar eta -mugei erantzuteko. Carmel CPUetan eredu bera jarraitzen dela uler daiteke, baina arkitektura bereko prozesagailuekin, heterogeneotasuna bakoitzean gaituta dagoen nukleo-kopuruaren arabera lortuz.



(a) CPUaren egitura **(b)** GPUaren egitura



(c) DLA azeleragailuen egitura

6.2 Irudia: Xavier SoC-aren osagai nagusien egiturak

irakurtzeko eta emaitzak memorian idazteko, *Load* eta *Store* motako aginduak ditu[2]; oinarrizkoenak: `LDR` (*Load Register*), `STR` (*Store Register*), `LDP` (*Load Pair*, aldi berean bi helbidetatik irakurtzeko) eta `STP` (*Store Pair*, aldi berean bi helbidetan idazteko).

Bertsioen arteko jauzirik handiena 7. eta 8.en artean eman zen, 2013. urtean, 64 biteko konputazio-eredua ere onartzen hasi baitzen arkitektura [71]. ARMv8-n AArch64 [3] exekuzio-modua agertu zen, agindu-multzo egokituarekin eta diseinuko aldaketekin[4].

6.2.2 Agindu-mailako paralelismoa

Nvidiak oso xehetasun gutxi publikatu ditu Carmel prozesagailuen diseinu zehatzari buruz; hala ere, komunitatean hedatua dago Denver aurrekariaren ezaugarririk esanguratsuenak presente daudela bertsio honetan ere [73].

[2] ARMv8 arkitekturaren dokumentazioko [70] kapituluan ikus daitezke `LD` eta `ST` motako agindu guztiak.
[3] 32bit-eko moduak AArch32 izena hartzen du.
[4] Ezaugarri batzuk: 32bit-eko luzera finkoko aginduak, 31 helburu orokorreko erregistro, AArch32-k baino baldintzapeko agindu gutxiago, SIMD paralelizazioaren soportea, 4 *byte*ko hitzak... [71, 72]

Dynamic Code Optimization edo DCO teknika 2015. urtean aurkeztu zuten Denver arkitekturaren berrikuntzetako bat da [74]. Diseinua 7-*way* supereskalarra dela esaten den arren, errealitatean ez da azpiegitura guztia ustiatzeko gai, eta *Instruction Level Parallelism* (ILP) maila oso mugatua du: ez du exekuzioan desordenik onartzen (*Out of Order*, OoO), deskodegailuak 2 μagindu soilik deskodetu ditzake zikloko... ARMv8 aginduak erabiliz egindako exekuzio-modu horrek markatzen du CPUaren oinarrizko errendimendua edo *baseline*a. Hortik aurrera, DCO *software*-azeleragailua erabiltzen saiatzen da, baliabideak xahutzeko. Kostu baxuko hari independente batean exekutatzera doan kodea aztertzen eta optimizatzen du, ohiko teknikak erabiliz: begiztak zabaltzea, erregistroak berrizendatzea... VLIW[5] motako agindu optimizatu berriak agindu-*cache*an gordetzen dira, ondorengo exekuzioetan erabili ahal izateko. Teknika honi esker ILP lor daiteke zirkuitu elektronikoak teknika tradizionaletan adina konplikatu gabe.

Carmel mikroprozesagailuei buruzko informazioan ere aipatzen da DCO teknika, 10-*way* zabalerako supereskalarra dela esateaz gain [67]; badirudi, beraz, kasu honetan ere *software* bidez lortzen dela agindu-mailako paralelismoa.

6.3 Volta GPUa

2017. urtean aurkeztu zen Volta arkitektura, Tesla V100 GPU modeloaren bidez [76]. Datu-zentroetan multzokatzeko sortutako modelo horrek hainbat berrikuntza ekarri zituen: ikasketa sakonerako optimizatutako *Streaming Multiprocessor* (SM) [6] egitura berritua, efizienteagoa; *Tensor Core* nukleo berrituak; L1 *cache* hobetua; harien planifikazio independentea[7]; *cluster*-etan elkartzeko NVLink teknologia [8]...

Xavier AGX plakak Volta arkitekturaren bertsio murriztu bat dauka, sistema txertatuaren tamaina-, konplexutasun- eta energia-mugak errespetatzeko. 6.2b irudian ikus daitekeenez, 8 SM ditu sistemak, bakoitza 64 ohiko CUDA nukleorekin, 8 *Tensor Core*-rekin eta

[5] *Very Long Instruction Word* agindu-mailako paralelismorako *software* bidezko teknika da, prozesagailua agindu primitiboekin elikatu ordez, agindu-sekuentziekin elikatzen duena. Sekuentzietan bildutako mikroaginduekin unitate-funtzionalak ahalik eta gehien erabiltzea bilatzen da. Teknika honek prozesagailu supereskalarrak eraikitzea sinplifikatzen du [75].

[6] SM bakoitzak 32 hari paraleloko *warp* deituriko taldeak sortzen eta kudeatzen ditu, *Single Instruction Multiple Threads* edo SIMT moduann (SIMD-ren antzekoa). Hari guztiak programaren puntu beretik hasten dira, eta adar ezberdinetatik joan daitezkeen arren, aldi bakoitzean adar bakarra exekuta daiteke. Ondorioz, errendimendurik onena hari guztiak bide beretik doazenean lortzen da. [51], 4.1 kapitulua.

[7] SIMT arkitektura jarraitu arren, *warp* bakoitzeko hariak era independentean planifikatu ditzake, ale xeheko konkurrentzia lortuz [76], *Volta SIMT Model* atala.

[8] Nvidiaren GPUak CPUekin edo beste GPUekin lotzeko eta *interconnection network* sareak osatzeko PCIe loturen teknologia alternatiboa da, Nvidiak 2016ko Pascal arkitekturarekin aurkeztua [77].

bere 128KB-ko L1 *cache*arekin; GPU osoak, berriz, 512KB-ko L2 *cache*a partekatzen du. SMak *Streaming Multiprocessor Partitions* edo SMP deituriko lau bloketan banatzen dira, erregistro-multzo eta L0 *cache* propioekin. Gailuak 1.37GHz-ko maiztasunarekin lan egin dezake, eta FP16 zehaztasunean, 11TFLOPS-eko konputazio-ahalmena du.

Aipagarria da Volta belaunaldian aurkeztu ziren *Tensor Core* nukleo berezien erabilera. "Matrizeen arteko biderketa eta metaketa" (*Matrix Multiply and Accumulate* edo MMA, *Multiply and ACcumulate* edo MAC) eragiketa egiteko bereziki diseinatutako nukleoak dira, kasu honetan erloju-ziklo bakoitzean 4×4 gelaxka prozesatzeko gai direnak. CUDAk, 10. bertsiotik aurrera, *warp* mailako MMA eragiketen APIa eskaintzen du [78]; bestalde, cuDNN eta TensorRT-k automatikoki erabiltzen dute teknologia hau Volta gailuetan.

6.4 DLA azeleragailuak

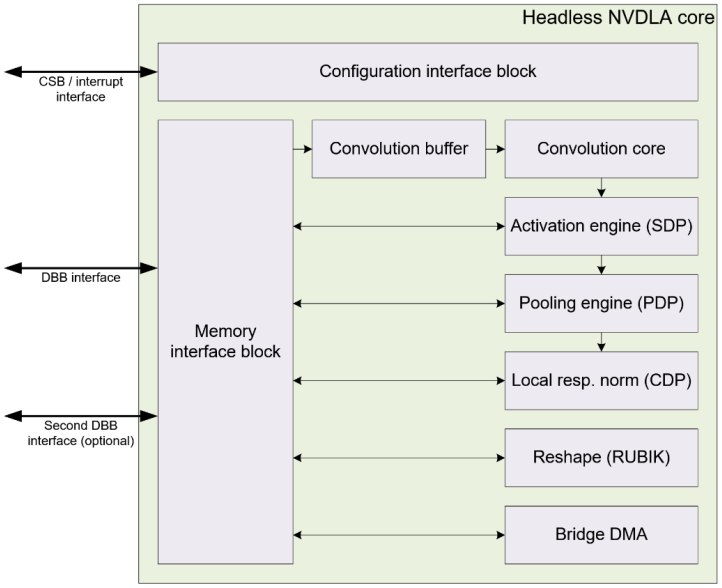
CNN sareen inferentzia-prozesua azeleratzeko bereziki eraikitako bi DLA (*Deep Learning Accelerator*) dauzka Xavier AGX plakak. Bateragarriak diren eragiketak bertan exekutatuz, GPUaren lan-karga txikitzea lor daiteke, bestelako kalkuluak (konplexuagoak, askotan) egin ahal izateko.

6.4.1 NVDLA arkitektura

NVDLA (Nvidia DLA) sare neuronalen inferentzia-azeleragailuak diseinatzeko arkitektura irekia da (*Open Source*), mota eta konplexutasun anitzetako sistemetan integratzeko aukera eskaintzen duena, diseinu modular eta konfiguragarriari esker [79] .

6.3 [80] irudian ageri da DLA nukleo baten barneko egitura, osagairik esanguratsuenekin eta euren arteko loturekin:

- **Eragiketa-motorrak:** CNN sareetan ohikoak diren kalkuluak egiteko bereziki diseinatutako zirkuituak. *Convolutional Core* nukleoa memoriatik irakurritako datuen gainean konboluzio-eragiketa egiteaz arduratzen da (*Convolutional Buffer* memorian pisuak, *bias*ak eta sarrerako datuak gordetzen ditu, memoria nagusira atzipenak gutxitzeko); *Activation Engine* edo *Single Data point Processor* (SDP) *bias*a gehitzeaz eta aktibazio-funtzioa(k) aplikatzeaz arduratzen da (*sigmoid*, *tanh*, *ReLU*, *LReLU*... onartzen ditu); *Pooling Engine* edo *Planar Data Processor* (PDP)



6.3 Irudia: NVDLA nukleoen barne-egitura

pooling eragiketak kalkulatzeko gai da (*maximum*, *average* edo *minimum*); eta *Local Reshape Normalization Engine* edo *Cross-channel Data Processor* osagaiak (CDP) kanal-dimentsioan (RGB, adibidez) lan egiten du, ohiko dimentsio espazialetan aritu ordez. Konponente horiek elkarren artean FIFO ilaren bidez ere elkartu daitezke *pipeline* edo hodi moduan, bloke baten emaitzak zuzenean hurrengoari pasatzeko, memoriatik igaro beharrik gabe, *fused* exekuzio-moduan. Osagaiek datu independenteetan lan egin behar dutenean, ostera, *indpendent* eran funtzionatzen dute, sarrerako datuek memoriatik irakurriz, eta emaitzak memorian idatziz.

- **Memoriako eragiketen blokeak:** memoriako datuetan aldaketak egiteko gai dira. *Data Reshape Engine* datuak formatuz aldatzeko diseinatu da, matrizeak iraultzeko edo kateatzeko, adibidez. *Bridge DMA*, berriz, memoria-interfaze bat baino gehiago daudenean haien arteko kopiak egiteko erabiltzen da[9].

- **Memoriaren kontrolagailua:** *Memory Interface Block* memoria nagusiarekiko komunikazioak kudeatzeaz arduratzen da.

- **Kanpoko interfazeak:** sistema osatzen duten gainerako gailuekiko komunikazioak egiteko loturak dira. *Configuration Space Bus* (CSB) 32 biteko kontrol-bus sinkronoa da, CPUak gailuko konfigurazio-erregistroak atzitzeko erabiltzen duena; *Inte-*

[9]Banda-zabalera altuagoa eta latentzia baxuagoa beharrezkoa den testuinguruetan, SRAM motako memoria bat txertatu daiteke sistemako DRAM memoria nagusiaz gain, *cache* moduan funtzionatzeko.

rrupt Interface bit bakarreko kanal asinkronoa CPUra eten-eskaerak bidaltzeko erabiltzen da, gertaerak notifikatzeko; eta, azkenik, *Data BackBone* (DBB) interfazea gailua memoria nagusiaren azpisitemarekin konektatzen duen datu-busa da[10].

Nukleoen arkitektura aurkeztearekin batera, NVDLA *core*ak sistema oso batean txertatzeko bi diseinu ere proposatzen dira: bertsio murriztuan (*Small NVDLA Model*) memoria bakarra sistemako DRAM memoria da, eta CPUa zuzenean arduratzen da gailua kontrolatzeaz; bertsio osoan (*Large NVDLA Model*), aldiz, SRAM memoria gehigarria txertatzen da sisteman, eta CPUarekiko zubi-lanak mikrokontrolagailu batek egiten ditu[11].

Bi proposamenen *hardware* diseinuaren HDL (*Hardware Description Language*) kodea [81] GitHub gordailuan aurkitu daiteke, Verilog lengoaian idatzita; horri esker, ASIC-ak eraikitzeaz gain, zuzenean FPGAtan erabil daiteke arkitektura. Horrekin batera, integraziorako beharrezko softwarea ere urki daiteke da (*driver*-ak, konpilaziorako tresnak...).

6.4.2 DLA motorrak

Plakak NVDLA arkitektura jarraitzen duten bi DLA motor dauzka. Arkitektura bera irekia izanagatik, ezin izan da inplementazio zehatzari buruzko informazio esanguratsurik topatu. Aurkezpen-artikuluan errendimenduari buruzko hainbat datu eskaintzen dira: motorretako bakoitza FP16 zehaztasunean 2.5TFLOPS exekutatzeko gai da, 0.5-1.5W arteko energia kontsumoarekin [66]. DLAk TensorRT liburutegiaren bidez atzitu behar dira, 5.0 bertsioan `IBuilder` interfazean txertatu ziren APIak erabiliz. *Software*- eta *hardware*-mugak muga, egon badaude osorik DLAtan exekutatu daitezkeen sare konboluzionalak (AlexNet, GoogleNet eta ResNet-50, esaterako); gainerakoetan, *GPU Fallback* modua aktiba daiteke, sostengurik gabeko geruzak GPUan exekutatzeko.

6.5 Energia-planak

Jetson familiako gailuak hainbat energia-plan edo -mailatan funtzionatzeko konfigura daitezke, testuinguruaren araberako beharretara egokitzeko. Konfigurazioak plakako osa-

[10] Bigarren interfazea balizko SRAM memoria gehigarria konektatu ahal izateko da.

[11] Koprozesagailua ale xeheko kudeaketaz eta planifikazioaz arduratzen da; ale larriko kudeaketa, S/I eragiketak eta sistemako gainerako osagaiekiko kudeaketa bezalako atazak, ordea, CPUaren esku gelditzen dira.

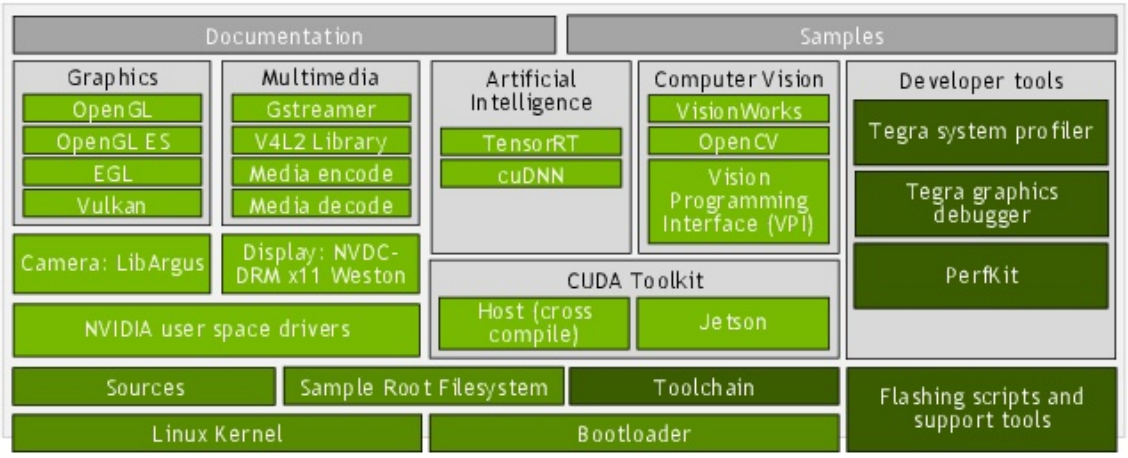
Energia-plana	MAXN	15W
Kontsumoa	30W	15W
CPU nukleoak	8	4
CPU maiztasuna	2265.6MHz	1200MHz
GPU Tensors/Core	4	2
GPU maiztasuna	1377MHz	670MHz
DLA nukleoak	2	2
DLA maiztasuna	1395.2MHz	750MHz
Memoriaren maiztasuna	2133MHz	1333MHz

6.1 Taula: MAXN eta 15W energia-planen alderaketa

gaien frekuentzia maximoan (CPUaren, GPUaren, DLA azeleragailuen, memoria nagusiaren... maiztasunak), CPUan gaitutako nukleo kopuruan eta GPUko nukleo bakoitzeko tentsore kopuruan du eragina, besteak beste. 6.1 taulan 15W energia-plan orekatua erabili da bi moduen arteko konparaketa egiteko. 15-30W tarteko zazpi energia-planetarako konfigura daiteke Xavier AGX plaka [82]; horien artetik, errendimendu maximoa eskaintzen duen MAXN modua erabili da proiektu honetan.

6.6 Software pila

Xavier AGX plataforma Nvidia Jetpack SDK (*Software Developer Kit*) tresna multzoarekin dator, Jetson familiako gailuak erabiliz Adimen Artifizialeko aplikazioak garatzeko beharrezko *software* pilarekin [83]. Tresna horien artean daude gailurako sistema-irudia, instalatzailea(k), konpilatzaile gurutzatua, adibideak eta dokumentazioa.



6.4 Irudia: L4T sistema-irudiaren osagaiak

Plakan instalatu beharreko sistema-irudia *Linux4Tegra* (L4T moduan ezaguna) da. 6.4 irudian L4T sistemaren osagaiak ikus daitezke ([84] L4T-ren gidatik hartua). Horien artetik, ondorengoak izan dira proiektuaren garapenari begira esanguratsuenak:

- **Sistema eragilea**: Linux 4.9 kernelaren gainean eraikitako sistema eragilea darabil, Ubuntun oinarritutako fitxategi-sistema eta erabiltzaile interfazearekin (`apt` pakete kudeatzailearen moduko tresnetara sarbidea ematen du horrek).

- `C/C++` **lengoaiak:** programazio-lengoaia horiekin lan egiteko beharrezko liburutegi eta tresnak dauzka (`gcc` konpiladorea, `glibc` liburutegi estandarra...).

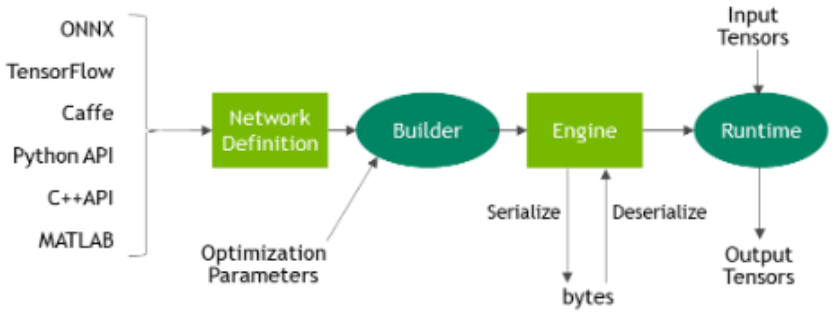
- **CUDA:** Nvidiaren GPUak programatzeko plataformaren [51] 10.2 bertsioa eta dagozkion liburutegiak dauzka instalatuta plakak, `C/C++` lengoaiekiko interfazea eta `nvcc` konpilatzailea barne.

- Adimen artifizialeko baliabideak:

  - cuDNN: Adimen Artifizialeko ohiko eragiketen GPUrako CUDA kernelak inplementatzen dituen liburutegia da cuDNN (*CUDA Deep Neural Networks library*) [85]. Ikasketa sakoneko aplikazioak programatzeko abstrakzio-maila bat gehiago eskaintzen du, sarri agertzen diren konboluzio-, *pooling*- eta *softmax*-eragiketen, ohiko aktibazio-funtzioen edo datu-formatuen bihurketen behemailako programazioa egin beharrik ez izateko.

  - **TensorRT:** *Tensor Real Time* sare neuronalen inferentzia-prozesua azkartzeko sortutako liburutegia da, cuDNN eta CUDA erabiliz [86]. Sareak eskuz sortzeko aukeraz gain (cuDNN baino are goi-mailakoagoa den interfaze baten bidez, eragiketa primitiboen ordez geruza osoak sortuz), ONNX formatuko modeloak zuzenean inportatzekoa ere eskaintzen du. Motorra sortzerakoan, ohiko fluxua sarea eraikitzea, optimizatzea eta bertsio berria serializatzea da (fitxategi bitar bihurtuz), behin bakarrik (bertsio bakoitzeko); inferentzia-garaian, berriz, nahikoa da motorra deserializatzearekin (ikus. [87] gidatik hartutako 6.5 irudia). Liburutegi honek DLA azeleragailuen eta GPUko *Tensor Cores* nukleoen erabilera gardena bihurtzen du erabiltzailearentzat. Onartzen ez dituen geruzak eskuz inplementatzeko aukera ere badago, *plugin* propioak sortzeko eskaintzen duen interfazearen bidez.

- **OpenCV:** *Open Computer Vision* konputagailu bidezko ikusmenerako baliabideak biltzen dituen liburutegi irekia da (iturburu-kodea [88] helbidean atzigarri dago).



6.5 Irudia: TensorRT liburutegia erabiltzeko lan-fluxua.

Ikusmen artifizialeko aplikazioak programatzen laguntzeko datu-motak, algoritmoak eta funtzionalitateak inplementatzen ditu, hainbat modulutan banatuta. Horien artean daude irudien prozesaketarako `imgproc`, irudien Sarrera/Irteerarako `imgcodecs`, sare neuronalak exekutatzeko `dnn` eta bideoa irakurri/idazteko `videoio`.

- **Tresna gehigarriak:** sistemaren errendimenduari buruzko informazio periodikoa (CPUaren eta GPUaren maiztasuna, memoriaren erabilera eta txipen tenperaturak, kasu) eskaintzen duen `tegrastats` komandoa erabili da sistemaren karga aztertzeko.

7. KAPITULUA

Soluzioaren garapena

Kapitulu honetan inferentzia azkartzeko proposatu den soluziora heltzeko emandako pausuen nondik norakoak azalduko dira. Hasteko, ebazpenaren atalik azpimarragarrienen xehetasunak azalduko dira; ondoren, inplementatu den Casio izeneko moduluaren egitura; eta, azkenik, bertsioen bilakaera ordena kronologikoan.

Ezer baino lehen, objektuen detekzioko aplikazioen azeleraziorako tekniketatik baztertu diren bideak aipatzea beharrezkoa da: *batching* edo multzokatzea ez da erabili, emaria handitzea lortzen duen arren latentzia ere handitzen baitu; eta ez da datuen *pipeline*ik sortu, exekuzio-urratsak tartekatuz, Polarisen zikloan zehar gainerako moduluentzat baliabideak askatu behar baitira.

7.1 Darknet-etik ONNX-era bihurketa

TensorRT liburutegiak sarea deskribatzen duten objektuak sortzeko bi interfaze nagusi eskaintzen ditu: eskuz egiteko, `IBuilder` interfazetatik eratorritako objektu eraikitzaile bat erabil daiteke, onartzen dituen geruzak, bakoitzaren konfigurazio-aukerak, elkarren arteko loturak (*buffer*-en erakusleen bidez) eta pisuak esleituz; edo ONNX [1] formatuko modelo bat inportatu daiteke, `nvonnxparser::IParser` interfazearekin[2].

[1] *Open Neural Network Exchange* edo ONNX Adimen Artifizialeko modeloak deskribatzeko *open source* ekosistema da, tresnen arteko bateragarritasuna bultzatzeko sortua [89].

[2] Aurreko bertsioetan sareak Caffe eta UFF formatuetatik ere inportatu zitezkeen arren, 7.0 bertsiotik aurrera *deprecated* moduan ageri dira bihurgailuak, eta 8.0 bertsioan ez dira egongo [90].

Optimizatu nahi den YOLOv4 sarea berau entrenatzeko erabilitako *Darknet framework*-eko formatu propioan dago: arkitektura `.cfg` fitxategi batean, eta pisuak (eta *bias*ak) `.weights` fitxategi batean. Bihurketarako [91] GitHub gordailuko `yolo_to_onnx.py` Python-eko *script*a erabili da.

Konfigurazio-fitxategitik geruzak banaka-banaka testu arruntetik objektu egituratu bihurtzen ditu, Python-eko hiztegiak erabiliz. Ondoren, `onnx.helper.make_node()` [92] funtzioaren argumentu moduan erabiltzen ditu egitura horiek, Darknet-eko geruzen ONNX-eko baliokideak sortzeko. Geruza guztiak ezin direnez zuzenean mapeatu, hainbat aldaketa aplikatzen dizkio modeloaren egiturari, baliokidetasun funtzionala mantentzen duen arren:

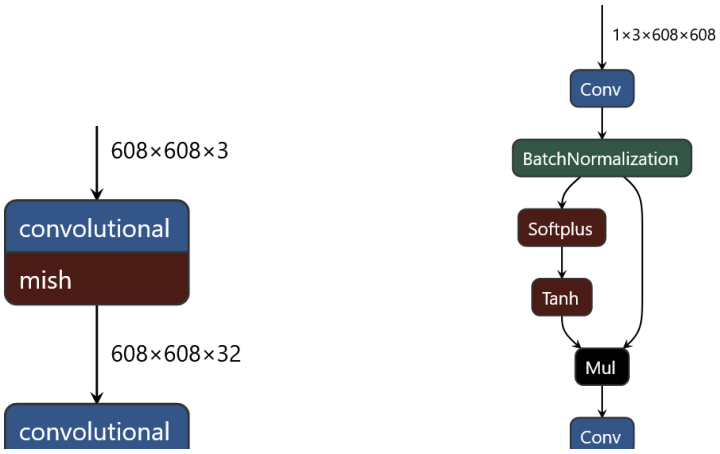
- Geruzak azpi-geruzetan banatzen ditu, ONNXeko eragiketa baliokideak erabili ahal izateko. Esaterako, *Convolution* motako kapetan konboluzio-eragiketa bera eta aktibazioa elkarren segidan dauden bi geruza independente bihurtuko dira. Horrez gain, hala dagokionean *Batch Normalization* eragiketa ere erauzten du.

- YOLOv4-n agertu den *mish(x)* aktibazio-funtzioa ez da ONNXen existitzen, baina definizioz $mish(x) = x \cdot tanh(softplus(x))$ moduan berridatz daiteke [38], eta osagai horiek badaude ONNXen. Beraz, *mish* motako aktibazio-funtzio bakoitza *tanh*, *softplus* eta biderketa funtzioetan deskonposatuko da.

- Sarearen irteerako hiru YOLO geruzek ez dute ONNXeko baliokiderik. Autoreak horren aurrean *dummy* (sic.) nodoak sortzea erabaki zuen, ezer egiten ez dutenak. Sarearen geruza kopurua-eta kontatzerakoan kontuan hartuko diren arren, programatzailearen esku egongo da azken bertsioan geruza horien funtzionalitatea inplementatzea (kasu honetan TensorRT-n) eta ordezkatzea.

Netron [93] tresna erabiliz erakutsi da sarearen lehen geruza konboluzionala 7.1 irudian. Ikus daitekeenez, Darknet formatuan kapa bakarra dena ONNX-era bihurtzerakoan bost geruza dira: konboluzio-eragiketa, *batch normalization*, bi aktibazio-funtzio eta biderketa, hain zuzen ere.

Sarearen egitura sortutakoan, pisu serializatuen `.weights` fitxategi bitarra irakurri eta interpretatzen du, eragiketa bakoitzari dagozkion pisu eta *bias*ak esleituz. Izan ere, ONNX fitxategian egitura eta parametroak elkarrekin gordetzen dira.



(a) Lehen geruza Darknet-en **(b)** Lehen geruza ONNXen

7.1 Irudia: YOLOv4 sareko lehen geruza konboluzionala Darknet eta ONNX formatuetan

7.2 ONNX-etik TensorRT-ra bihurketa

Aurreko pausuan sortutako ONNX fitxategia `nvonnxparser` liburutegiko funtzioak erabiliz irakur daiteke, TensorRT-k ulertuko duen formatuko (`INetworkDescription`) sareegitura lortzeko. Bihurgailutik lortutako objektuari, eskuz, irteerako eginbeharrez arduratuko diren hiru geruzak gehitu beharko zaizkie (gogoratu, ONNX-en ez direla existitzen mota horretako kapak).

7.2.1 YOLO irteerako geruza

TensorRT-k aukera ematen dio erabiltzaileari geruza-mota propioak definitzeko, [87] gidako 4. kapituluak azaltzen duen moduan, *plugin* gehigarriak erabiliz. `IPluginCreator` interfazetik eraikitzaile bat eratorri eta inplementatu behar da, bertako metodo birtualak berridatziz. Mota horretako objektuek aukera emango diote erabiltzaileari *plugin*ari buruzko informazioa eskuratzeko eta geruza konfiguratzeko; TensorRT-ri, berriz, geruza serializatu eta deserializatzeko. Eraikitzaileak `IPluginV2` interfazearekin bateragarria den geruza sortu eta itzuliko du, erabiltzaileak sarean txertatu ahal izateko. Kalkuluak exekutatzeko `forwardGPU()` eta `enqueue()` funtzioen barrutik norberak definitutako CUDA kernelak dei daitezke, konputazioa GPUan egiteko. TensorRT-ren berezko geruzen katalogoan ere ez direnez YOLO geruzak ageri, *plugin*en sistema hori erabili da irteerako geruzak inplementatzeko.

ONNX-erako bihurketan egin bezala, orain ere oinarritzat [91] gordailuko kodea hartu da.

Bertako inplementazioaren egitura berrerabili arren (klasearen konfigurazio-atributuak, serializazio-tamaina...), kalkuluez arduratzen den CUDA *kernel*-a hutsetik berridatzi da, YOLOv4 arkitekturaren berezko irteerarekin bateragarri bihurtzeko. Izan ere, ondorengo ezaugarriak zituen jatorrizko kodeak:

- Hari bakarra detekzio baten propietate guztiak prozesatzeaz arduratzen zen; hots, $5+C$ kalkulu egiteaz, C klase-kopurua izanik.
- Sarearen berezko irteeran detekzio bakoitzari ondorengo formatua dagokio [34]: $\{x, y, w, h, p_{obj}, p_0, ..., p_{c-1}\}$. Adibide honek, berriz, *kernel*aren barruan konputatzen ditu probabilitate maximoak eta kategoria-zenbakiak, $\{x, y, w, h, p_{obj}, C_{pmax}, p_{max}\}$ 7 balioko tentsoreak sortuz, C_{pmax} probabilitate maximoko klasearen zenbakia izanik.

Inplementatutako bertsioak irteerako formatua errespetatzen du, eta GPUko hari bakoitza prozesaketa bakarraz arduratzen da, paralelismo-maila handiagoa lortu asmoz.

Gogoratu, YOLOv4 sareko irteerako hiru geruzetako bakoitzak tamaina ezberdineko sareta bat inplementatzen duela, gelaxka bakoitzean a aingura-kopurua adina detekzio egiteko. Sarearen sarrerako bereizmena $W \times H$ bada, i geruzetako bakoitzak k_i faktorea erabiliko du saretaren tamaina kalkulatzeko: $w'_i = w/k_i$ eta $h'_i = h/k_i$. Beraz, geruza bakoitzaren irteerako detekzio-kopurua $d_i = w'_i \cdot h'_i \cdot a$ izango da; prozesatu beharreko elementu-kopurua, aldiz, $n_i = d_i \cdot (5+C)$. Proiektu honetan $(W,H) = (608,608)$ eta $k = (8,16,32)$ denez, irteerako hiru geruzek 17328, 4332 eta 570 detekzio antzemango dituzte *frame* bakoitzeko.

CUDAko *kernel*a deitzerakoan dimentsio bakarreko tamainak erabili dira, sarrera/irteerako *buffer*ak ere halakoak direlako, nahiz eta maila logikoan bestelako interpretazio bat eman: $n_i/k_i = w'_i \cdot a \cdot (5+C)$ bloke eta $32 \cdot \lceil h'_i/32 \rceil$ hari. 32 da Volta GPUko *warp* tamaina (planifikazio-unitaterik txikiena), eta arkitekturaren atalean azaldu bezala, *warp* barruko aginduen dibergentziak errendimendu-galera suposa dezake ([76] 4.1 atala eta [94] *Independent Thread Scheduling* atala). Hasierako asmoa bloke bakoitza sarrerako gelaxka batez eta hari bakoitza sarrerako propietate batez arduratzea zen arren, exekutatu beharreko aginduak propietate bakoitzarentzat espezifikoak dira. *Warp* bakoitzkeo hari guztiek exekuzio-fluxu bera izateko, geruzako zutabe/propietate bikote bakoitzeko bloke bat definitu da, hariak errenkadez arduratuko direlarik.

7.2a irudian azken YOLO geruzaren sarrerako datuei aplikatu zaien hiru dimentsioko interpretazio logikoa ikus daiteke. `blockIdx.x` eta `threadIdx.x` balioetatik ondorioz-

tatuz hasieratu dira `my_anchor`, `my_detection` eta `my_column` aldagaiak: aingura edo sarrerako taula, detekzioa edo sarrerako zutabea eta propietatea edo sarrerako errenkada, hurrenez hurren. 7.2b irudian ikusten den irteerako antolaketan, berriz, taula, errenkada eta zutabea adierazten dituzte. Planteamendu logiko horiek jarraituz kalkulatzen ditu hari bakoitzak dagokon sarrerako eta irteerako posizioak.

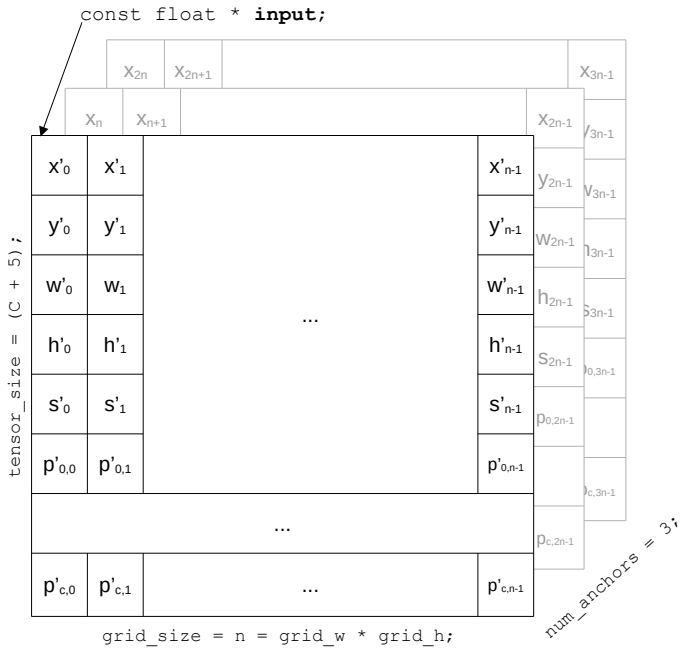
Helbideraketak definitutakoan, hari bakoitzak `*my_output = f(*my_input);` formako agindu bat exekutatuko du, kasu bakoitzean dagokion `f` funtzioa aplikatuz. Era matematikoan, A sarrera eta Z irteeraren arteko erlazioa honela definituko litzateke: $Z = f(A^\top)$. Probabilitateen kasuan $z = \sigma(a)$ da zuzenean; x eta y posizioetan saretako gelaxkaren koordenatuak eta hiperparametro moduan definitutako eskala erabiltzen dira *sigmoid* aplikatuz lortutako aktibazioa desplazatzeko; azkenik, altuera eta zabaleran, funtzio esponentziala, aingurak eta irudiaren tamaina.

7.2.2 Aurre-prozesaketarako geruza

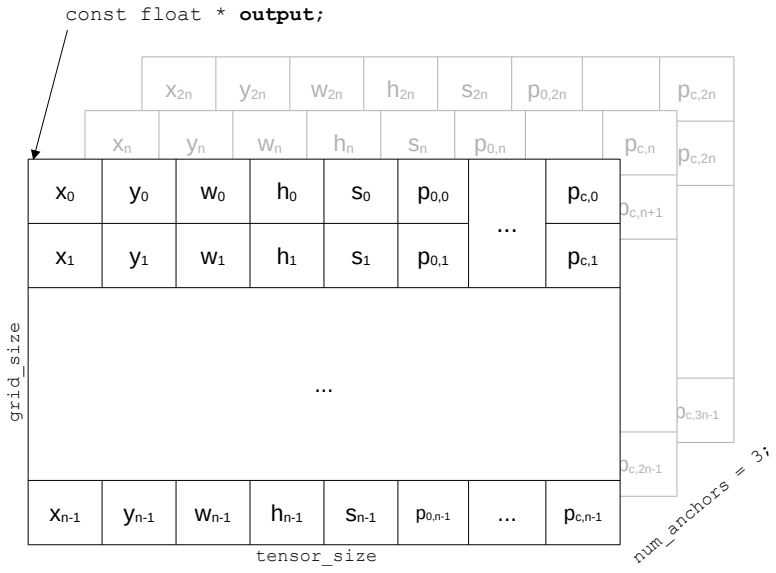
TensorRT-n sortutako sareak ondorengo aurre-prozesaketak eskatzen ditu (prozesua 7.3 irudian laburtu da):

- Irudia jatorrizko bereizmenetik sareak eskatzen duenera eskalatu behar da.
- Irudiko pixelen koloreen balioak BGR (*Blue*, *Green*, *Red*) ordenan jasotakoan, RGB-ra bihurtu behar dira.
- `cv::Mat` motako irudietan pixelen balioak NHWC formatuan daude; hau da, multzoko irudi bakoitzeko pixel bakoitzeko R, G eta B koloreen balioak elkarren segidan gordetzen dira. TensorRT-n, berriz, sareak NCHW formatua eskatzen du sarreran, hiru koloreetako bakoitzaren pixel guztien balioak elkarren segidan gordetzea.
- $\{0,\ldots,255\}$ multzoko balio osoak normalizatu egin behar dira, denak sareak eskatzen dituen $[0,1]$ tartean egoteko.

Horren aurrean, CPUko ohiko kode sekuentziala erabili ordez, aurreko puntuko bide bera jarraitu eta *plugin* pertsonalizatuak inplementatzea erabaki da; azken batean, $3 \times W \times X$ datu prozesatu behar baitira irudi bakoitzeko (horrek, erabili den 608×608 modeloarentzat, 1.110.816 balio esan nahi du).

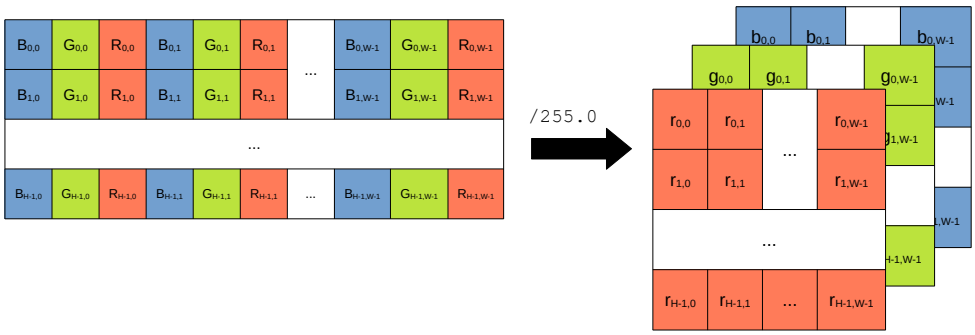


(a) YOLO geruzaren sarrerako datuak



(b) YOLO geruzaren irteera

7.2 Irudia: YOLO geruzen sarrera- eta irteera-datuen antolaketa eta indexazioa



7.3 Irudia: Sarearen sarrerako pixelen datuen bihurketa

Kasu honetan hutsetik sortu da `IPluginV2IOExt` motako *plugin*aren klasea eta bere eraikitzailea, bere eremu eta metodo guztiekin, beste inplementazioaren eta TensoRT-ko gidako informazioaren laguntzaz.

CUDAko *kernel*ean hari bakoitza sarrerako datu bat bihurtzeaz arduratzen da: normalizatzeaz eta dagokion memoriako posizioan idazteaz. Indizeak antolatzerakoan, R eta B kanalen alderantzizkatzea ere kontuan hartu da.

Salbuespen gisa, irudiaren tamaina egokitzea CPUan egingo da bertsio honetan ere, eskuz inplementatu ordez OpenCV-ko `cv::Resize()` funtzioa erabiliz, `cv::INTER_LINEAR` (*Linear Interpolation*) algoritmoarekin[3].

7.3 Konfigurazioak

Sarea sortzeaz gain, beharrezkoa da berau optimizatzeaz arduratuko den `IBuilder` eraikitzailea konfiguratzea, `IBuilderConfig` klaseko objektu batean aukerak adieraziz. Exekuziorako DLA gailuetako bat edo GPUa (lehenetsia) erabili aukeratu daiteke geruza bakoitzarentzat; edo sare osoarentzat, `setDefaultDeviceType(DeviceType deviceType)` funtzioa erabiliz. DLAk aukeratu badira, bi motorretatik zein erabili ere espezifika daiteke, `setDLACore(int core)` funtzioarekin.

[3]OpenCV-k hainbat algoritmo eskaintzen ditu, baina oraingo bertsioan erabiltzen den bera aukeratu da, emaitzak konparatzeko.

Bestelako aukerak `setFlag(BuilderFlag flag)` funtzioaren bidez definitu daitezke, bakoitzarentzat *flag* bat aktibatuz. Eskuragarri daudenetatik, ondorengo adierazleak erabili dira: `kFP16` edo 0 bita, FP16 zehaztasuna erabiltzeko; `kGPU_FALLBACK` edo 3 bita, DLA batean exekutatzeko konfiguratu arren DLAk onartzen ez dituen geruzak GPUan exekutatzeko (bestela egoera horretan optimizazio-prozesua bertan behera utziko du); eta `kDISABLE_TIMING_CACHE` edo 6 bita, geruza berdinetan denbora neurketak ez berrerabiltzeko.

Beste aukera garrantzitsu bat `setWorkspaceSize()` da, optimizazioan erabil dezakeen memoria maximoa definitzen baitu.

Konfigurazio-etapa honetako aginduetan bereizten dira, programatzailearen ikuspegitik, sarea GPUrako edo DLAtarako optimizatzeko prozesuak.

7.4 Optimizazioak eta serializazioa

Sarea sortu eta konfiguratutakoan (bi pausuen ordena alderantzizkagarria da), `IBuilder` motako objektu baten `buildEngineWithConfig()` funtzioa deituz optimizazio-prozesua hasten da.

Eraikitako bertsio optimizatua fitxategi batean gordetzeko aukera ere badago, inferentziagaraian zuzenean bertatik irakurtzeko, optimizazio-prozesu osoa berregin beharrik gabe. Liburutegian *serialization* izenez ezagutzen da pausua, eta sareak (baita bere *plugin* guztiek ere) horretarako beharrezko metodoak inplementatzen dituzte. Sarearen `serialize()` deian pausua zentralizatu daiteke, memoriako *buffer* bat lortuz, bera arduratuko baita sarearen informazioa eta geruzak serializatuko dituzten funtzioak deitzeaz. Ondoren, programatzailearen esku dago datuak fitxategi batean idaztea.

7.5 Exekuziorako testuingurua

Inferentzia egiterakoan aurrez gordetako motor optimizatu bat deserializatu daiteke, zuzenean erabiltzeko. Motor bakarretik hainbat exekuzioko testuinguru independente sor daitezke, modeloaren hainbat instantzia aldi berean kargatuta edukitzeko, memorian pisuen kopia bakarrarekin (`IExecutionContext`). Eraikitako objektu horietako bakoitzak bere sarrera/irteerako *buffer*-ak beharko ditu: sarrerako geruzari dagokion bat, eta irteerako geruza bakoitzeko beste bat (tamaina guztiak aurrez ezagunak eta finkoak dira).

CUDA aplikazioak Tegra eredua jarraitzen duten gailuetara eramateko [95] orriko aholkuak jarraituz, memoria-eredu ezberdinak probatu dira. Emaitzarik onenak *Unified Memory* edo memoria bateratuak eman ditu; logikoa, bi gailuen memoria fisiko bera partekatzen dutela kontuan izanik. Memoria hau `cudaMallocManaged()` funtzioarekin erreserbatzen da, eta CPUak eta GPUak, biek *cache*atzen dute, atzitzeko memoria-transferentzia espliziturik gabe. Horren ordez *prefetching* modu efizientean egiteko pistak eman dakizkioke programari, memoriako espazio bat gailu bakoitzean noiz erabiliko den adierazteko (funtzio asinkronoa da).

7.6 Casio azpi-modulua

Azken inplementazioa **Casio**[4] izeneko azpi-moduluan bildu da, problema orokorrago bati emandako soluzio orokorrago bat izan asmo baitu, ez soilik YOLOv4 esku arteko Xavier AGX modelo jakin honetara optimizatuko duen kode zurruna.

Casio moduluak interfaze bateratu bat eskaintzen du sareak zuzenean OpenCV-rekin edo TensorRT-rekin exekutatzeko. Diseinatzerakoan klase abstraktu eta konkretuen hierarkia ireki bat erabili nahi izan da, etorkizunean modelo eta *backend* berriak gehitu ahal izateko: modelo bakoitzarentzat, *backend* bakoitzarentzat edo bien konbinazio bakoitzarentzat espezifikoak diren pausuak bereizi dira, modulua hedatzea errazteko asmoarekin. Funtzionalitateen inplementazioak ez ezik, konpilazioko *Makefile*ak[5] eta oinarrizko exekutagarriak[6] ere bildu dira.

Ondorengo zerrendan inferentzia-garaian zeresanik handiena ematen duten klaseak eta euren azalpen laburrak bildu dira, moduluaren egituraren erakusle moduan.

- `INetworkDescription` klase abstraktuak modelo bakoitzarentzat espezifikoak diren ezaugarriak eta funtzioak zein diren definitzen du. Eraikitzaile babestua inplementatzen du, eremu konstanteak hasieratzeko (sarearen sarrera/irteerako geruzak, haien tamainak, zabalera eta altuera, kategoria kopurua...). Hemendik eratorritako klaseek eremuak hasieratu eta funtzio espezifikoak inplementatu beharko dituzte. Nahitaez inplementatu beharreko bakarra sarearen irteerako *buffer*-etako informa-

[4] Irudimenari bide emanez, *CAf Signalling Inference Optimizer* izenaren akronimo moduan uler daiteke.

[5] CAF Signalling-eko egitura jarraitu behar izan da, goi-mailako moduluen konpilaziotik era gardenean deituta liburutegiak sortzeko.

[6] Oinarrizko programa nagusiak; oraingoz, YOLOv4 ONNXtik TensorRT-ra bihurtzeko `CasioONNXParser`, eta inferentzia egin eta emaitzak irteera estandarrean erakusteko, `CasioTest`.

zioa prozesatuko duena da (klase egokia aukeratu, NMS aplikatu, detekzioen objektuak hasieratu, koordenatuen eskala aldatu parametro moduan jasotako tamainarekin...). TensorRT-ko sare-definizio bat emanik, hari beharrezko *plugin*ak gehituko dizkion funtzio birtual bat ere badago (hautazkoa da gainidaztea, portaera lehenetsia NOP da).

  - `CNetworkDescriptionYolo4` klaseak YOLOv4 sarearen egitura estandarra errepresentatzen du, arkitekturan finkoak diren parametroak eremu konstante moduan eta konfiguragarriak aldakor moduan gordez.

- `EBackends` enumeratuan eskura dauden exekuzio-plataformak zerrendatzen dira; gaur gaurkoz, OpenCV eta TensorRT. `EModels`-en, berriz, moduluan inplementatu diren modeloak. Interfazea errazteko, *Factory* patroia jarraitu da `EModels` modelo bat, `w` sarrerako irudiaren zabalera, `h` altuera eta `c` kategoria-kopurua emanik saredeskribapeneko objektuak sortzeko.

- `IInferencer`: Polaris-ekin definitutako interfazea funtzio guztiz birtual moduan zehazten du goiburukoan.

  - `CInferencerTRT`: `IExecutionContext` objektu bat dauka klaseko atributu bezala, eta testuinguru hori erabiltzen du inferentzia inplementatzeko. Klase hau arduratzen da irudiaren tamaina doitzeaz eta *buffer*-en arteko memoria-eragiketez. `CModelTRT` objektu batean sarearen deskribapena eta dagoeneko irakurritako `ICudaEngine` motorra jasotzen ditu.
  - `CInferencerOCV`: aurrez Polarisen erabiltzen zen OpenCV-ko `cv::dnn::Net` sare-objektua da klaseko eremu bat, eta bera erabiltzen du detekzioak egiteko, `forward()` funtzioaren bidez. Kasu honetan, `cv::dnn::blobFromImage()` funtzioa erabiltzen da aurreprozesaketa guztia egiteko. `CModelOCV` motako modelo bat jasotzen du, sarearen mota eta fitxategiarekin (bi fitxategi, Darknet-en kasuan).

Modeloa optimizatzeko, berriz, `c_onnx_parser.h` eta `.cpp` fitxategietako baliabideak erabil daitezke. `SConfig` egituran TensorRT-ren egituran konfigurazio-aukerak bildu dira, interfaze sinplifikatuago eta intuitiboago bat eskaintzeko. Bihurketa, berriz, ondorengo funtzioan egiten da:

```
void parse_onnx(const char *onnx_path, const char *trt_path,
                INetworkDescription &descr, SConfig config);
```

Hemen azaldutakoez gain, bestelako datu-egiturak, funtzio-laguntzaileak eta klaseak ere sortu dira programa osatzerakoan. Aipatu, halaber, errore-tratamendua inplementatzeko `C++` liburutegi estandarreko `std::exception` klasetik eratorritako salbuespenak erabili direla: eskatutako fitxategia existitzen ez denean edo ezin izan denean irakurri, argumentu okerrak erabili direnean... Hala ere, eraikitakoa oraindik prototipoa da, eta ez da modu sakonean probatu - oraindik ezin da sistema sendoa denik esan.

7.7 Tarteko bertsioak

Proiektuaren hasieratik azken bertsiora bitartean bilakera nabarmena izan du inplementatutakoak bateragarritasunari, sendotasunari, eta erabilgarritasunari dagokienez. Eboluzio horretako mugarririk garrantzitsuenak eta bakoitzera heltzeko emandako pausuak hauek izan dira, modu sinplifikatuan azalduta:

1. Ikerketa-fasean ikasi bezala, TensorRT-n YOLOv4 optimizatu ahal izateko ONNX-eko bertsio bat eskura izatea beharrezkoa zela ikusi zen. Arazoa hasieratik ezagutuz gero saihesbideak egon badauden arren[7], dagoeneko Darknet-en entrenatutako sarea bihurtzeko [96] biltegiko kodea erabili zen, *plugin* espezifikorik gabeko bihurketa egiteko gai baita.

2. [86] TensorRT-ren gidatik, [91, 97] adibideetatik eta bestelako erreferentzietatik ateratako informazioarekin, eta aurreko pausuan lortutako modelo bateragarriarekin, lehen prototipoa osatu zen, optimizazio-prozesuko osagairik garrantzitsuenen inplementazioekin. Pausu honetan Nvidia driverrekin, prozesuan parte hartzen duten TensorRT objektu ugariekin, liburutegiak eskaintzen dituen aukerekin, memoriaren edo *buffer*-en kudeaketarekin, konpilazioarekin, dependentziekin eta sarearen sarrera/irteerako datuen interpretazioarekin izandako arazoak analizatu eta konpondu behar izan ziren.

3. Inferentziaren irteeran emaitza koherenteak lortutakoan, aurreko pausuan inplementatutakoa Casio azpi-moduluan egituratu eta bildu zen, klase-hierarkia baten baitan. Oraindik ez zen polimorfismoa eta herentzia ustiatu; aitzitik, prototipoko funtzionalitate berak biltzen zituzten klase konkretuak erabili ziren zuzenean, bihurketarako eta inferentziaren probarako programa nagusi banarekin.

[7] ONNX-en kode-gordailuan bertan YOLOv4-ren bertsio bat ageri da, TensorFlow-rako inplementaziotik eratorria.

4. Saiakera labur bat egin zen moduluaren inplementazioan CAF taldeko segurtasun- eta integritate-liburutegiak integratzeko (*Caesar* eta *RosaE* izena dute), PER taldean orokorrean bide hori jarraitzen hastea adostu baitzen. Hala ere, liburutegien eta proiektuaren ezaugarri bateraezinak eta testuinguruak (I+G+B proiektua izatea, aldatzeko erraza izatea...) eraginda asmoa baztertzea erabaki zen, enpresako arduradunarekin eta proiektuaren zuzendariarekin hala adostu ostean. Salbuespena *Caesar*-eko konpilazio-ereduak izan dira, gainerako sistemekin bateragarria izateko txantiloi hori jarraitu baita.

5. Aurreko puntuetako lanak prozesuak, osagaiak eta elkarren arteko menpekotasunak identifikatzeko balio izan zuen, etorkizunean modelo eta *backend* berrietara hedatzeko aukera emango zuen diseinu bat proposatzekoa. Klase abstraktuen eta konkretuen hierarkia bat definitu zen, Casio modulua errefaktorizatuz.

6. Lehen emaitzetan hobekuntza nabaritu zen arren, erabilitako ONNX bertsioaren irteera ez zen YOLOv4 arkitekturaren berezkoarekin bateragarria; gainera, *plugin*en gabeziari geruza asko era konplexuan konbinatuz egiten zion aurre. Horrek, errorea metatzeko arriskuaz aparte, arazketa-lanak zailtzea ekarri zuen. Sarea fintzeak adimen artifizialaren ezagutza sakonegia eskatuko zukeenez, beste erreferentziak bilatzea hobetsi zen. CAF taldeko zein taldetik kanpoko ikertzaileekin izandako hartu-emanetik egokiena aurrez aipatutako [91] helbideko *script*a erabiltzea zela erabaki zen. Modeloa erabili ahal izateko, YOLO geruzen TensorRT *plugin*ak eta euren CUDA *kernel*ak inplementatu ziren.

7. Saretik kanpoko datuen prozesaketa azkartzeko bideak jorratu ziren jarraian. Horien artean, CUDAko memoria bateratua erabiltzea eta aurreprozesaketa geruza pertsonalizatu batean inplementatzea.

8. Azkenik, Polaris moduluan integratu eta lehen probak egin dira, aurrez sendotasuna handitzeko oinarrizko errore-kudeaketa inplementatu ostean.

8. KAPITULUA

Emaitzen analisia

Kapitulu honen asmoa proiektuaren amaieran proposatutako soluzioaren errendimendua neurtu eta analizatzea da. Horretarako, Darknet-eko YOLOv4 sare batetik eta bere ONNX baliokidetik abiatuta, ondoko ikuspegietatik neurtuko da proiektuan lortutakoa: TensorRT-ren bihurketaren interpretazioa, GPUrako eta DLAtarako, lortutako modelo optimizatuei buruzko informazioarekin; modelo optimizatuen irteerako detekzioen zehaztasunaren baliozkotzea; eta azken bertsioaren errendimenduaren (abiaduraren) ebaluazioa.

8.1 Inferentzia motorren eraikuntza

TensorRT liburutegi itxia denez, zaila da sareari aplikatzen dizkion optimizazioak zehazki ezagutzea. Hala ere, aldaketetako batzuei buruzko informazioa badago [98]: erabiltzen ez diren geruzak ezabatzen ditu; ahal duenean, "fusio bertikala" egiten du (adibidez, eragiketa berari dagozkion konboluzioa, *pooling*a eta aktibazioa kapa bakarrean elkartzen ditu); "fusio horizontalean" antzeko eragiketak egiten dituzten geruza txikiagoak handi bakarrean elkartzen ditu... Ondoren, grafo berriko geruza bakoitzarekin hainbat *tactic* edo algoritmo probatzen ditu, geruzaz geruza uneko konfiguraziorako azkarrena dena hautatuz.

8.1.1 DLAtarako optimizazioa

Prozesua DLAtarako konfiguratuta aplikatzean ez dira esperotako emaitzak lortu. Gaur gaurkoz, DLAtan ez dira sareak darabiltzan *Softplus* eta *LeakyRelu* aktibazioak onartzen (ezta *plugin*ak ere) [99], eta geruza horiek GPUan exekutatu beharko dira. Arazoa are larriago bihurtzen da TensorRT-ren bigarren muga batengatik: gehienez 8 azpi-grafo ezberdin exekutatu daitezke DLA batean. *Subgraph* terminoaren definiziorik eskaintzen ez duen arren, testuingurutik uler daiteke hasieran sare osoa azpi-grafo moduan interpretatzen duela, eta gailuen arteko aldaketa bat egin behar den aldiro, azpi-grafo bat erauzten, helburuko gailurako. Arazo horiengatik, ia sare osoa GPUan exekutatzen zen bertsio honetan, GPUaren erabileraren murrizketarako aukerak desagertuz; gainera, egindako lehen probetan oso errendimendu txarra lortu zuen, latentzia handiarekin. Irteerako fitxategiaren analisi sakonagoa eta lehen emaitzak eskutan, **DLAtarako bertsioa alboratzea erabaki da**, gainerako azterketetatik kanpo utziz.

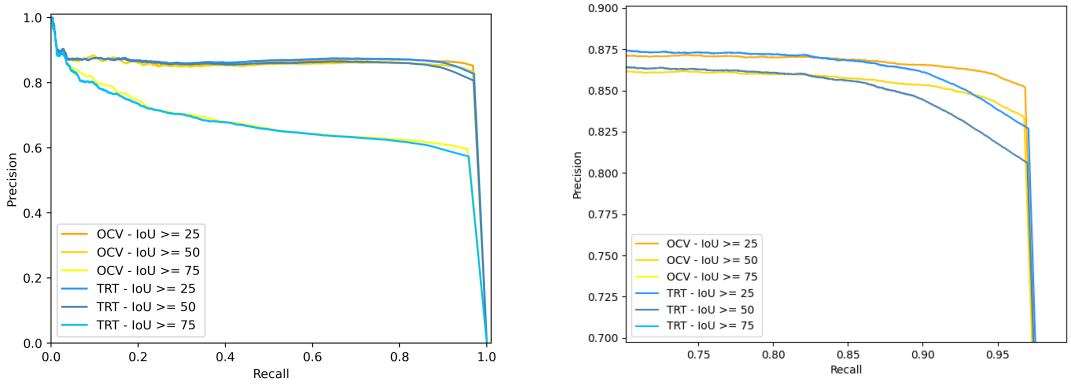
8.1.2 GPUrako optimizazioa

ONNX modeloa GPUrako konfigurazioarekin optimizatzerakoan lortutako irteerako fitxategiko unerik esanguratsuenak 8.1 taulan laburtu dira, horien artean aurrez aipatutako eraldaketak. Ikusten da sinplifikazio handi bat egiteko gai dela, azkenean geruza-kopurua erdira murrizten baita. Grafo berria sortutakoan, kapa bakoitzarentzat hainbat taktika probatzen ditu [1]. GPUrako bertsioa izango da azken proposamena, eta hemendik aurrerako emaitzen analisien oinarria.

Pausua	Geruza kopurua
`Original`	510
`After dead layer removal`	510
`After vertical fusions`	236
`After tensor merging`	231
`After concat removal`	224

8.1 Taula: Sarearen optimizazioen bilakaera (GPU)

[1] Modu opakoan tratatzen ditu erabilitako algoritmoak, TensorRT liburutegi itxia den heinean ez baitu horiei buruz identifikadore bat eta exekuzio denbora ez beste informazio lagungarririk ematen.



(a) Kurba osoak **(b)** Eskualde nagusia

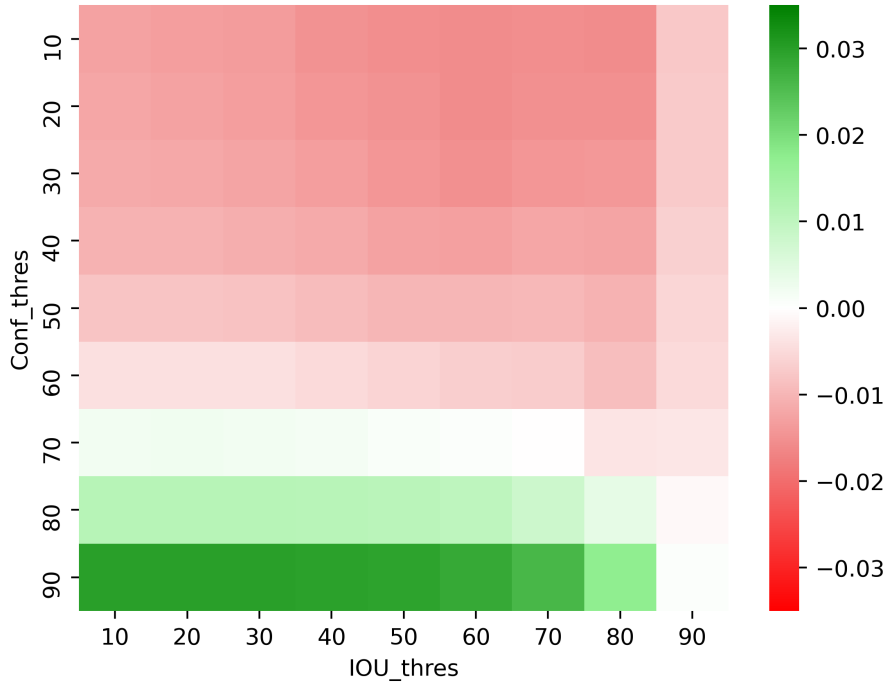
8.1 Irudia: *Precision-Recall* kurbak

8.2 Detekzioen zehaztasunaren baliozkotzea

Egindako optimizaziokin lortutako sarea ez da jatorrizkoaren berdina izango, eta kalkuluak egiteko erabilitako tresnak ere aldatu egin dira. Ondorioz, beharrezkoa da proposamen berriaren zehaztasuna edo kalitatea egokia dela ziurtatzea, trenean erabili ahal izateko. Hala ere, kasu honetan helburua ez da erabilitako modeloaren kalitatea neurtzea, bertsioen arteko diferentzia baizik; beraz, ez da beharrezkotzat jo objektuen detekzioan ohikoak diren metrika aurreratu eta estandarizatuak [100] kalkulatzea. Aitzitik, B eranskineko 4. zatian azaldutako *Precision* (zehaztasuna) eta *Recall* (estaldura), eta horietatik eratorritako F_1 balioa eta *PR* kurbak erabili dira, 5000 irudiz osaturiko datu-base batentzat kalkulatuta.

8.1a irudian IoU (B eranskineko B.2 ekuazioa) atalase konkretu batzuen *Precision-Recall* kurbak ikus daitezke. Oro har, *threshold* bakoitzari dagozkion TensorRT eta OpenCV kurbek oso antzeko forma dute. Dibergentziarik handieneko zonaldea 8.1b irudian erakutsi da, tamaina handiagoan. Bertan, bistakoa da jatorrizko sarearen lerroak gehiago (gehixeago) gerturatzen direla optimora ($(P,R) = (1.0,1.0)$).

8.2 irudian bertsio optimizatuak ($F_1(TRT)$) eta jatorrizko sareak ($F_1(OCV)$) lortzen dituzten F_1 balioen arteko aldea irudikatu da ($F_1 = F_1(TRT) - F_1(OCV)$). Oraingoan, gainditu beharreko konfidantza altua denean behintzat, bertsio optimizatuak emaitza hobeak ematen ditu. Dena dela, aldeen eragina zalantzan jar daiteke diferentzien tamaina txikiarengatik. Gainera, errealitatean Polaris moduluan aplikatu ohi den $P_{obj} \geq 0.7$ eta $IOU \geq 0.5$ muga-bikotearentzat ia emaitza bera lortzen dute bi modeloek.



8.2 Irudia: F_1 balioaren aldea atalase ezberdinentzat

Zenbakizko neurketez aparte, bideo bidezko analisi kualitatibo bat ere egin da, agertoki erreal batean modeloen erantzuna ikusteko. Bideo berari bi modeloak aplikatuz, irteeretan begi hutsez ez da alde esanguratsurik antzeman.

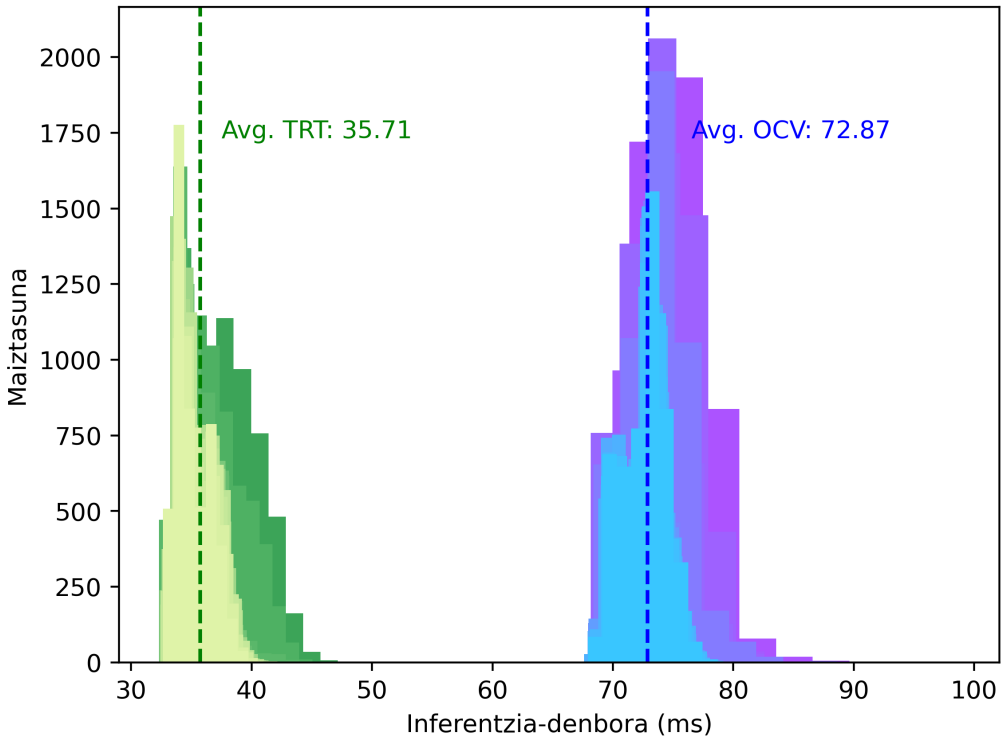
Emaitza hauek modelo berria onartzeko **behar bezain onak** direla iritzi da. Erabilitako neurketaren arabera batak edo besteak emaitza hobeak lortzen ditu, baina oso-oso alde txikiz. Era berean, kontuan izan behar da detektatutako objektuen tamaina txikia dela (urrun daudenean, batez ere), eta *ground truth* moduan ibilitako datubasea eskuz etiketatua izan dela.

8.3 Errendimenduaren ebaluazioa

Proiektuaren helburu nagusietako bat inferentziaren exekuzio-denbora edo latentzia murriztea da, ziklo osoaren zati txikiagoa hartzeko. Helburua bete den ikusteko hainbat exekuzio independente planteatu dira, jatorrizko sarearen eta sare berriaren errendimendua neurtzeko. Aurreko puntuko datu-base berarekin, 5000 irudikoarekin, 19 exekuzio planteatu dira: bana 0.1 eta 1.0 arteko konfiantza-atalase bakoitzeko, 0.05eko jauziarekin.

Esperimentu independenteek ondorio sendoagoak ateratzea ahalbideratuko dute, eta ez laginaren tamaina handitzen delako soilik: datuak biltzeko prozesua luzeagoa izango da, eta sistema egoera ezberdinetan egongo da bitartean, gainerako prozesuek eragindako lankarga txikiago edo handiagoarekin. Bi sareen arteko exekuzioak txandakatzeari esker, egoera gehiagotako datuak eskuratuko dira bi bertsioentzat.

8.3 irudian inferentzien denborak era bisualean laburtu dira, exekuzioei dagozkien histogramak gainjarriz (TensorRT-rekin lortutakoak, berdez; OpenCV-rekin lortutakoak, urdinez). Ikus daitekeenez, bertsio optimizatuaren ia denbora guztiak $[35, 45]$ tartean metatzen dira; jatorrizkoarenak, berriz, $[70, 80]$ inguruan. Grafikoak garbi erakusten du errendimenduan hobekuntza nabarmena egon dela, ggb. exekuzio-denbora erdia behar baitu bertsio optimizatuak.

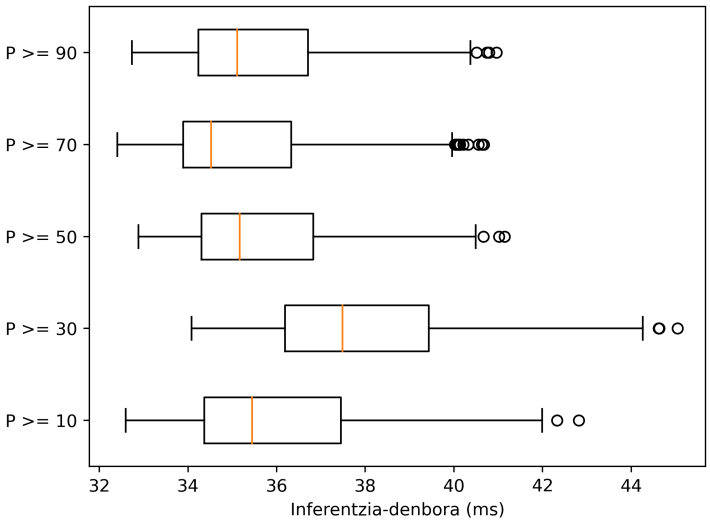


8.3 Irudia: Inferentzia-denboren histogramak, TensorRT (berdez) eta OpenCV (urdinez) erabiliz

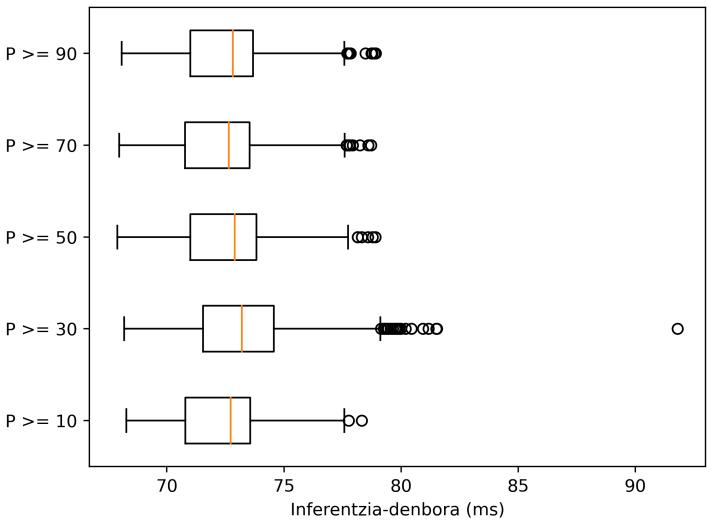
Esperimentuetako batzuk prozesu osoaren ordezkari moduan hautatuz, bakoitzaren neurketa guztiekin kutxa-diagramak eraiki dira, exekuzioen arteko aldeak eta *outlier*-ak garbiago erakusteko (8.4 irudia). Ez du ematen denborak atalasearen menpekoak direnik, ez baitago handitzeko/txikitzeko joera argirik. Bestetik, oso *outlier* gutxi daude, eta modelo

Ekintza	T_exe (ms)	Denbora %
Resize	1.023	2.921
Host2Dev	0.0101	0.031
Inference	32.8656	93.766
Dev2Host	0.0205	0.058
Filter	1.13	3.224

8.2 Taula: Exekuzio-denbora osoaren deskonposaketa



(a) TensorRT erabiliz



(b) OpenCV erabiliz

8.4 Irudia: Exekuzio-denboren kutxa-diagramak, konfidantzaren atalase ezberdinentzat

optimizatuan, horiek kontuan hartuta ere, garbi ikusten da salbuespenetan ere ez dagoela 50ms-ko latentzia baino handiagorik. Orokorrean, exekuziotik exekuziora aldakortasuna txikia izan da. Kasu berezia $P_{obj} \geq 30$ kasuarena da, gainerakoen aldean ezohiko emaitzak ematen baititu: TensorRT-ren kasuan batezbestekoa osoa baino 2ms motelagoa da; OpenCV-renean, aldiz, besteetan baino *outlier* gehiago identifika daitezke. Hipotesi moduan, aldea periodo horretan sistemaren karga ohi baino handiagoa izateak eragin duela suposatu da.

8.3 taulan esperimentu guztien emaitzak laburbildu dira: exekuzio-denbora maximoa, minimoa, batezbestekoa eta desbiderapen estandarra. **Jatorrizko** modeloarekin, batezbesteko **exekuzio-denbora 72.867 ms-koa izan da**; modelo **optimizatuarekin, 35.714koa**: lehengoaren **% 49**. Kasurik onena eta txarrena alderatuz ere emaitzak positiboak dira: bertsio zaharraren exekuzio guztietatik denborarik onena hartuta ere, bertsio berriaren txarrena baino motelagoa da.

Frame bat prozesatzeko behar den denborarekiko (s-tan) alderantziz proportzionala da emaria neurtzen duen FPS edo *Frame* segunduko metrika (betiere, *batching* erabiltzen ez bada). Emandako datuak itzuliz, **14FPStik 28FPSra igarotzea lortu da**.

Denboren azterketarekin amaitzeko, 8.2 taulan esperimentuetako baten exekuzioa prozesuko osagaien denboretan deskonposatu da. Denbora osoaren zatirik handiena (% 93 baino gehiago) TensorRT-n egiten den sarearen exekuzioak hartzen du. Hasierako tamaina-aldaketak eta irteera atalasearen arabera filtratzeak, elkarrekin, gainerako % 7aren zatirik handiena hartzen dute, CUDA-ren memoria-eragiketek (*prefetching*a abiarazteak) denboraren % 1 soilik behar baitute, memoria bateratua erabiltzeari esker.

Azkenik, baliabideen erabilera nolakoa den aztertu da `tegrastats` tresnarekin. Arkitekturaren eta liburutegiaren arteko bateraezintasunengatik DLA koprozesagailuak ezin izan direnez aprobetxatu, GPUaren erabileran ez da murrizketarik lortu: GPUak bere maiztasunaren % 88-90 inguruan funtzionatzen du inferentzian batezbeste, bi bertsioetan. Interpretazioa ezberdina da testuingurua bere osotasunean ikusiz: Polarisen ziklo osoan 35 milisegundu gehiagoz egongo da GPUa libre, bestelako atazetarako. CPUari dagokionez, bi kasuetan erabilera % 12-14 artekoa izan da, nukleoen arteko banaketa nahiko orekatuarekin (kontuan balio hori sistemako prozesu guztien artean osatzen dela). Datu horiek lortzeko programak irudi guztiak memorian kargatzen ditu hasieran, *frame* bakoitza inferentziaren aurretik irakurri beharrean. Horri esker, `tegrastats` hasierako kargaren ostean abiarazita datuen kutsadura murriztu da.

Bertsioa	P_{min}	Irudi kop.	T_{min} (ms)	T_{max} (ms)	$\bar{x}$ (ms)	σ
TensorRT	0.1	5000	32.592	42.818	35.896	1.837
	0.15	5000	32.862	41.695	35.462	1.501
	0.2	5000	34.193	48.594	38.642	2.292
	0.25	5000	32.361	41.154	35.235	1.491
	0.3	5000	34.079	45.046	37.917	2.102
	0.35	5000	32.311	43.995	35.419	1.776
	0.4	5000	32.401	41.944	35.448	1.580
	0.45	5000	32.431	41.412	35.407	1.595
	0.5	5000	32.879	41.145	35.550	1.513
	0.55	5000	32.567	40.808	35.371	1.514
	0.6	5000	32.352	41.032	35.315	1.573
	0.65	5000	32.507	41.398	35.497	1.551
	0.7	5000	32.407	40.675	35.064	1.475
	0.75	5000	32.534	42.146	35.420	1.575
	0.8	5000	32.689	41.982	35.520	1.553
	0.85	5000	32.447	41.703	35.391	1.546
	0.9	5000	32.733	40.962	35.477	1.509
	0.95	5000	32.661	40.520	35.454	1.531
	1.0	5000	32.633	41.538	35.084	1.472
	GUZTIRA				**35.714**	**1.877**
OpenCV	0.1	5000	68.266	78.313	72.360	1.743
	0.15	5000	68.339	98.778	75.118	2.739
	0.2	5000	68.172	86.221	73.250	2.303
	0.25	5000	68.638	91.880	74.727	2.504
	0.3	5000	68.176	91.799	73.165	2.219
	0.35	5000	67.986	78.742	72.490	1.803
	0.4	5000	67.994	79.250	72.410	1.825
	0.45	5000	68.388	90.899	73.762	2.466
	0.5	5000	67.889	78.903	72.579	1.850
	0.55	5000	67.926	78.830	72.362	1.851
	0.6	5000	67.965	78.983	72.452	1.813
	0.65	5000	67.905	78.742	72.539	1.818
	0.7	5000	67.970	78.714	72.314	1.831
	0.75	5000	67.956	78.283	72.419	1.788
	0.8	5000	67.896	79.186	72.387	1.840
	0.85	5000	67.612	79.197	72.504	1.806
	0.9	5000	68.073	78.913	72.500	1.808
	0.95	5000	67.934	78.736	72.489	1.830
	1.0	5000	67.950	79.814	72.649	1.831
	GUZTIRA				**72.867**	**2.156**

8.3 Taula: Esperimentuen emaitzen laburpena (inferentzia-denborak eta desbiderapena)

9. KAPITULUA

Etorkizunerako lan-ildoak

Proiektu honekin hasitako bidean gehiago sakontzeko etorkizunean jorratu daitezkeen lan-ildoak identifikatu dira, orokorrean ikusmen artifizialeko fluxu osoarekin edo zehazki objektuen detekzioarekin lotutakoak.

Hasteko, Casio moduluaren inplementazioa hobetu daiteke, oraindik prototipo bat dela esan baitaiteke. Sendotasunaren izenean, errore-tratamendu aurreratuagoa eta *log* fitxategien sistema bat inplementatu dakizkioke. Era berean, interesgarria litzateke JSON fitxategien bidez konfiguratzeko aukera eskaintzea, gainerako moduluetan bezala. Aukera hori baliatu liteke, gainera, sareen hiperparametroak konfiguratzea errazteko, momentuz horietako zenbait `.h` fitxategietan zuzenean idatzi baitira.

Bertsio honetan modelo optimizatua `FP16` zehaztasunarekin sortuagatik, plakak *fast INT8* modua ere inplementatzen du. Detekzioen kalitatean galera espero daitekeen arren, interesgarria litzateke `INT8`rako kuantizazioa aplikatu eta probak egitea, galdutako zehaztasuna balizko abiadura hobekuntza batek konpentsatzen ote duen ikusteko. Horretarako, Casio moduluaren bihurgailua eta bere konfigurazioak aldatu beharko dira.

Gaur gaurkoz PER sisteman YOLOv4 sarea erabiliagatik, bestelako arkitekturekin probak egitea ongi legoke. Sare horiek Casio-n integratzeko sarrerako eta irteerako interfazeak aztertu, eta beharrezko prozesaketak egingo dituzten metodoak inplementatu beharko dira. TensorRT-ko optimizazioak gainbegiratzea ere eskatuko du, eta beharrezkoa denean, *plugin*ak inplementatzea. Sare berriak probatzeko beharra hainbat iturritatik etor liteke: *Satate of the Art* aldatu delako, eta modelo hobe bat agertu delako; bestelako erabilera-kasu bati erantzun nahi zaiolako (ateen inguruko pertsonak detektatzea, adibidez); DLA-

tan exekutatu daitezkeen ereduak nahi direlako...

Bestalde, epe laburrean aztertuko diren azeleragailuekin errendimendua konparatzeko balio dezakete txosten honetako *hardware*aren analisiak eta emaitzek. Izan ere, CAF Signalling parte den zenbait Europa mailako proiektutan (SELENE [101] eta FRACTAL [102], esaterako) FPGAtan inplementatzeko azeleragailuak diseinatzen ari dira. Horrekin lotuta, eskaintzen duten *software*-pilaren arabera, Casio-n integratu daitezkeen (*backend* berri moduan) edo era independentean tratatu behar diren ebaluatu beharko da.

Une honetan Nvidia etxearen baliabideak erabiltzen direnez, garrantzitsuak izango dira datozen urteetako berrikuntzak [103]. 2021eko uztailerako espero da Jetpack bertsio berria, liburutegi eta tresna eguneratuekin. Horien artean TensorRT-ren eguneraketa espero da, eta kontsultatutako iturriaren arabera *new DLA* moduko kontzeptuak irakurtzen dira. Eguneraketzeak moduluan izango duen eragina neurtu eta beharrezko aldaketak egin beharko dira, hala dagokionean.

Errendimendua ebaluatzeari dagokionez, baliabideen analisi sakonago bat egiteko `NSight` tresna balia daiteke; batez ere, CUDAko exekuzioak aztertzeko. Errendimenduaren azterketa sakonago horren parte izan liteke zehaztasuna metrika aurreratuagoekin lantzea ere.

Amaitzeko, proiektuaren irismenetik haratago doazen arren, fluxu nagusia eraldatuz Polarisen latentzia orokorra murriztea lortzeko hainbat planteamendu ere proposatzen dira, etorkizuneko esperimentu moduan:

- Erabilitako sarea (sare berria) DLA motorrekin bateragarria bada, jatorrizko irudia tamaina txikiagoko bi sare arinago elikatzeko erabil daiteke, bakoitzak irudiaren eremu bat jasoz sarrera moduan (zati gainjarri batekin). Bi inferentziak paraleloan exekutatuz latentzia gutxituko dela espero daiteke.
- Irudia zatikatzeko estrategia bera jarraitu liteke GPUan ere, CPUko eragiketekin dauden datu-dependentziak tartekatzeko. Irudiaren zati bateko detekzioen distantziak kalkulatzen has daiteke CPUa, aldi berean beste zatiaren inferentzia egiten den bitartean.

10. KAPITULUA

Ondorioak

Sare neuronalen azeleragailuen azterketaren helburua taldeari informazio lagungarria eskaintzea izan denez, garrantzi handiagoa eman zaio ikuspegi orokor batetik egungo egoera laburtzeari, sakontasunaren ordez. Salbuespena FPGAk izan dira: Europako proiektuetatik FPGAtarako sortutako azeleragailuak etorriko direnez enpresara etorkizun hurbil/ertainean, gailu-mota horri buruzko informazio gehiago eskaini da.

Xavier AGX gailuaren azterketa egiterakoan bereziki landu da NVDLA arkitekturaren atala, gailu horiek erabiltzeko ahalegin bat egin delako, eta taldeak aurrerago arkitektura hori jarraitzen duten soluzioak erabili beharko dituela aurreikusten delako. Gainerakoan, nahiz eta ondorio praktiko garbirik ez izan, espero gabeko hainbat bitxikeria ere aurkitu dira; besteak beste, CPU supereskalarraren ezohiko diseinua.

Inplementazioari dagokionez, YOLOv4 sarearen optimizazioa ustekabeko erronken segida (ia) amaiezin bat izan da: formatuen arteko bateraezintasunak; TensorRT-ren zenbait atali buruzko informazio eskasia; literaturako sarearen bertsio ezberdinen arteko aldeak; DLA gailuen mugak... Hala ere, azkenean TensorRT liburutegia, *plugin* espezifikoak eta eskuz inplementatutako CUDA *kernel*ak uztartuz sarea optimizatzea lortu da, jatorrizko bertsioaren errendimendua asko hobetuz (GPUrako bertsioan).

DLA motorren erabileran ezin izan da esperotako emaitzarik lortu, geruzen bateraezintasunak eraginda. Optimizazio-prozesu desegokia ikusirik, emaitzen analisitik kanpo utzi da bertsioa. Hala ere, etorkizunerako atea zabalik gelditzen da, ondo bidean Casio moduluak gai izan behar lukeelako sare bateragarri bat DLAtarako optimizatzeko.

GPUarekin, berriz, proiektuaren hasieran ezarritako helburu nagusia bete da. Zehaztasu-

nean galera txiki bat dagoela ematen duen arren, aldea baztergarria dela erabaki da, hainbat arrazoirengatik: erabilitako datu-basea eskuz etiketatua izan da, eta lehen begiratuan etiketatu gabeko objektuak aurkitu dira (batez ere urrun daudenean); detektatutako objektuetako asko oso txikiak dira, eta muga-laukien aldaketa txikiek emaitzan eragin handia izan dezakete; eta bi bertsioen arteko aldeen eskala oso mugatua da, eta aldeak, onargarriak. Helburuetan zehaztutako errendimenduaren hobekuntza agerikoa da inferentziadenborak neurtzerakoan: bertsio optimizatuak jatorrizkoaren erdia behar du. Irudiak banaka prozesatzen direnez, ziklo hori latentziaren berdina eta emariaren alderantzizkoa da: 14FPS-tik 28FPS-ra igaro da *throughput*a.

Sortu den modulu berriak, Casio-k, aukera ematen du erabili nahi den bertsioa exekuziodenboran zehazteko, eta baita modeloak konfigurazio ezberdinekin optimizatzeko ere. Beraz, egoki betetzen du Polaris-en objektuen detekziorako azpi-modulu lana. Horrez gain, ahal izan den heinean, aurrerago beste *framework* edo/eta modeloekin hedatzea erraztea bilatu da diseinu-erabakiekin.

Laborategian egindako probetan sarearen bertsio optimizatua PER sisteman integratzeko prest dagoela erakutsi den arren, memoria idazterakoan oraindik ezin izan da trenean, eszenatoki errealean probatu. Soluzioa proposatu denetik aurrera, trenarekin izan diren probetan bestelako funtzionalitateek, hobekuntzek eta zuzenketek lehentasuna izan dute.

Prozesu horretan zehar hainbat hobekuntza edo ildo identifikatu dira, nahiz eta proiektuaren irismenetik kanpo egon, edo gauzatzeko denborarik ez eman. Ikusi da sare neuronalen optimizazioa ikerketa-arlo zabala dela, eta beti egongo dela zer hobetua.

Laburbilduz, ezarritako helburuak bete dira: objektuen detekzioa *hardware* azpiegiturara egokitzea lortu da, errendimendua nabarmen hobetuz; soluzio konkretua orokortu eta *software* modulu batean bildu da, erabilgarriago egiteko; eta egungo egoera ikertu eta informazio esanguratsua bildu da PER taldearentzat. Eta bidean ikusi eta ikasitakoak garbi erakusten duenez, proiektu hau lehen pausu bat baino ez da izan treneko ikusmen artifizialeko sistemaren *hardware* bidezko azelerazioaren bidean.

A. ERANSKINA

ATP eta ATO sistemak

A.1 ATP

ATP sistemak trenaren operazioak babesteaz arduratzen dira, *tracksidetik* jasotako eta aurrez ezagututako informazioaren bidez trena zuzenean kontrolatuz [104]. XX. mendearen hasieran lehen sistemak martxan jarri ziren, eta ordutik hainbat herrialdek estandar propioak garatu dituzte.

Segurtasun- edo *safety*-funtzionalitateak bete behar dituztenez, integritate-irizpide eta kontrol zorrotzak bete behar dituzte, funtzionamendua egokia izango dela ziurtatzeko. Betebehar horiek SIL (*Safety Integrity Level*) mailetan sailkatzen dira [105], eta ETCS gainean ATPa SIL4 da, mailarik altuenekoa.

ETCS

Herrialdeetako babes-sistema propioak ordezteko jaiotako europar iniziatiba da ETCS (*European Train Crontrol System)*. Europar Batasuneko herrialdeetako trenbide-sareen ateko batergarritasuna handitu, eta ATP estandar moderno bat definitu nahi ditu.

Hiru ETCS maila nagusi daude definituta [106]:

- **Level 1** (A.1a irudia)**:** ATP sistemak trenaren egoera (uneko abiadura, balaztakurba...) etengabe monitorizatzen du trenbideko balizekin aldizkako komunikazioa mantenduz (baliza batetik igarotzean informazio eguneratua jasotzen du). Maila

honetan beharrezkoak dira bide-ertzeko seinale tradizionalak ere. ETCSren osagarri moduan, *trackside* azpiegitura arduratzen da kantoietako trenak detektatzeaz, eta integritatea diagnostikatzeaz.

- **Level 2** (A.1b irudia)**:** aurreko mailako elementuez gain, ATPak irrati-uhinen bidezko etengabeko komunikazioa du bideko RBC (*Radio Block Centre*) irrati-gailuekin. Informazio konstantea eta osoagoa jasotzen duenez trenak, maila honetan, abiadura mugak eta bide bereko trenn frekuentzia handitzeaz gain, ohiko seinaleak seinale birtualen bidez ordezka daitezke.

- **Level 3** (A.1c irudia)**:** trenaren eta kontrol-zentroaren arteko etengabe komunikazioa ezartzen da, gutxieneko balizak eta komunikaziorako beharrezko azpiegitura izan ezik, *tracksideko* gainerako elementuak kentzeko aukera emanez. Trenen kokapena eta integritatea ere trenak kontrolatzen ditu. Informazio-jario horrek kantoi dinamikoak eta bestelako hobekuntzak ezartzeko aukera ematen du, lineak abiadura handiagoan eta dentsitate handiagoz okupatu ahal izateko, segurtasuna bermatzen jarraituz.

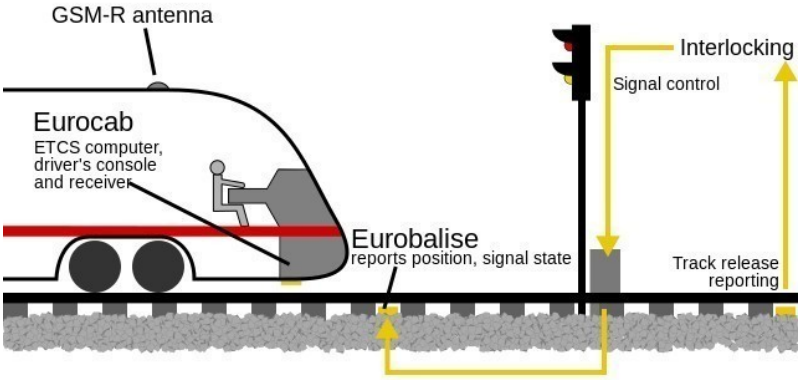
  Hiru maila horiez gain, bateragarritasuna mantentzeko bi maila berezi ere definitzen dira estandarrean: **Level 0** mailan ETCSdun trenak sistema hori ez daukaten bideetan barrena doazenean ematen da; **Level STM** maila, berriz, B klaseko babes-sistemak dituzten bideetan daudenean. Azken kasu horretan, ETCSren funtzioa gidariaren eta dagokion herrialdeko sistema propioaren arteko zubi-lanak egitea da.

A.1.1 ERTMS

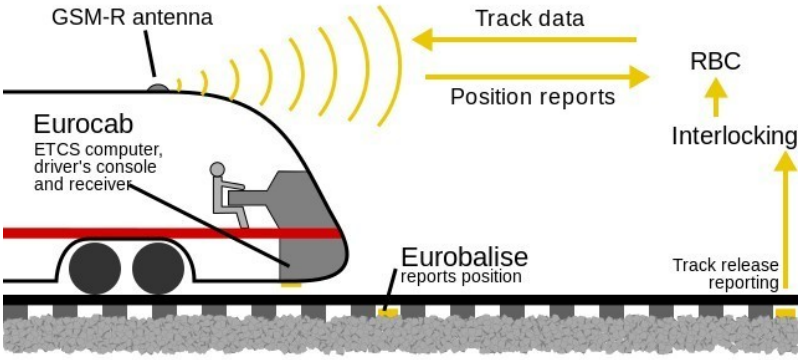
ERTMS edo *European Rail Traffic Management System* Europar Batasunaren bateragarritasuna bultzatzeko teknologia estandarrak biltzen dituen iniziatiba da; zehatzago, kudeaketa definitzen duen ETCS estandarrak eta bere elementuen arteko komunikaziorako aukeratutako teknologiak (GSM-R) osatzen dute [107].

A.2 Automatizazio-graduak

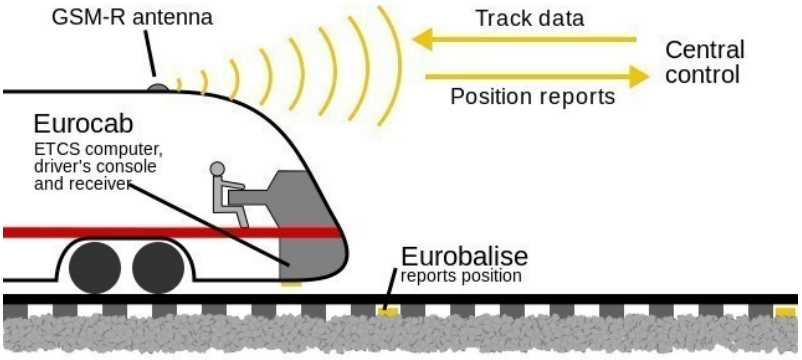
Europar Batasuneko erakundeek eta Europako tren-industriako konpainiarik garrantzitsuenek (CAF taldea, Alstom, Hitachi, Siemens, Thales...) 2009an sortu zuten iniziatiba



(a) 1. mailako ETCS

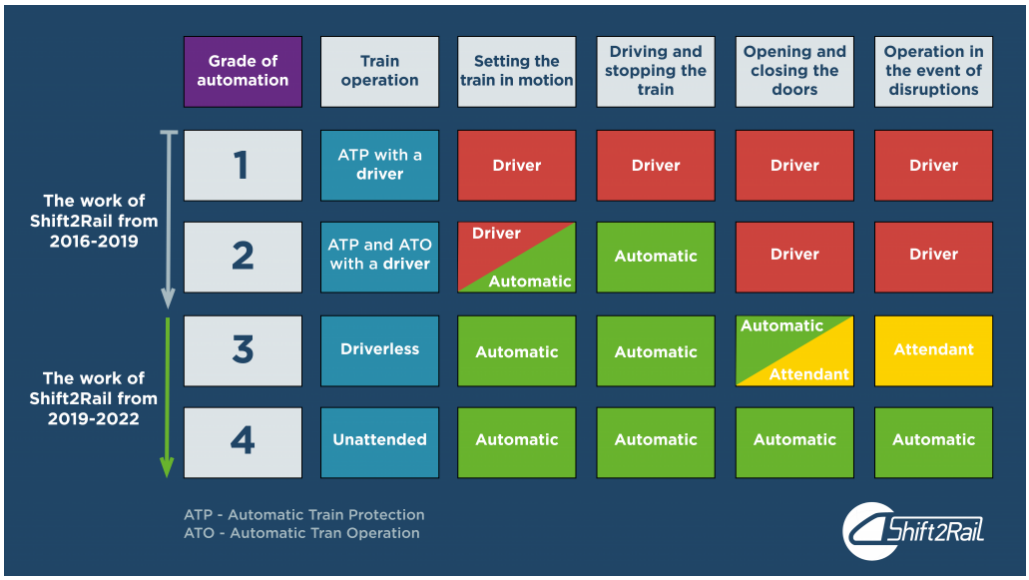


(b) 2. mailako ETCS



(c) 3. mailako ETCS

A.1 Irudia: ETCS sistemaren mailen funtzionamenduaren azalpen grafikoa



A.2 Irudia: Trenen automatizazio-graduak

da *Shift2Rail* edo S2R, kontinenteko trenbide-sarea eta trenen operazioak bateratu eta modernizatzeko helburuarekin [108]. Bide horretan sektoreko I+G+B aurrerapenak elkarlanean eta era koordinatuan egiteko balio izan du iniziatibak, seinalizazioaren berriztapena, estandar berrien garapena... ahalbideratuz.

Lan honetan trenen automatizazioaz eta automatizazio-graduez hitz egiterakoan *Shift2Rail* egitasmoan definitutako sailkapena hartuko da erreferentziatzat [109]. A.2 irudian sailkapen horretako lau mailak ikus daitezke: **GoA1**ean trena gidatzeaz gidaria arduratzen da, baina laguntza moduan babeserako sistema automatikoak ditu (ATPa, adibidez); **GoA2**n, trenaren operazioen zati bat **ATO**aren edo pilotu automatikoaren esku gelditzen da (egoera normalean gidatzea), baina ataza espezifikoak (geltokietako kontrola, eta ezohiko gertaeren aurreko erantzuna) sistemaren irismenetik kanpo daude; **GoA3** edo **DTO** (*Driverless Train Operation*) mailan trenaren kontrol osoa automatikoa da, baina gidariaren edo langileren baten gainbegiraketa mantentzen da larrialdietarako (sistemarik aurreratuenetan gidaria trenetik kanpo egon daiteke); azkenik, **GoA4** edo **UTO** (*Unattended Train Operations*) graduko trenak guztiz era autonomoan funtzionatzen dute, gizakien eskuhartzerik gabe.

Gaur egungo linearik gehienek segurtasunerako funtzionalitateak gutxienez automatizatuta izan ohi dituzte (GoA1); izan herrialde bakoitzeko inplementazio propioaren bidez, izan ETCS (Europan) edo bestelako estandarren betetzen duten sistemen bidez. Trenak automatikoki gidatzen aitzindaria izan zen Erresuma Batuko Londres hiriko metroko *Victoria* linea, 1970eko hamarkadarako kontrol automatikoa martxan baitzeukan [110]; egun,

ostera, GoA2 mailako ATO soluzioak metrotik ohiko trenbideetara jauzia ematen ari da [111]. [112] GoA3 automatizazioa ezartzeko proiektuetako baten adibidea da: bideo- eta sentsore-bidezko hautematea, urruneko agintea etab. ari dira probatzen I+G+B proiektu moduan. GoA4 trenen erabateko automatizazioa lortzea litzateke, eta horregatik azken helburu moduan planteatu ohi da [109]. Azken bi mailetako sistemak modu operazionalean ingurune kontrolatuetan soilik agertzen dira oraingoz: metroetan, tren-biltegietan...

Industriak trenbideen ustiaketaren efizientzia nabarmen hobetzea espero du automatizazioari esker [113]: ezarritako perfilarentzat gidatze optimotik gertuago gidatuz (gehiegi trakzionatu eta gehiegi frenatu gabe) energiaren kontsumoan aurreztea espero da, bidaiarien erosotasuna hobetzearekin batera; abiaduraren eta ibilbideen kontrol zorrotzagoari esker, puntualitatean irabaztea; eta giza-akatsak gutxituz, modu seguruan trenbideetako trenen dentsitatea handitzea.

B. ERANSKINA

Sare neuronalen entrenamendua

Sare neuronal baten irteera esperotakoaren edo helburuaren zenbat eta antzekoagoa izan, sareak problemari orduan eta erantzun egokiagoa eman dio. **Pisuak eta *bias*ak doitzeko prozesua**ri entrenamendu edo ***training*** deritzo. Problema motaren arabera, ikasketa gainbegiratua (*supervised*), gainbegiratugabea (*unsupervised*) edo bien arteko nahasketa (*semi-supervised*) izan daiteke [114]:

- ***Supervised** Learning*: sarrerako datuen desiotako irteerak (*ground truth* izena hartu ohi dute) ezagunak dira, etiketatze- edo *labeling*-lan bati esker. Entrenatzerakoan sarearen irteera esperotako emaitza horietatik ahalik eta gutxien desbideratzea da helburua, prozesuan zehar sarearen parametroak horretarako doituz. Metodo hau sailkapen eta detekzioetan, iragarpenetan eta aurreikuspenetan erabili ohi da gehien (irudien, testuen edo ahotsaren sailkapena, eguraldi-iragarpenak...).

- ***Unsupervised** Learning*: datu-multzo handi batetik patroi ezkutuak erauzten saiatzen da sarea. Horretarako, datuen arteko antzekotasunak ustiatzen ditu, aurrez eskuzko etiketatzea egiteko beharrik gabe. Oso erabilgarriak dira multzokatzean (*clustering*, denda bateko bezeroak taldekatzeko), asoziazioan (*association*, antzeko abestiak gomendatzeko) edo dimentsio-murriztapenetan (datuak aurre-prozesatzeko).

- ***Semi-supervised** Learning*: datu etiketatuak eta etiketatu-gabeak elkarrekin erabiltzean data. Bolumen altuko datu-baseak osorik etiketatu beharrik ez izateko, datuetatik ezaugarri jakinak erauzteko, zehaztasuna fintzeko... helburuekin erabil daiteke, eta inplementatzeko modua erabilera-kasutik erabilera-kasura aldatzen da.

Aipatu, hirugarren paradigma bat ere badagoela, normalean problema dinamikoetan (mugimenduak edo ekintzak egiterakoan) aplikatu ohi dena: *reinforcement learning*. Sinplifikatuz, metodo honetan ekintza-sari eta -zigor bikoteak erabiltzen dira, entrenatu nahi den agenteak ingurunearekiko harremanaren bidez sariak metatutko dituzten ekintzak aukeratzen ikasteko[115].

Irudietan objektuak detektatzeko **entrenamendu gainbegiratua** erabili ohi denez, datozen puntuetan teknika horren ezaugarriak sakonduko dira.

B.1 Kostu-funtzioa

Sarearen irteeraren eta egiazko balioaren arteko aldea neurtzeko kostu-funtzioa edo *cost-function* deiturikoa erabiltzen da[1]. Funtzioa sarearen irteeraren eta esperotakoaren arteko aldeari (zuzenean kalkulatutakoa) aplikatzen zaio; ondorioz, kostua, inplizituki, pisuen eta *bias*en menpe dago.

Irudietan objektuak detektatzerakoan bi helburu bete behar dira aldi berean: objektua zein motatakoa den igartzea, eta non dagoen kalkulatzea. Irudi bateko detekzio bakoitzeko, detekzioa detektatu nahi den kategoria edo mota bakoitzekoa izateko probabilitateak biltzen dituen bektoreak eta objektuaren kokapenak (koordenatuak eta tamaina) osatzen dute.

Sailkapenaren zehaztasuna neurtzeko MSE (*Mean Square Error*), MAE (*Mean Absolute Error*) eta *Logarithmic Loss* dira kostu-funtziorik erabilienak. Esaterako, x sarrerako datu bakoitzarentzat $y(x)$ sarearen irteera-bektorea eta $y^*(x)$ sarreraren etiketatutako irteera izanik (dagokion klasea 1, gainerakoak 0), W pisuen matrizea eta B *bias*-en bektorea bada, B.1 ekuazioak erakusten du n sarrerarentzat MSE balioa.

$$C(W,B) = \frac{1}{2n}\sum_{x=1}^{n} ||y^*(x) - y(x)||^2 \tag{B.1}$$

Iragarritako kokapenaren eta egiazkoaren arteko antza, berriz, IoU familiako metriken bidez eman ohi da (***Intersection over Union***). Kokapen bakoitza ***Bounding Box*** baten bidez adierazten bada ($BB = (x,y,w,h)$, adibidez), eta bere azalera $A(BB)$ bidez, BB' detekzioaren emaitza eta BB^* objektuaren kokapen erreala izanik, B.2 ekuazioan ikus daiteke IoU balioaren kalkulua.

[1] Literaturan galera- eta helburu-funtzio ere deitu ohi zaio; *loss-function* eta *objective-function*.

$$IoU = \frac{A(BB' \cap BB^*)}{A(BB' \cup BB^*)} \tag{B.2}$$

Detekziorako teknikek bi neurketa horietatik eratorritako metrika aurreratuagoak konbinatu behar dituzte, eskatzen zaizkien bi helburuak bete ahal izateko.

B.2 Backpropagation algoritmoa

Sarearen irteeraren arabera kalkulatzen den kostu-funtzioak sareko pisu eta *bias* guztiekiko menpekotasuna du, baina milaka parametro dituen sare batean konbinaziorik egokiena era analitikoan bilatzea ez da bideragarria. Aitzitik, parametroen balio posibleen azpi-atal bat soilik espoloratzen duten teknikak erabili behar izaten dira.

Literalki itzulita, errorea atzerantz hedatzea esan nahi du ***error backpropagation*** algoritmoaren izenak. 1970eko hamarkadan proposatu zen, eta 1986tik sare neuronalak entrenatzeko algoritmorik erabiliena da.

[19] liburuak B.3 ekuazio-multzoan ageri diren lau ekuazioak aipatzen ditu *backpropagation* algoritmoaren oinarri modura[2]. l geruzako j neurona bakoitzaren baturak sarearen kostu-funtzioan duen eragina neurtzen da, δ^l_j, C funtzioaren z^l_j-rekiko deribatu partzialaren bidez B.3a. Azken geruzaren deribatu partzialek $\nabla_a C$ bektorea osatzen dute. Bestalde, geruza baten sarrerako errorea (δ^l) hurrengoaren errorearen, bere aktibazioaren deribatuaren eta bi geruzen arteko pisuen menpekoa izango da B.3b. Bigarren ekuazio horrek errorea sarean zehar atzetik aurrera "mugitzeko"aukera ematen du. Azkenik, kostua edozein *bias* edo pisuren arabera nola aldatzen den azter daiteke: definizioz, *bias* baten aldaketak kostu osoan duen eragina dagokion neuronaren errorearen berdina da B.3ceq; $l-1$ geruzako k neurona eta l geruzako j lotzen dituen pisuaren eragina, berriz, l kapako j neuronaren errorearen eta aurreko geruzako k neuronaren aktibazioaren menpekoa izango da B.3d.

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{B.3a}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{B.3b}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{B.3c}$$

[2] 2. kapitulua algoritmoa azaltzeari dedikatzen dio. Oinarri matematikoagoa sakonago ezagutzeko, jo hona: *Chapter 2: How the backpropagation algorithm works*.

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{B.3d}$$

Algorithm 1: *Backpropagation* algoritmoa

Input: $X = \{x_1, ..., x_n\}$, entrenamendurako datu-multzoa
Input: $\sigma(z)$, neuronen aktibazio-funtzioa
Input: $\sigma'(z)$, aktibazio-funtzioaren lehen deribatu-funtzioa
Input: $f(x)$, sarrerako neuronen aktibazio-funtzioa
Input: $W = (w^1 ... \ w^L)$, pisuen matrizea (bektoreen bektorea)
Input: $B = (b^1 ... \ b^L)$, *bias*en matrizea
Input: η, ikasketa-koefizientea
Input: L, geruza-kopurua
Output: W eta B eguneratuak

for $i \leftarrow 1$ ***to*** n **do**
  $x \leftarrow x_i$
  $a^{1,i} \leftarrow f(x)$ // Lehen geruzaren aktibazioa sarrerako datuetatik

  // Feedforward
  for $l \leftarrow 2$ ***to*** L **do**
    $z^{i,l} \leftarrow w^l a^{i,l-1} + b^l$
    $a^{i,l} \leftarrow \sigma(z^{i,l})$
  end

  $\delta^{i,L} \leftarrow \nabla_a C_i \odot \sigma'(z^{i,L})$ // Azken geruzaren errorea kalkulatu

  // Backpropagation
  for $l \leftarrow L-1$ ***to*** 2 **do**
    $\delta^{i,l} \leftarrow ((w^{l+1})^\top \delta^{x,l+1}) \odot \sigma'(z^{i,l})$
  end
end

// Stochastic Gradient Descent
for $l \leftarrow L$ ***to*** 2 **do**
  $w^l \leftarrow w^l - \frac{\eta}{n} \sum_{i=1}^{n} \delta^{i,l} (a^{i,l-1})^\top$
  $b^l \leftarrow b^l - \frac{\eta}{n} \sum_{i=1}^{n} \delta^{x,l}$
end

1 algoritmoan ekuazioak sare baten parametroak doitzeko nola aplikatzen diren ikus daiteke. Datuak sarreran jarri eta sarea zeharkatzen da irteerak lortu arte; irteera horiekin azken geruzaren errorea kalkulatzen da; eta errorea modu ponderatuan atzerantz igortzen da. Geruza bakoitzean errore hori erabil daiteke parametroak doitzeko, *Gradient Descent*

[3] bezalako eguneraketa-irizpideren bat jarraituz.

Datuen bolumena handia denean, ***mini batch*** deituriko sortetan eman ohi zaizkio datuak algoritmoari, eta parametroen eguneraketa multzo horretako datuen arabera egin [4], datubase osoa kontuan hartu beharrean. Iterazio bakoitzean datu-basetik datu-multzo bat aukeratzen da, eta hainbat iterazioren ondoren instantzia guztiak erabili direnean, ***epoch*** bat igaro dela esan ohi da.

B.3 Hipermarametroen doikuntza

Sare baten parametroak bere neuronen *bias*ak eta loturen pisuak dira, eskuragarri dauden datuetara bereziki egokituak, entrenamenduaren ostean. Hipermarametroak, ordea, entrenatu aurretik ezartzen direnak dira, sarearen eta entrenatze-prozesuaren ezaugarriak definitzen dituztenak: sarearen topologia edo arkitektura (neurona- eta geruza-kopurua), parametroak eguneratzeko ikasketa-koefizientea, sortaren tamaina, *epoch* kopurua... Horiez gain, irudiekin lan egiterakoan bestelako erabakiak ere hartu behar dira; besteak beste, sarrerako irudien bereizmena edo *pixel* kopurua.

Gaur egun, topologia ezagunak edo topologia-familia ezagunak oinarrizko ikerketan proposatu ohi dira, eta aplikazio praktikoetan, erabilerara gehien egokituko dena aukeratu. Kapen eta neuronen antolamendua, aktibazio-funtzioak eta abar arkitekturan bertan definitzen dira.

Entrenamenduarekin lotutako hipermarametroak ezartzeko **zehaztasunaren eta abiaduraren arteko *tradeoff*** a ("Zenbat denbora dago entrenatzeko?") kontrolatu behar da, *batch* bakoitzaren tamaina, ***epoch*** kopurua eta ***learning-rate*** balioak egokituz. Azken horrek, gainera, eragina izango du optimo lokalak ekiditerakoan ere. Heuristikoen bidez aukeratzen dira hiperparametro egokiak [117]: aurreko esperientzian edo literaturan oinarrituz; konbinazio multzo diskretuak aztertzen dituen *grid search* teknikarekin; bilaketa heuristiko aurreratuagoekin...

Hiperparametro hauek garrantzi handia dute *Machine Learning* mota ugariren (sare neuronalak barne) entrenamenduan agertzen diren arazoak ekiditeko:

[3] Iterazio bakoitzean kostu-funtzioaren gradientea (edo bere hurbilpen bat) erabiltzen da parametroak eguneratzeko, minimo lokalera gerturatzeko [116].

[4] Azpimultzo edo lagin bat erabiliz estimatutako gradienteak erabiltzeari *Stochastic Gradient Descent* deritzo [116].

- ***Overfitting*:** parametroak entrenamenduko datuetara gehiegi egokitu dira, eta modeloak orokortasuna galdu du. Ebatzi nahi den problemaren ezaugarriak erauzi beharrean, erabilitako adibideenak ezagutzen ditu, eta datu horiekin oso emaitza onak eman arren, ez du hain ongi funtzionatuko kasu berriekin.
- ***Underfitting*:** sarea behar baino gutxiago entrenatu da, eta oraindik ez da gai izan datuetatik informazio esanguratsua lortzen ikasteko.

B.4 Balidazio-metrikak

Objektuen detekzioaren kalitatea ebaluatzeko erabiltzen diren teknika eta metrika ugarien artetik [100], atal honetan azaldutakoak dira proiektuan erabili direnak.

Bi baldintza hauek kontuak izanik:

1. Detekzioari sareak esleitu dion probabilitatea gutxienez aurrez ezarritako konfidantza-atalasea bezain handia izatea.
2. Detektatutako objektuaren eta dagokion *ground truth*aren arteko *IoU* balioa aurrez ezarritako *IoU* atalasea bezain handia izatea, gutxienez.

1. baldintza betetzen ez duten detekzioak *False Negative* edo FN moduan ulertuko dira; 1.a soilik betetzen dutenak, *False Positive* edo FP. Bi baldintzak betetzen dituztenak, berriz, *True Positive* edo TP. Azkenik, detekzio baliokiderik ez duten *ground truth*eko objektuak ere FN izango dira.

Neurri horietatik ondoriozta daitezke ***Precision*** edo P (B.4 ekuazioa) eta ***Recall*** edo R (B.5 ekuazioa) balioak. Lehenak sareak egindako detekzioetatik zuzenek osatzen duten zatia neurtzen du; bigarrenak, egin beharko liratekeen detekzioetatik zenbat egin diren. Bi balioen arteko oreka kontuan hartzen du ***F-score*** familiako F_1 (B.6 ekuazioa) balioak.

$$P = \frac{TP}{TP + FP} \tag{B.4}$$

$$R = \frac{TP}{TP + FN} \tag{B.5}$$

$$F_1 = 2\frac{TP}{TP + \frac{1}{2}(FP + FN)} = 2\frac{P \times R}{P + R} \tag{B.6}$$

Bibliografia

[1] Jon Mikel Olmos Serna. Reconocimiento de señales verticales ferroviarias utilizando redes neuronales. Master's thesis, Euskal Herriko Unibertsitatea, Informatika Fakultatea, 7 2020.

[2] Señalización ferroviaria, surgimiento y desarrollo hasta el ERTMS. *Railway Signalling EU*, 2015.

[3] Russian autonomous train makes first test run. https://www.railtech.com/digitalisation/2019/08/30/russian-autonomous-train-makes-first-test-run/.

[4] Computer vision for self-driving trams in Shanghai. https://www.railtech.com/digitalisation/2020/04/23/computer-vision-for-self-driving-trams-in-shanghai/.

[5] CAF Signalling. 1. eta 2. mailako ETCS. https://www.cafsignalling.com/eu/produktuak-eta-zerbitzuak/1-eta-2-mailako-etcs/.

[6] CAF Signalling. ATO ERTMS-arekin. https://www.cafsignalling.com/eu/produktuak-eta-zerbitzuak/auriga-ato/.

[7] Joanes Plazaola Madinabeitia. Trenbide-arloko ibilgailu autonomoak: Denbora errealeko ikusmen teknika sakonak sistema txertatuetan, 6 2020.

[8] Muhammad Asad Bilal Fayyaz and Christopher Johnson. Object detection at level crossing using deep learning. *Micromachines*, 11(12), 2020.

[9] Johannes Wallner., Tito Tang., and Markus Lienkamp. Development of an emergency braking system for teleoperated vehicles based on LiDAR sensor data. In

Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics - Volume 1: ICINCO,, pages 569–576. INSTICC, SciTePress, 2014.

[10] Desheng Xie, Youchun Xu, and Rendong Wang. Obstacle detection and tracking method for autonomous vehicle based on three-dimensional LiDAR. *International Journal of Advanced Robotic Systems*, 16(2):1729881419831587, 2019.

[11] Shane Murray. Autonomous vehicle RADAR perception in 360 degrees. https://developer.nvidia.com/blog/autonomous-vehicle-radar-perception-in-360-degrees/.

[12] Asra Aslam and M. S. Ansari. Depth-map generation using pixel matching in stereoscopic pair of images. *ArXiv*, abs/1902.03471, 2019.

[13] Jon Otegui, Alfonso Bahillo, Iban Lopetegi, and Luis Enrique Díez. A survey of train positioning solutions. *IEEE Sensors Journal*, 17(20):6788–6797, 2017.

[14] CAF Signalling. CAF Signalling-ek arrakastaz egin ditu NS-ren (Holandan) unitateen gidatze automatikoko lehen probak. https://www.cafsignalling.com/en/caf-signalling-success-in-the-ato-test-on-etcs-in-ns-of-netherlands/. 2019/12/17.

[15] Classification, object detection and image segmentation. https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk/learning-resources/image-segmentation-deeplab-neural-processing-sdk/classification-object-detection-segmentation.

[16] Zhengxia Zou, Z. Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *ArXiv*, abs/1905.05055, 2019.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[18] ImageNet large scale visual recognition challenge (ILSVRC). https://www.image-net.org/challenges/LSVRC/.

[19] Michael Nielsen. Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com.

[20] Eda Kavlakoglu. AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the difference? https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks. 2020/05/27.

[21] Larry Hardesty. Explained neural networks and deep learning. *MIT News*.

[22] Chigozie Nwankpa, W. Ijomah, A. Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *ArXiv*, abs/1811.03378, 2018.

[23] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, T. Liu, X. Wang, Gang Wang, Jianfei Cai, and T. Chen. Recent advances in convolutional neural networks. *Pattern Recognit.*, 77:354–377, 2018.

[24] Asifullah Khan, A. Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, pages 1 – 62, 2020.

[25] Christian Szegedy, W. Liu, Y. Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, V. Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[26] K. Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[27] W. Liu, Dragomir Anguelov, D. Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and A. Berg. SSD: Single shot multibox detector. In *ECCV*, 2016.

[28] Ross B. Girshick, J. Donahue, Trevor Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[29] Shaoqing Ren, Kaiming He, Ross B. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015.

[30] Joseph Redmon, S. Divvala, Ross B. Girshick, and A. Farhadi. You Only Look Once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[31] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017.

[32] Sik-Ho Tsang. Review: YOLOv2 & YOLO9000.

[33] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *ArXiv*, abs/1804.02767, 2018.

[34] Alexey Bochkovskiy, Chien-Yao Wang, and H. Liao. YOLOv4: Optimal speed and accuracy of object detection. *ArXiv*, abs/2004.10934, 2020.

[35] Alexey Bochkovskiy (GitHub/AlexeyAB). YOLO v4, v3 and v2 for Windows and Linux. https://github.com/AlexeyAB/darknet. Kontsultatutako bertsioaren azken eguneratzea: 2021/04/26.

[36] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37:1904–1916, 2015.

[37] Shu Liu, Lu Qi, Haifang Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[38] Diganta Misra. Mish: A self regularized non-monotonic neural activation function. *ArXiv*, abs/1908.08681, 2019.

[39] Real-Time object detection on COCO. https://paperswithcode.com/sota/real-time-object-detection-on-coco.

[40] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[41] Yizhi Liu, Y. Wang, Ruofei Yu, Mu Li, Vin Sharma, and Yida Wang. Optimizing CNN model inference on CPUs. In *USENIX Annual Technical Conference*, 2019.

[42] Selima Curci, D. Mocanu, and Mykola Pechenizkiy. Truly sparse neural networks at scale. *ArXiv*, abs/2102.01732, 2021.

[43] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Penksy. Sparse convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Liu_Sparse_Convolutional_Neural_2015_CVPR_paper.pdf. 806-814 orrriak. CV Foundation bidez atzigarri.

[44] Ziheng Wang. Sparsednn: Fast sparse deep learning inference on CPUs. *ArXiv*, abs/2101.07948, 2021.

[45] Intel. OpenVINO toolkit overview. https://docs.openvinotoolkit.org/latest/index.html.

[46] Intel. Intel Deep Learning Boost. https://www.intel.com/content/dam/www/public/us/en/documents/product-overviews/dl-boost-product-overview.pdf.

[47] Pradeep Gupta. CUDA refresher: Reviewing the origins of GPU computing. https://developer.nvidia.com/blog/cuda-refresher-reviewing-the-origins-of-gpu-computing/. 2020/04/23.

[48] Top500. Top500 - list statistics. https://www.top500.org/statistics/list/. 2020ko azaroko zerrenda. "Accelerator/CP Family"kategoria.

[49] Anthony Spadafora. Nvidia now powers a majority of the world's top 500 supercomputers. https://www.techradar.com/news/nvidia-now-powers-a-majority-of-the-worlds-top-500-supercomputers.

[50] GPU-based deep learning inference: A performance & power analysis. https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf. 2015/11.

[51] Nvidia. CUDA C++ programming guide: Design guide. https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf. Kontsultatutako bertsioa: v11.3, 2021/04.

[52] Jonathan A. Thompson and Kristofer Schlachter. An introduction to the OpenCL programming model. 2012.

[53] K. Karimi, N. Dickson, and F. Hamze. A performance comparison of CUDA and OpenCL. *ArXiv*, abs/1005.2581, 2010.

[54] Diksha Moolchandani, Anshul Kumar, and Smruti R. Sarangi. Accelerating cnn inference on ASICs: A survey. *Journal of Systems Architecture*, 113:101887, 2021.

[55] Jeff Loucks Duncan Stewart, Mark Casey and Craig Wigginton. Bringing AI to the device: Edge AI chips come into their own. https://www2.deloitte.com/us/en/insights/industry/technology/technology-media-and-telecom-predictions/2020/ai-chips.html.

[56] Neuromation. What's the deal with "AI chips" in the latest smartphones? https://medium.com/neuromation-blog/whats-the-deal-with-ai-chips-in-the-latest-smartphones-28eb16dc9f45.

[57] Cliff Young Kaz Sato and David Patterson. An in-depth look at Google's first Tensor Processing Unit (TPU). *Google Cloud Blog*.

[58] Shashank Prasanna. A complete guide to AI accelerators for deep learning inference. https://towardsdatascience.com/a-complete-guide-to-ai-accelerators-for-deep-learning-inference-gpus-aws-inferentia-and-amazon-7a5d6804ef1c.

[59] Umer Farooq, Zied Marrakchi, and Habib Mehrez. *FPGA Architectures: An Overview*, pages 7–48. Springer New York, New York, NY, 2012.

[60] K. Guo, Shulin Zeng, J. Yu, Y. Wang, and H. Yang. A survey of FPGA based neural network accelerator. *ArXiv*, abs/1712.08934, 2017.

[61] K. Abdelouahab, M. Pelcat, J. Sérot, and F. Berry. Accelerating CNN inference on FPGAs: A survey. *ArXiv*, abs/1806.01683, 2018.

[62] Ran Wu, Xinmin Guo, Jian Du, and Junbao Li. Accelerating neural network inference on FPGA-based platforms—a survey. *Electronics*, 10(9), 2021.

[63] Erwei Wang, J. J. Davis, P. Cheung, and G. Constantinides. Lutnet: Rethinking inference in FPGA soft logic. *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 26–34, 2019.

[64] RedHat. What is edge computing? https://www.redhat.com/en/topics/edge-computing/what-is-edge-computing.

[65] Nvidia. Embedded system with Jetson. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.

[66] Dustin Franklin. Nvidia Jetson AGX Xavier delivers 32 teraops for new era of AI in robotics. https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/?ncid=so-fac-mdjngxxrmllhml-69163, 2018.

[67] Nvidia. Jetson AGX Xavier module data sheet. Kontsultatutako bertsioa: v0.9; 2014-2015.

[68] S. Wang, Anuj Pathania, and T. Mitra. Neural network inference on mobile SoCs. *IEEE Design & Test*, 37:50–57, 2020.

[69] ARM Ltd. What do we mean by architecture? https://developer.arm.com/documentation/102404/0200/What-do-we-mean-by-architecture-. Kontsultatutako bertsioa: v2.0.

[70] ARM Ltd. C3.2 Loads and Stores. In *ARM Architecture Reference Manual: ARMv8, for Armv8-A architecture profile*, pages 219–237. 2013. Kontsultatutako azken bertsioa: Armv8.7, 2021.

[71] Andrew Wafaa. Introducing the 64-bit ARMVv8 architecture. http://andrew.wafaa.eu/files/EuroBSDConARMv8.pdf.

[72] ARM Ltd. ARMv8 architecture introduction and overview. In *ARM Architecture Reference Manual: ARMv8, for Armv8-A architecture profile*, pages 34–62. 2013. Kontsultatutako azken bertsioa: Armv8.7, 2021.

[73] Wiki Chip. Carmel microarchitecture Nvidia. https://en.wikichip.org/wiki/nvidia/microarchitectures/carmel. Wiki komunitarioa.

[74] Darrell Boggs, G. Brown, Nathan Tuck, and K. S. Venkatraman. Denver: Nvidia's first 64-bit ARM processor. *IEEE Micro*, 35:46–55, 2015.

[75] Jaime H Moreno Krishnan Kailas. VLIW architecture. https://researcher.watson.ibm.com/researcher/view_group_subpage.php?id=2833.

[76] Nvidia. Nvidia Tesla V100 GPU architecture. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf. Kontsulta @misctutako bertsioa: v1.1, 2017/08.

[77] Nvidia. Nvidia Tesla P100, NVLink high speed interconnect. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-

architecture-whitepaper-v1.2.pdf. 20-24 orriak. Kontsultatutako bertsioa: v1.1.

[78] S. Markidis, S. W. Chien, E. Laure, I. Peng, and J. Vetter. Nvidia tensor core programmability, performance & precision. *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 522–531, 2018.

[79] Nvidia. NVDLA. http://nvdla.org/.

[80] Nvidia. NVDLA hardware architectural specification. http://nvdla.org/hw/v1/hwarch.html.

[81] Nvidia. NVDLA open source hardware. https://github.com/nvdla/. Azken aldatze-data kontsultatzerakoan: 2019/08.

[82] Nvidia. L4T supported modes and power efficiency. https://docs.nvidia.com/jetson/l4t/Tegra%20Linux%20Driver%20Package%20Development%20Guide/power_management_jetson_xavier.html#wwpID0E0VO0HA. Kontsultatutako bertsioa: 32.5.

[83] Nvidia. Jetpack SDK. https://developer.nvidia.com/embedded/jetpack.

[84] Linux4Tegra development guide. https://docs.nvidia.com/jetson/l4t/index.html#page/TegraLinuxDriverPackageDevelopmentGuide/introduction.html. Kontsultatutako bertsioa: *Release 32.5*, 2021/02/02.

[85] Nvidia. Nvidia cuDNN developer guide. https://docs.nvidia.com/deeplearning/cudnn/pdf/cuDNN-Developer-Guide.pdf. Kontsultatutako azken bertsioa: v8.2.0, 2021/04.

[86] Siddhart Sharma Houman Abbasian, Josh Park and Sirisha Rella. Speeding up deep learning inference using TensorRT. https://developer.nvidia.com/blog/speeding-up-deep-learning-inference-using-tensorrt/. 2020/04/21.

[87] Nvidia. https://docs.nvidia.com/deeplearning/tensorrt/pdf/TensorRT-Developer-Guide.pdf. Kontsultatutako bertsioa: v7.2.3, 2021/02.

[88] OpenCV. https://opencv.org/.

[89] ONNX, open standard for machine learning interoperability. https://github.com/onnx/onnx.

[90] Nvidia. TensorRT release 7.x.x. https://docs.nvidia.com/deeplearning/tensorrt/release-notes/tensorrt-7.html#rel_7-2-3.

[91] JK Jung. TensorRT demos. https://github.com/jkjung-avt/tensorrt_demos/tree/master/yolo.

[92] Creating a ONNX network using helper functions. https://github.com/onnx/onnx/blob/master/docs/PythonAPIOverview.md#creating-an-onnx-model-using-helper-functions.

[93] Netron. https://netron.app/.

[94] Inside Volta: The world's most advanced data center GPU. https://developer.nvidia.com/blog/inside-volta/.

[95] Nvidia. CUDA for Tegra: Application note. https://docs.nvidia.com/cuda/pdf/CUDA-for-Tegra-AppNote.pdf. Kontsultatutako bertsioa: v11.3, 2021/04.

[96] GitHub/Tianxiaomo. Pytorch-YOLOv4. https://github.com/Tianxiaomo/pytorch-YOLOv4. Erabilitako bertsioaren azken eguneratzea: 2020/09/02.

[97] Wang Xinyu. TensorRTx. https://github.com/wang-xinyu/tensorrtx/.

[98] Ryan Olson Allison Gray, Chris Gottbrath and Shashank Prasanna. Deploying deep learning Nvidia TensorRT. https://developer.nvidia.com/blog/deploying-deep-learning-nvidia-tensorrt/.

[99] Nvidia. DLA supported layers. https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#dla_layers.

[100] Rafael Padilla, Wesley Lobato Passos, Thadeu Dias, Sergio Netto, and Eduardo da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10:279–306, 01 2021.

[101] Selene Consortium. Self-monitored dependable platform for high-performance safety-critical systems. https://www.selene-project.eu/.

[102] Fractal project. https://fractal-project.eu/.

[103] Nvidia. Jetson Roadmap. https://developer.nvidia.com/embedded/develop/roadmap. 2021Q1 bertsioa.

[104] Dr. P. Connor and Prof. F. Schmid. Train protection. *Railway Technical*.

[105] Crossco. Determining safety integrity levels for your process application. https://www.crossco.com/resources/articles/determining-safety-integrity-levels-for-your-process-application/.

[106] European Commission. ETCS levels and modes. https://ec.europa.eu/transport/modes/rail/ertms/etcs-levels-and-modes_en.

[107] European Commission. ERTMS - how does it work? https://ec.europa.eu/transport/modes/rail/ertms/how-does-it-work_en.

[108] Shift2Rail. S2R initiative. https://shift2rail.org.

[109] Shift2Rail. Innovation in the spotlight: Towards unattended mainlain train operations (ATO GoA4). https://shift2rail.org/highlight/innovation-in-the-spotlight-towards-unattended-mainline-train-operations-ato-goa4/.

[110] 1963: train drives iteself. http://news.bbc.co.uk/onthisday/hi/dates/stories/march/21/newsid_2546000/2546071.stm. BBCren hemeroteka.

[111] Klive Kessell. Main Line ATO becomes a reality. *Rail Engineer UK*.

[112] Pavel Popov. Developing and deploying Automatic Train Operations in Russia. *Global Railway Review*.

[113] Keith Borrow. Automatic Train Operations takes to the main line. *Rail Journal*.

[114] Julianna Delua. Supervised vs unsupervised learning. https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning. 2021/03/12.

[115] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. Proceedings of Machine Learning Research, New York, New York, USA, 20–22 Jun 2016. PMLR. Introduction, Preliminaries eta Tasks atalak.

[116] Microsoft Research Leon Bottou. Stochastic gradient descent tricks. https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/tricks-2012.pdf.

[117] Razvan Andonie and A. Florea. Weighted random search for CNN hyperparameter optimization. *ArXiv*, abs/2003.13300, 2020.