Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nik Prinčič

Vizualno sledenje objektov na vgrajenih napravah

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Luka Čehovin Zajc

Ljubljana, 2023

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Kandidat: Nik Prinčič
Naslov: Vizualno sledenje na vgrajenih napravah
Vrsta naloge: Diplomska naloga na visokošolskem programu prve stopnje Računalništvo in informatika
Mentor: doc. dr. Luka Čehovin Zajc

Opis:
V okviru projekta izberite primeren sledilnik, ki je bil predlagan v zadnjih letih (idealno tak, ki deluje na podlagi globokega učenja) in ga preoblikujte za uporabo na napravah v sklopu vgrajene platforme DepthAI. Implementacijo primerjajte z originalno verzijo kar se tiče hitrost ter uspešnosti.

Title: Visual tracking on embedded devices

Description:
For this project select a suitable visual tracking algorithm that was presented in the last years (ideally a method that is based on deep learning) and adapt it so that it can be run on the DepthAI platform devices. Compare the adapted implementation with the original version in terms of speed and tracking performance.

Zahvaljujem se mentorju, doc. dr. Luku Čehovinu Zajcu, za izjemno odzivnost na vsa moja vprašanja ter vso pomoč pri izdelavi diplomskega dela. Zahvaljujem se družini, ki me je podpirala in spodbujala med izdelavo diplomskega dela in v času celotnega študija.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Pregled področja 2

1.2 Struktura diplomske naloge 5

2 Metodologija **7**

2.1 Umetne nevronske mreže 7

2.2 Konvolucijske nevronske mreže 8

2.3 Arhitektura transformer 9

2.4 Model STARK 13

3 Implementacija **19**

3.1 Luxonis OAK-1 19

3.2 DepthAI 20

3.3 OpenVINO 21

3.4 Uporabljene tehnologije 22

3.5 Prilagoditev modela 23

3.6 Prevajanje modela 25

4 Evalvacija **33**

4.1 Hitrost delovanja 34

4.2 Platforma 34

4.3 Primerjava verzij metode 35

5 Zaključek **39**

Celotna literatura **41**

Seznam uporabljenih kratic

kratica	angleško	slovensko
AI	Artificial inteligence	Umetna inteligenca
ANN	Artificial neural network	Umetna nevronska mreža
BNN	Biological neural network	Biološka nevronska mreža
DNN	Deep Neural Network	Globoka nevronska mreža
CNN	Convolutional neural network	Konvolucijska nevronska mreža
FPS	Frames per second	Sličice na sekundo
OAK	OpenCV AI Kit	OpenCV AI komplet
OAK	OpenCV AI Kit	OpenCV AI komplet
BBOX	Bounding box	Omejitveni okvir
IoT	Internet of things	Internet stvari
SOT	Single object tracking	Sledenje posameznega objekta
MOT	Multiple object tracking	Sledenje več objektom
VOT	Visual object tracking	Vizualno sledenje
VPU	Visual processing unit	Vizualna procesna enota
ROI	Region of interest	Območje interesa
USB	Universal serial bus	Univerzalno serijsko vodilo
PoE	Power over ethernet	Napajanje preko etherneta

Povzetek

Naslov: Vizualno sledenje objektov na vgrajenih napravah

Avtor: Nik Prinčič

V okviru diplomskega dela je bilo implementirano in ovrednoteno delovanje vizualnega sledilnika na vgrajeni napravi Luxonis OAK-1. Izbran je bil sledilnik STARK, ki spada v družino sledilnikov, ki jih sestavljajo globoke nevronske mreže. Bolj specifično sledilnik uporablja arhitekturo transformer, ki je trenutno uporabljena v vseh najboljših vizualnih sledilnikih. Sledilnik je bilo potrebno rahlo predelati ter ga prevesti v OpenVINO format, ki omogoča uporabo na vgrajeni napravi. Poleg tega je bilo potrebno zasnovati cevovod, po katerem se podatki na napravi pretakajo. Z vsem naštetim smo dosegli, da lahko vgrajena naprava izvaja vse potrebne funkcije popolnoma avtonomno, vse kar potrebuje od gostiteljskega sistema (npr. osebni računalnik) je začetni omejitveni okvir tarče *(ang. bounding box)*, gostiteljskemu sistemu pa vrača vse naslednje omejitvene okvirje tarče. S tem smo dosegli, da so performance sledenja neodvisne od gostiteljskega sistema.

Ključne besede: računalniški vid na vgrajenih napravah, DepthAI, vizualni sledilnik.

Abstract

Title: Visual object tracking on embedded devices

Author: Nik Prinčič

In this diploma thesis, a state-of-the-art visual tracker is implemented and evaluated on the embedded device Luxonis OAK-1. The tracker STARK is chosen, which belongs to the family of deep neural network-based trackers. More specifically the tracker uses the novel transformer neural network architecture, which is making its way into increasingly more best performing visual trackers. During the implementation process we had to properly modify the tracker, convert it to the OpenVINO format, which can be used for inference on the embedded device. To run the processing completely on the embedded device the correct pipeline architecture was also developed, which allows for all the processing to be executed on the embedded device and consequently allows autonomous operation of the embedded device. The host system only needs to provide an initial bounding box, after which all the future predicted bounding boxes are transferred from the embedded device to the host system. With this level of autonomy, we successfully decoupled tracking performance from the performance of the host system.

Keywords: embedded computer vision, DepthAI, visual tracker.

Poglavje 1

Uvod

Računalniški vid je področje, ki se v zadnjih letih zelo hitro razvija. Z napredkom v razvoju avtonomnih sistemov, kot so avtonomni avtomobili, droni, roboti in še mnogi drugi ter vse večjem številu IoT naprav opremljenih s kamero, se vedno bolj pojavlja želja po uporabi modernih pristopov, ki omogočajo sledenje v realnem času, na majhnih, manj zmogljivih in energetsko varčnih napravah. Področje računalniškega vida zavzema kar nekaj sklopov, v tem delu smo se osredotočili na vizualno sledenje *(ang. visual object tracking, VOT)*.

Vizualno sledenje je področje, ki se ukvarja z iskanjem in sledenjem objektom v videu. V tem delu smo se osredotočili na sledenje v realnem času, pri kateremu sledilnik sledi samo eni tarči. Sledilniku je naprej potrebno podati začetni omejevalni okvir tarče *(ang. bounding box)*, ki ji želimo slediti, nato pa sledilnik na podlagi prostorske, pri najbolj modernih pa celo časovno-prostorske informacije, sledi želeni tarči in nam za vsako naslednjo sličico vrne pripadajoči omejevalni okvir. V preteklosti so bili najbolj popularni tako imenovani klasični algoritmi (npr. KCF [8], MOSSE [2] ...). Ti algoritmi so v grobem operirali zgolj na nizko nivojskih lastnostih, kot so barva, histogrami in gradienti ter s pomočjo različnih hevristik in obdelav napovedovali pozicijo tarče. Danes pa prevladujejo sledilniki, ki temeljijo na nevronskih mrežah. Uporaba nevronskih mrež je omogočila operiranje nad bolj

kompleksnimi globokimi opisniki *(ang. deep features)*. V zadnjih nekaj letih je upravičeno vedno bolj popularna arhitektura transformer [17], ki je bila primarno razvita in uporabljena za razumevanje in generacijo teksta *(ang. natural language processing, NLP)*, v zadnjih letih pa je bila adaptirana na vizualne sledilnike, kjer dosega odlične rezultate.

Da bi lahko zagotovili dobre performance pri manjši porabi energije, se je začel razvoj namenske strojne opreme za izvajanje umetnih nevronskih mrež in strojno pospeševanje operacij nad slikovnimi tokovi. Primer takšne specializirane strojne opreme je tudi kategorija procesorskih enot VPU *(ang. vision processing unit)*. V to družino procesorskih enot spada tudi čip, Intel Movidius Myriad X[1], ki je v osrčju uporabljene naprave v tem diplomskem delu.

V okviru diplomske naloge smo se osredotočili na sledilnik STARK [19] ter vgrajeno napravo Luxonis OAK-1[2]. Cilj naloge je bilo sledilnik prilagoditi uporabi na tej napravi, ga prevesti v potreben format in ga umestiti v cevovod. Cevovod je bilo potrebno tudi smiselno zasnovati, da v model pridejo pravilno oblikovani podatki.

1.1 Pregled področja

Na temo vizualnega sledenja vsako leto izide mnogo del. Skozi leta se neprestano spreminjajo pristopi, ki prinašajo najboljše rezultate. Sledilnike lahko delimo na t.i. klasične algoritme ter sledilnike, ki delujejo na osnovi strojnega učenja in umetnih nevronskih mrež. Leta 2010 so v članku [2] prvič predstavili uporabo korelacijskih filtrov v namen vizualnega sledenja. Kasneje, leta 2014, so v delu [8] predstavili uporabo kerneliziranih korelacijskih filtrov. Prednost klasičnih algoritmov je v tem, da niso računsko potratni in za delovanje v realnem času ne potrebujejo posebne strojne opreme. Njihova slabost pa je, v primerjavi s kompleksnejšimi modernimi pristopi, relativno slaba ro-

[1] `https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu/movidius-myriad-x.html`

[2] `https://shop.luxonis.com/products/oak-1`

bustnost. Eden izmed razlogov za slabšo natančnost je uporaba preprostih lastnosti slike, kot so barva, histogrami in gradienti. Leta 2015 so se začeli pojavljati sledilniki, ki so za učenje značilk uporabili konvolucijske nevronske mreže *(ang. convolutional neural network, CNN)* in dobljene značilke uporabili kot vhodne podatke za korelacijske filtre. Kot primer takšnega sledilnika lahko izpostavimo sledilnik CFCF, ki je bil predstavljen v članku [6]. Uporaba konvolucijskih nevronskih mrež je omogočila operiranje nad globokimi opisniki, ki nosijo precej več informacije kot pa nizkonivojske lastnosti, ki so bile uporabljene v preteklosti. Leta 2018 so se začeli pojavljati sledilniki na podlagi Siamskih nevronskih mrež *(ang. Siamese neural network)*. Primer takega sledilnika je RPT [14]. Leta 2021 pa so se pojavili sledilniki, ki temeljijo na arhitekturi umetnih nevronskih mrež poimenovani transformer. Takšno arhitekturo uporablja tudi sledilnik, ki smo ga uporabili v tem diplomskem delu, STARK [19]. Prednost sledilnikov, ki uporabljajo principe strojnega učenja, je boljša učinkovitost pri sledenju, slabost pa precej višja računska zahtevnost. Zaradi višje računske zahtevnosti je za delovanje v realnem času pogosto potrebna uporaba posebne strojne opreme, ki omogoča visok nivo paralelizacije. Med tako strojno opremo spadajo grafične procesorske enote, ali pa namensko razvite procesorske enote, ki med drugim pogosto podpirajo tudi strojno pohitritev najbolj pogostih operaciji.

Različne trende in uspešnosti sledilnikov najbolje razberemo, če pregledamo rezultate izziva VOT *(ang. VOT challenge)* [9], ki je eden najbolj uveljavljenih testov na področju vizualnega sledenja. VOT izziv vsako leto priredi evalvacijo novih sledilnikov. Sledilnike so v zadnji izvedbi, VOT2022 [11], testirali v sedmih različnih kategorijah. Glede na rezultate izziva v zadnjih dveh letih lahko opazimo porast v popularnosti in uspešnosti sledilnikov, ki uporabljajo arhitekturo transformer. Iz rezultatov VOT2022, objavljenih v [11] lahko razberemo, da devet od desetih najboljših sledilnikov uporablja arhitekturo transformer, opazimo lahko tudi, da je kar 47% vseh testiranih sledilnikov uporabilo to arhitekturo.

Na področju vizualnega sledenja na vgrajenih napravah je bilo objavljenih

že nekaj del, a nobeno od njih ni uporabilo sledilnika na osnovi arhitekture transformer. V članku *Evaluation of Visual Tracking Algorithms for Embedded Devices* [12] so primerjali performance med petimi klasičnimi algoritmi, teste so izvedli na napravi Raspberry Pi 3 B v1.2 in ugotovili, da z uporabo sledilnika KCF dobijo najboljše razmerje med hitrostjo in natančnostjo. V članku *Real-Time Multiple Object Visual Tracking for Embedded GPU Systems* [5] so predstavili MOT *(ang. multiple object tracking)* na napravi Nvidia Jetson TX2, kjer so uporabili večstopenjsko arhitekturo, detektor (YOLOv3 [16]) in sledilnik (KCF [8]). Sledilnik KCF so dodatno pralelizirali in optimizirali za procesorsko arhitekturo ARM. Njihova implementacija pa je pri sledenju do 45 objektov naenkrat dosegla povprečno 25.2 FPS in za 5.2 do 8.8 točk MOTA *(ang. multiple object tracking accuracy)* boljše rezultate od prejšnjih MOT implementacij na vgrajenih napravah. Vse to pa so dosegli pri maksimalni porabi do 15 W.

Tako kot hitro napreduje razvoj novih metod za uporabo v vizualnih sledilnikih, napreduje tudi razvoj strojne opreme. Zaradi vse večje popularnosti metod strojnega učenja na splošno se tudi razvoj strojne opreme vedno bolj usmerja v podporo teh metod. Vodilna podjetja tega področja so NVIDIA, Intel in Google. Google je na tem področju najbolj poznan po razvoju TPU[3] *(ang. tensor processing unit)*, ki jo uporabljajo v svojih serverskih poljih. NVIDIA je največji proizvajalec grafičnih procesorskih enot. Večina njihovih grafičnih procesorskih enot podpira pospešeno izvajanje nevronskih mrež, največji doprinos pa prinašajo izdelki iz družine Tesla[4]. Gre za splošno namenske grafične procesorske enote, ki vsebujejo velike količine spomina in veliko število procesorskih jeder CUDA. Poleg grafičnih procesorskih enot pa NVIDIA ponuja tudi vgrajene naprave iz družine Jetson[5]. V podjetju Intel pa poleg splošno namenskih procesorskih enot razvijajo tudi namenske procesorske enote za vizualno procesiranje - VPU *(ang. vision processing unit)*.

[3] `https://cloud.google.com/tpu/docs/system-architecture-tpu-vm`

[4] `https://www.nvidia.com/en-us/data-center/`

[5] `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/`

Eno izmed takih procesorskih enot, Intel Movidius Myriad X[6], uporablja tudi naprava, ki smo jo uporabili v tem delu. Naše delo področju doprinaša predstavitev možnosti uporabe najmodernejših pristopov na področju vizualnega sledenja na namenski vgrajeni napravi. Natančneje predstavlja možnost uporabe sledilnika, ki uporablja arhitekturo umetnih nevronskih mrež poimenovano transformer in čim višje izkoriščanje namenske procesorske enote z dobro implementiranim cevovodom, ki skoraj v celoti razbremeni splošno namenska precesorska jedra.

1.2 Struktura diplomske naloge

Diplomsko delo je razdeljeno na pet poglavij. V Poglavju 2 bomo opisali metodologijo, kar zavzema hiter opis nevronskih mrež na splošno, arhitekture CNN in transformer ter predstavitev izbranega sledilnika. V Poglavju 3 bomo opisali podrobnosti implementacije, kar vključuje opis uporabljene vgrajene naprave, prilagoditev modela ter njegovo pretvorbo v pravilen format, opis implementacije cevovoda in opis štirih različnih načinov delovanja, ki smo jih pripravili. V Poglavju 4 bomo predstavili rezultate evalvacije, osredotočili se bomo na primerjavo med sledilnikom, ki je bil pognan samostojno na vgrajeni napravi proti tistemu, ki je pognan na osebnem računalniku. Predstavili bomo tudi rezultate primerjave med dvema načinoma delovanja, robni *(ang. edge mode)* proti gostiteljskemu *(ang. host mode)* načinu delovanja. V Poglavju 5 je zaključek, ki povzema ugotovitve ter predstavi možne izboljšave za nadaljnji razvoj.

[6] `https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu/movidius-myriad-x.html`

Poglavje 2

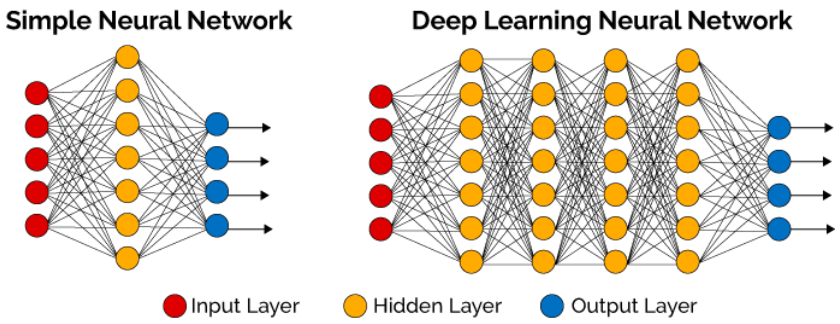
Metodologija

Ker je bil cilj dela preizkusiti možnost uporabe enega izmed najsodobnejših vizualnih sledilnikov na vgrajeni napravi, ki temelji na nevronskih mrežah, bomo v tem poglavju predstavili delovanje nevronskih mrež, arhitekture CNN in transformer ter opisali izbrani sledilnik.

2.1 Umetne nevronske mreže

Umetne nevronske mreže *(ang. Artificial Neural Network, ANN)* so vrsta modelov strojnega učenja, ki posnemajo delovanje bioloških nevronskih mrež *(ang. Biological Neural Network, BNN)*. Sestavljene so iz množice umetnih nevronov, ki so med seboj povezani z uteženimi povezavami in združeni v sloje. Sloje delimo na tri vrste - vhodni sloj *(ang. input layer)*, skrite sloje *(ang. hidden layers)* in izhodni sloj *(ang. output layer)*. Mreže z več kot enim skritim slojem imenujemo globoke nevronske mreže *(ang. Deep Neural Network, DNN)*, ostale pa smatramo kot plitve nevronske mreže. Na Sliki 2.1 je prikazana primerjava zgradbe med plitvo nevronsko mrežo (levo) in globoko nevronsko mrežo (desno).

Najpogosteje se za učenje nevronskih mrež uporablja algoritem vzvratnega razširjanja *(ang. backpropagation)*. Algoritem se izvaja v več iteracijah, pri katerih se s pomočjo kriterijske funkcije izračuna napaka, ki je



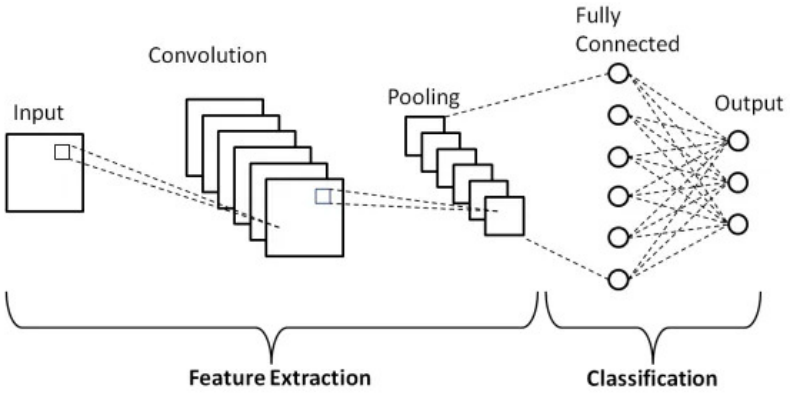
Slika 2.1: Primerjava zgradbe NN in DNN [15].

razlika med želenim izhodom in dejanskim izhodom. Napako uporabimo za izračun gradienta, ki nam pove, kako moramo spremeniti vrednost uteži, da se napaka zmanjša. Kriterijsko funkcijo moramo pravilno izbrati glede na vrsto problema, ki ga želimo rešiti. Za reševanje regresijskih problemov se najpogosteje uporablja srednjo kvadratno napako *(ang. Mean Squared Error, MSE)*, za klasifikacijske probleme pa najpogosteje kategorično križno entropijo *(ang. Categorical Cross-Entropy)* ali pa binarno križno entropijo *(ang. Binary Cross-Entropy)*. Za iskanje optimalnih vrednosti uteži se uporabljajo različni optimizacijski algoritmi, ki na podlagi gradientnega spusta *(ang. gradient descent)* iščejo minimum kriterijske funkcije. Med njimi sta najbolj znan stohastični gradientni spust *(ang. Stochastic Gradient Descent, SGD)* in Adam *(ang. Adaptive Moment Estimation)*.

2.2 Konvolucijske nevronske mreže

Konvolucijske nevronske mreže *(ang. Convolutional Neural Network)* so vrsta umetnih nevronski mrež, ki se pogosto uporablja v nalogah računalniškega vida, kot so prepoznavanje objektov, klasifikacija slik, detekcija obrazov in drugo. CNN modeli so tipično sestavljeni iz več ponovitev konvolucijskih slojev *(ang. convolutional layers)* in združevalnih slojev *(ang. pooling layers)*. Za zadnjim združevalnim slojem najpogosteje sledi nekaj polno-povezanih slojev. Poenostavljena arhitektura konvolucijske nevronske mreže je prika-

zana na Sliki 2.2.



Slika 2.2: Poenostavljena arhitektura konvolucijske nevronske mreže [1].

Namen konvolucijskih slojev je pridobivanje značilk *(ang. feature extraction)*, namen združevalnih slojev je zmanjševanje dimenzionalnosti podatkov, polno-povezani sloji pa so namenjeni preslikovanju pridobljenih značilk v končni izhod.

Konvolucijski sloji delujejo na principu matematične operacije konvolucije, ki je definirana na dveh funkcijah. Rezultat konvolucije nam pove, kako oblika ene spremeni obliko druge. Definirana je z enačbo:

$$(f * g)(t) = \int_{\tau=-\inf}^{\inf} f(\tau)g(t-\tau)d\tau \tag{2.1}$$

Kjer je f predstavlja vhodno funkcijo, g pa jedro konvolucije. Ker se pri obdelavi digitalnih podatkov pogosto uporablja diskretna konvolucija, z Enačbo 2.1 spremenjena v:

$$(f * g)[t] = \sum_{k=-\inf}^{\inf} f[k]g[n-k] \tag{2.2}$$

2.3 Arhitektura transformer

Transformer je arhitektura nevronske mreže, ki temelji na mehanizmu pozornosti, ki je bil predstavljen leta 2017 v članku *Attention is all you need*

[17]. Primarno je bila arhitektura razvita za naloge procesiranja naravnega jezika *(Natural language processing, NLP)*, sedaj pa postaja popularna tudi v drugih domenah strojnega učenja, med drugim tudi v računalniškem vidu. Osnovna arhitektura, ki je bila predstavljena v [17], vključuje kodirni *(ang. encoder)* in dekodirni modul *(ang. decoder)*. Kodirnik je sestavljen iz dveh podslojev, dekodirnik pa iz treh.

2.3.1 Mehanizem pozornosti

Mehanizem pozornosti deluje na podlagi ključev *(ang. key, K)*, poizvedb *(ang. query, Q)* in vrednosti *(ang. value, V)*. Mehanizem iz matrike ključev in matrike poizvedb izračuna matriko pozornosti. Z matričnim množenjem matrike pozornosti in matrike vrednosti dobimo linearno kombinacijo vrednosti, ki predstavljajo izhod. Razlikujemo med samo-pozornostjo in med-pozornostjo. O samo-pozornosti govorimo, ko vse tri vhodne parametre dobimo iz iste množice podatkov, pri med-pozornosti pa poizvedbe pridobimo iz ene množice podatkov, ključe in vrednosti pa iz druge.

Vhodne vektorje $x_i, x_{i+1}, ..., x_n$ združimo v matriko $X_{n \times d_m}$, kjer d_m predstavlja dimenzionalnost modela. Naučene parametre pa predstavljajo matrike $W^Q_{d_m \times d_m}$, $W^K_{d_m \times d_m}$ in $W^V_{n \times d_m}$. Iz navedenih matrik lahko izračunamo

$$\begin{aligned} Q &= XW^Q, \\ K &= XW^K, \\ V &= XW^V, \end{aligned} \tag{2.3}$$

Matrika pozornosti je definirana z enačbo:

$$A(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_m}})V \tag{2.4}$$

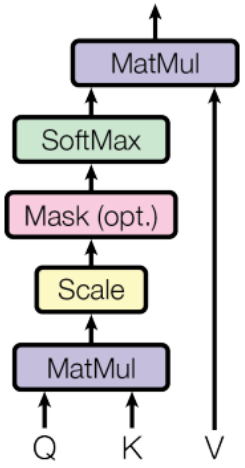
kjer funkcija *softmax* spremeni vektor n realnih števil v vektor verjetnostne porazdelitve.

Iz 2.3 in 2.4 lahko izračunamo končno izhodno vrednost mehanizma.

$$S = AV, \tag{2.5}$$

Na Sliki 2.3 je potek izračuna prikazan v obliki diagrama.

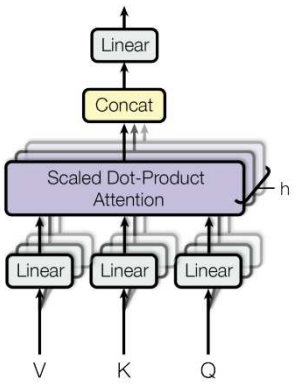
Ko se mehanizem pozornosti uporabi v dekodirnem modulu, je potrebno določen del podatkov skriti. Bolj natančno, modelu je treba preprečiti interakcijo s podatki, ki jih še ni napovedal.



Slika 2.3: Diagram izračuna pozornosti [17].

2.3.2 Več-glava pozornost

Da bi model lahko zajel več lastnosti vhodnih podatkov, je potrebno mehanizem pozornosti nadgraditi. Več-glava pozornost vzporedno izračuna več ločenih pozornosti (glav) in jih združi v končni rezultat. Vsaka glava se osredotoči na eno lastnost podatkov. Vsaka glava i ima svoje matrike naučenih uteži, končni rezultat pa se pridobi s strnitvijo posameznih matrik pozornosti. Na Sliki 2.4 je prikazan diagram poteka več-glave pozornosti.



Slika 2.4: Diagram izračuna več-glave pozornosti [17].

2.3.3 Vhodna vdelava in pozicijsko kodiranje

Preden vhodni podatki prispejo do kodirnega bloka, jih je potrebno najprej pravilno predelati. Za ta namen se uporablja vdelava vhodnega niza v d_m dimenzionalni latentni prostor *(ang. embedding space)*. Vhodna vdelava *(ang. input embedding)* vhodne besede pretvori v vektorje, s tem model pridobi informacijo o pomenu posamezne besede. Pozicijsko kodiranje *(ang. positional encoding)* pa modelu poda informacijo o vrstnem redu besed v nizu. Da bi se izognili velikim vrednostim v pozicijskem kodiranju, se za kodiranje uporabljata Enačbi 2.6.

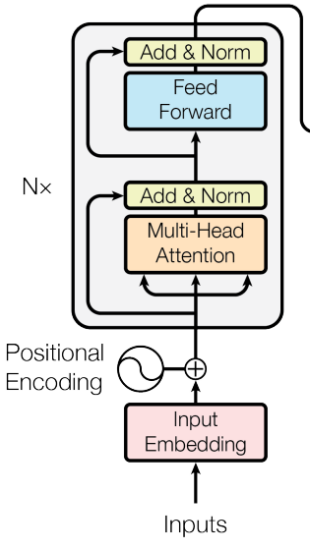
$$\begin{aligned} PE(pos, 2i) &= sin(pos/10000^{2i/d_m}) \\ PE(pos, 2i+1) &= cos(pos/10000^{2i/d_m}) \end{aligned} \tag{2.6}$$

V Enačbi 2.6 predstavlja *pos* pozicijo besede v nizu, i pa dimenzijo kodiranja, kar pomeni, da ima vsaka dimenzija pozicijskega kodiranja pripadajočo sinusoidno vrednost.

Informacijo o pomenu in poziciji posamezne besede združimo tako, da matriki seštejemo.

2.3.4 Kodirni modul

Kodirni modul ali kodirnik je sestavljen iz N identičnih slojev, ki so sestavljeni iz dveh podslojev. Prvi podsloj je več glava pozornost *(ang. multi-headed attention)* in polno povezane usmerjene nevronske mreže *(ang. Feedforward neural network, FFN)*.



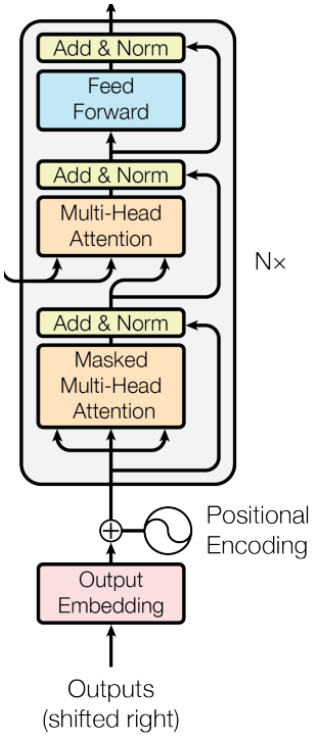
Slika 2.5: Diagram kodrinega modula [17].

2.3.5 Dekodirni modul

Dekodirni modul ali dekodirnik je sestavljen iz N identičnih slojev, ki so sestavljeni iz treh podslojev. Prvi podsloj je več glava pozornost z možnostjo maskiranja vhodnih podatkov. Drugi podsloj je več glava pozornost, tretji sloj pa predstavlja polno povezana nevronska mreža.

2.4 Model STARK

Model STARK [19] spada v družino SOT *(ang. single object tracking)* sledilnikov. Primarno je sestavljen iz dveh arhitektur, konvolucijskih nevronskih



Slika 2.6: Diagram dekodirnega modula [17].

mrež in arhitekture transformer, ki je bila prilagojena za vizualne sledilnike. Navdih za njegov nastanek je bil predhodni model za detekcijo DETER [3]. Ena od novosti, ki so jo uvedli v tem modelu, je uporaba časovne in prostorske komponente *STARK_ST*. Prostorska komponenta vsebuje informacijo o izgledu objekta, ki mu sledi. Časovna komponenta pa nosi informacijo o spremembi pozicije objekta skozi čas. Arhitektura, ki so jo predlagali, vsebuje tri ključne elemente: kodirni modul, dekodirni modul in napovedovalno glavo *(ang. predictio head)*. Model kot vhod prejme trenutno sliko, začetno predlogo *(ang. template)* in dinamično predlogo, ki se skozi čas dinamično posodablja. Z uporabo dinamične predloge, ki se skozi čas posodablja, lahko model zajame prostorsko in časovno informacijo o objektu, ki mu sledimo.

Prednost tega sledilnika je v tem, da ne potrebuje kompleksne predobdelave *(ang. preprocessing)* vhodnih podatkov in naknadne obdelave *(ang.*

postprocessing) izhoda. Ob inicializaciji prejme kot vhod sliko in omejitveni okvir tarče. Na podlagi teh dveh vhodnih podatkov se najprej iz celotne vhodne slike izreže iskalno območje *(ang. search area)*, ki se uporabi za izračun predloge. Ob vsaki naslednji sličici pa so vhodni podatki iskalno območje izrezano iz vhodne slike, začetna predloga in dinamična predloga, sledilnik pa vrne izračunan omejitveni okvir.

2.4.1 Arhitektura modela STARK Lightning

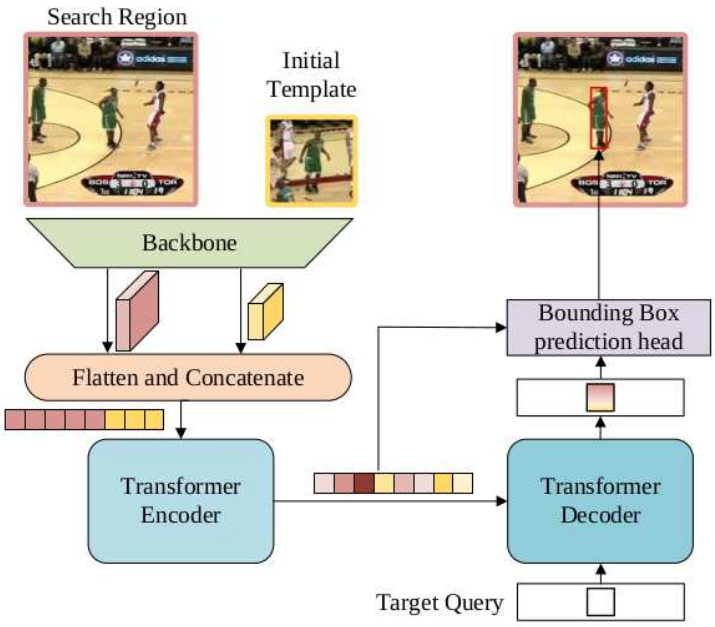
V tem delu smo zaradi manjše procesorske in prostorske zahtevnosti uporabili različico modela STARK Lightning[1]. Gre za poenostavljeno različico modela STARK, ki je dodatno optimizirana za manjšo porabo spomina in procesorske moči. Poleg vseh optimizacij pa ta različica modela uporablja samo prostorsko informacijo. Glavne spremembe med osnovnim in optimiziranim modelom so drugačna osnova konvolucijske hrbtenice, uporaba naučene pozicijske vdelave namesto sinusoidne vdelave in za polovico manjše število skritih slojev. Pričakovano ima ta različica slabšo natančnost pri sledenju. V nadaljevanju bomo predstavili delovanje in arhitekturo modela.

Arhitekturo modela lahko razdelimo na tri glavne dele: *(1)* konvolucijska hrbtenica *(ang. convolutional backbone)*, *(2)* kodirni-dekodirni transformer in glavo za napovedovanje omejitvenega okvirja *(ang. bounding box prediction head)*. V nadaljevanju bomo predstavili vsakega od teh delov.

Konvolucijska hrbtenica

Konvolucijsko hrbtenico sestavlja model RepVGG [4], ki so mu odstranili zadnjo sekcijo in polno povezane sloje. Pri osnovni različici modela STARK se za osnovo konvolucijske hrbtenice uporablja model ResNet-50 ali ResNet-100 [7]. Kot vhod prejme konvolucijska hrbtenica začetno predlogo $z \in \mathbb{R}^{3\times H_z\times W_z}$ in iskalno območje $x \in \mathbb{R}^{3\times H_x\times W_x}$. Po prehodu čez hrbtenico dobimo dve matriki značilk $f_z \in \mathbb{R}^{C\times \frac{H_z}{s}\times \frac{W_z}{s}}$ in $f_x \in \mathbb{R}^{C\times \frac{H_x}{s}\times \frac{W_x}{s}}$.

[1] `https://github.com/researchmm/Stark/blob/main/lib/tutorials/STARK_Lightning_En.md`



Slika 2.7: Diagram arhitekture modela STARK Lightning [17].

2.4.2 Kodirnik

Na matriki značilk, ki jo izračuna hrbtenica, je najprej potrebno izvesti predobdelavo. Predobdelava vključuje sloj z ozkim grlom *(ang. bottleneck layer)*, ki matrikam zmanjša prostorsko dimenzionalnost iz C na d. Nato je matriki potrebno sploščiti in združiti vzdolž prostorske dimenzije. Rezultat prejšnjih dveh operaciji proizvede sekvenco značilk dolžine $\frac{H_z}{s}\frac{W_z}{s} + \frac{H_x}{s}\frac{W_x}{s}$ in dimenzionalnosti d. Novo pridobljeni sekvenci značilk prištejemo vrednosti pozicijskega kodiranja vhoda. Ta sekvenca značilk je vhod za kodirni modul. Kodirni modul je sestavljen iz N enakih slojev. Sloji so sestavljeni iz več-glave samo-pozornosti in FFN. Kodirni modul zajame odvisnosti med vsemi značilkami v vhodni sekvenci in s tem omogoča modelu, da se nauči o diskirminativnih značilkah, ki se uporabijo za lokalizacijo objekta.

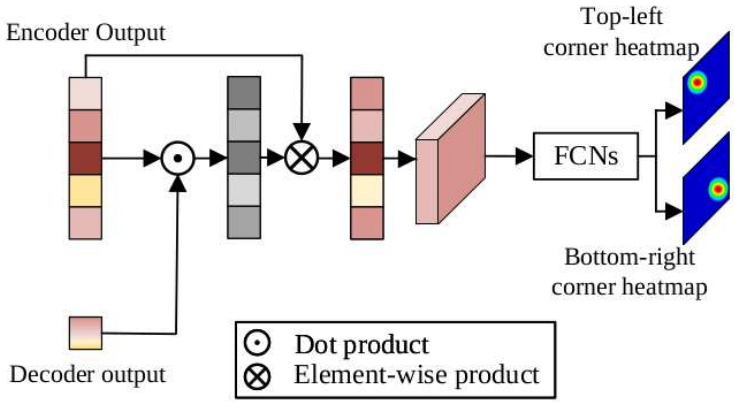
2.4.3 Dekodirnik

Dekodirnik kot vhod prejme sekvenco značilk, ki jo je izračunal kodirnik, in eno poizvedbo. Podobno kot kodirnik je tudi dekodirnik sestavljen iz M identičnih slojev. Vsak sloj je sestavljen iz samo-pozornosti, med-pozornosti kodirnik-dekodirnik in FFN. V sloju med-pozornosti lahko ciljna poizvedba deluje nad vsemi pozicijam na vhodni predlogi in iskalnem območju, kar omogoča učenje robustnih reprezentacij predikcij končnega omejevalnega okvirja.

2.4.4 Glava za napovedovanje omejevalnega okvirja

Model uporablja novo zasnovani sistem za napovedovanje omejevalnega okvirja, ki deluje na napovedovanju verjetnostne distribucije robov okvirja. Glava kot vhod prejme sekvenco značilk iskalnega območja, ki jih je izračunal kodirni modul in izhodno vdelavo *(ang. output embedding)*, ki jo je izračunal dekodirnik in med njimi izračuna podobnosti. Podobnosti pomnoži s sekvenco značilk iskalnega območja, s tem se poveča pomembnost pomembnih regij. Rezultat te operacije je nova sekvenca značilk, ki jo je potrebno preoblikovati v matriko $f \in \mathbb{R}^{d \times \frac{H_s}{s} \times \frac{W_s}{s}}$. Novo pridobljena matrika je posredovana v polno povezano konvolucijsko mrežo *(ang. fully connected convolutional network)*, ki iz matrike značilk izračuna dve verjetnostni matriki, $P_{tl}(x, y)$ in $P_{br}(x, y)$, ki predstavljata levi zgornji in desni spodnji kot omejevalnega okvirja. Končne koordinate omejevalnega okvirja $(\hat{x_{tl}}, \hat{y_{tl}})$ in $(\hat{x_{br}}, \hat{y_{br}})$. Na Sliki 2.8 je z diagramom predstavljen opisan potek napovedovanja omejevalnega okvirja, zadnji korak računanja kordinat iz verjetnostnih matrik pa je opisan v Enačbi 2.7.

$$\begin{aligned}(\hat{x_{tl}}, \hat{y_{tl}}) &= (\sum_{y=0}^{H}\sum_{x=0}^{W} x \cdot P_{tl}(x, y), (\sum_{y=0}^{H})\sum_{x=0}^{W} y \cdot P_{tl}(x, y)), \\ (\hat{x_{br}}, \hat{y_{br}}) &= (\sum_{y=0}^{H}\sum_{x=0}^{W} x \cdot P_{br}(x, y), (\sum_{y=0}^{H})\sum_{x=0}^{W} y \cdot P_{br}(x, y)).\end{aligned} \tag{2.7}$$



Slika 2.8: Diagram poteka izračuna omejitvenega okvirja [17].

Poglavje 3

Implementacija

V tem poglavju bomo najprej predstavili vgrajeno napravo Luxonis OAK-1, ogrodji OpenVINO in DepthAI, na kratko predstavili vsa ostala uporabljena orodja in tehnologije ter na koncu podrobno opisali postopek implementacije sledilnika na ciljni platformi.

3.1 Luxonis OAK-1

Luxonis je ameriško podjetje, ki se ukvarja z razvojem naprav za uporabo na področju prostorske umetne inteligence *(ang. spatial AI)* in računalniškega vida. Od ostalih se razlikujejo po odprtokodnost vseh svojih naprav in ogrodju *(ang. framework)* DepthAI. Ponujajo več različnih različic naprav, vsem pa je skupno to, da imajo integriran čip Intel Movidus MyriadX, ki ponuja relativno visoke performance, pri tem pa zavzame malo prostora in porabi malo energije. Njihove izdelke lahko na grobo razdelimo glede na dve karakteristiki. Glede na zmožnost zajemanja slike (mono ali stereo) in glede na način napajanja in komunikacije (USB ali Ethernet in PoE). V tem delu smo uporabili napravo OAK-1, ki omogoča zajem mono slike, napaja in komunicira pa preko USB-C. Naprava OAK-1 je prikazana na Sliki 3.1.

Slika 3.1: Slika prikazuje uporabljeno napravo Luxonis OAK-1 [13].

V osrčju naprave je modul RVC2[1] *(Robotic Vision Core 2).* Modul ponuja 4 TOPS procesorske moči, od katerih je 1.4 TOPS rezerviranih za izvajanje nevronskih mrež. Podpira pa tudi strojno kodiranje slikovnih tokov (H.264, H.265, MJPEG), pospešeno izvajanje pogostih operacij v računalniškem vidu (skaliranje, rezanje, zaznavanje robov itd.). V osrčju modula je Intelov sistem na čipu *(ang. system on chip, SoC)* Movidus Myriad X, ki vključuje Intelov NCE *(Neural compute engine)*[2], 16 vektorskih procesorskih enot SHAVE[3], 20 strojnih pospeševalnih enot poimenovanih *Enhanced Vision Accelerators* ter 2.5 MB vgrajenega hitrega spomina.

3.2 DepthAI

DepthAI[4] je hkrati programsko ogrodje *(ang. framework)* in tudi ekosistem odprtokodne programske in strojne opreme, ki ga razvija podjetje Luxonis. Ogrodje je na voljo v dveh izvedbah, ogrodje za programski jezik Python in izvedba za programski jezik C++. Ogrodje nam olajša uporabo naprav, saj ponuja programski vmesnik *(ang. Application Programming Interface, API)*, s katerim lahko dostopamo do resursov naprave. Princip delovanja stoji na

[1] `https://docs.luxonis.com/projects/hardware/en/latest/pages/BW1093.html#rvc2-inside`

[2] `https://www.intel.com/content/www/us/en/products/docs/processors/movidius-vpu/myriad-x-product-brief.html`

[3] `https://en.wikichip.org/wiki/movidius/microarchitectures/shave_v2.0`

[4] `https://github.com/luxonis/depthai`

cevovodni arhitekturi. Cevovod je sestavljen iz med-seboj povezanih vozlišč, ki se izvajajo na napravi. V ogrodju imamo na razpolago več različnih tipov vozlišč, spodaj je navedenih nekaj najbolj uporabljenih:

- vozlišče za manipuliranje slike *ImageManip*, ki nam omogoča enostavno manipulacijo slike (skaliranje, izrezovanje, pretvorba formatov, itd.),
- vozlišče za konfiguriranje in interakcijo s kamero *ColorCamera*,
- vozlišče za pretok podatkov preko USB povezave v smeri iz naprave na gostiteljski sistem *XLinkOut*,
- vozlišče za pretok podatkov preko USB povezave v smeri iz gostiteljskega sistema v napravo *XLinkIn*,
- vozlišče za izvajanje pomeri narejenih skript na napravi - *Script*. Skripte morajo biti napisane v programskem jeziku Python,
- vozlišče za uporabo pomeri narejenih nevronskih mrež na napravi - *NeuralNetwork*. Nevronske mreže morajo bit prevedene v pravilen format.

3.3 OpenVINO

OpenVINO[5] je komplet odprtokodnih orodji, ki nam omogočajo optimizacijo in prevajanje modelov nevronskih mrež v format, ki je primeren za delovanje na najrazličnejših napravah, med drugimi tudi VPU Intel Movidus Myriad X. Vsak model je potrebno najprej pravilno prilagoditi, da ustreza zahtevam in omejitvam ciljnega sistema.

Pri prilagajanju moramo biti pozorni na tipe slojev, ki so uporabljeni v modelu, saj niso vsi podprti na vseh napravah. Upoštevati je treba tudi omejitev, da model ne more več pomniti dinamičnega stanja. Predstavljamo si lahko, da model ni več kos programske opreme, temveč samo zaporedje matematičnih operacij, ki prejme vhod in vrne izhod.

[5] `https://github.com/openvinotoolkit/openvino`

Po prilagoditvi modela sledi korak optimizacije. Pri tem koraku se s pomočjo orodja *Model optimizer* izboljša računska in prostorska poraba modela. V tem koraku se poda tudi ciljni podatkovni tip uteži in ostalih fiksnih parametrov model, imena izhodnih in vhodnih podatkov ter dimenzionalnost vhodnih podatkov. Pri podajanju dimenzionalnosti vhodnih podatkov moramo upoštevati, da nekatere ciljne naprave ne podpiranjo dinamičnih velikosti.

Po optimizacijskem koraku sledi še zadnji korak, prevajanje. Model je potrebno prevesti v namensko obliko, primerno za izvrševanje na ciljni napravi. Pri prevajanju moramo podati tip končne naprave, podatkovni tip vhodnih podatkov in število vektorskih procesorjev SHAVE, ki jih bo model uporabil. Rezultat prevajanja je binarna datoteka v formatu *.blob*. Dobljena datoteka se kasneje preko Depthai API prenese na napravo in se tako lahko uporablja za izvajanje.

3.4 Uporabljene tehnologije

Uporabljen je bil že naučen model STARK [19], bolj specifično STARK Lightning. Gre za različico sledilnika, ki uporablja samo prostorsko informacijo, poleg tega pa se različica Lightning razlikuje še po tem, da porabi precej manj procesorske moči in prostora, seveda pa je posledica tega slabša natančnost. Različica je bila razvita za hitrejše izvajanje in izvajanje na manj zmogljivih platformah. Sledilnik je razdeljen na dve ločeni nevronski mreži, od sedaj naprej jih bomo poimenovali **initialization** in **tracking**. Mreža *initialization* se uporablja samo pri inicializaciji sledilnika, *tracking* pa pri korakih sledenja, več o posameznih nevronskih mrežah pa je opisano v Podpoglavju 3.5. V neoptimizirani obliki umetna nevronska mreža *tracking* uporablja 2.08 M parametrov in zahteva 1.46G MAC operacij, *initialization* pa uporablja 1.52 M parametrov in zahteva 197.98 M MAC operacij. Sledilnik so [19] implementirali v ogrodju PyTorch z uporabo programskega jezika Python, v katerem smo tudi mi nadaljevali z razvojem. Za lažji razvoj sta se upo-

rabili dve kontejnerski okolji Docker. Eno okolje je bilo namenjeno razvoju modela v programskem jeziku Python, drugo okolje pa je bilo uporabljeno za namestitev in uporabo kompleta orodji OpenVINO. Prednost uporabe kontejneriziranih okolij je prenosljivost in reproduciranje rezultatov.

3.5 Prilagoditev modela

Model pri delovanju potrebuje vhodno iskalno območje, ki je izrezano iz vhodne slike. Iskalno območje je fiksne dimenzije, če gre za inicializacijski korak, je to dimenzija 128×128, pri vseh nadaljnjih korakih pa je dimenzija iskalnega območja 320×320. Izrezovanje iskalnega območja deluje tako, da se iz vhodne slike izreže območje, ki ga omejuje omejevalni okvir, ki ga povečamo za določen faktor. Izračun faktorja C, ki predstavlja vrednost, za katero se vhodni omejevalni okvir poveča, poteka po Enačbi 3.1, kjer w predstavlja širino, h pa višino vhodnega omejevalnega okvirja. F predstavlja faktor iskanja. Pri inicializacijskem koraku ima F vrednost 2, pri vseh ostalih korakih pa 5.

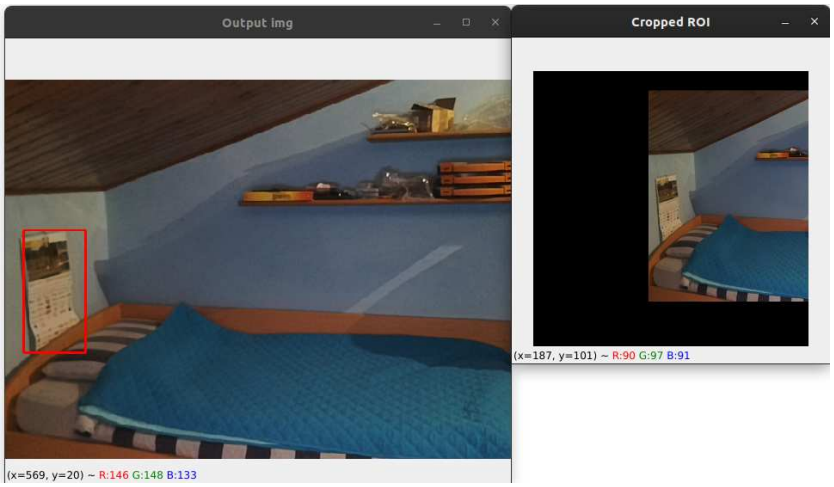
$$C = \lceil \sqrt{w \times h} \times F \rceil \tag{3.1}$$

Pri izrezovanju iskalnega območja je potrebno upoštevati, da je lahko iskalno območje pri robu slike, v tem primeru se sliki dodajo dodatni robovi *(ang. padding)* z vrednostjo 0. Da dodani robovi ne bi vplivali na končni rezultat, jih je potrebno maskirati, zato se poleg iskalnega območja pripravi tudi maska. Maska ima vrednost 1, kjer je bil dodan rob, na ostalih mestih pa ima vrednost 0. V začetni implementaciji je bilo izdelovanje maske umeščeno v predprocesiranje, ker pa si na vgrajeni napravi ne smemo privoščiti preveč obsežnega predprocesiranja, smo se odločili, da izdelovanje maske vključimo v sam model. Na začetek modela smo dodali dodaten modul, ki izračuna masko. Naj bo $X_{1\times3\times H\times W}$ vhodna iskalna regija, maska pa se izračuna po naslednjih korakih:

1. Izračunamo povprečje po 2. dimenziji matrike. Rezultat je matrika $M_{1\times1\times H\times W}$.

2. Za vsak piksel, pri katerem je povprečna vrednost enaka 0, nastavimo vrednost maske na 1, za vse ostale piksle pa nastavimo vrednost maske na 0.

3. Da izločimo morebitne napake, ki bi jih lahko ta pristop povzročil (predpostavimo, da obstaja neničelna verjetnost, da bo kamera proizvedla vrednost piksla, kjer bo povprečna vrednost 0), izvedemo še operacijo maksimalnega združevanja *(ang. max pooling)* z velikostjo okna 3×3 in vrednostjo 1.



Slika 3.2: Primer situacije, kjer je izrezanemu iskalnemu območju dodan rob.

Za enostavnejšo umestitev modela v cevovod je model razdeljen na 2 dela, *initialization* in *tracking*. *Initialization* zajema samo hrbtenico, sloj ozkega grla in pozicijsko kodiranje. Ta model se uporabi ob inicializaciji. *Tracking* pa vključuje celotno funkcionalnost modela STARK in se uporablja pri vsakem nadaljnjem koraku sledenja.

3.6 Prevajanje modela

Za lažjo pripravo delovnega okolja in programskega paketa OpenVINO je bilo uporabljeno kontejnersko okolje Docker.

Model je najprej potrebno iz ogrodja PyTorch izvoziti v format *ONNX*. *ONNX* je odprtokodni format za shranjevanje modelov, ki ga je ustvaril Microsoft. Za izvoz lahko uporabi funkcionalnost, ki je vgrajena v PyTorch. Pri izvozu moramo podati primer vhodnih podatkov ter poimenovati - labelirati vhodne in izhodne argumente. V Tabeli 3.1 so podani uporabljeni argumenti.

labela	**opis**	**dimenzije**	**pod. tip**
Model: *initialization*			
img	iskalno območje	$1 \times 3 \times 128 \times 128$	float16
Model: *tracking*			
img_x	iskalno območje	$1 \times 3 \times 320 \times 320$	float16
feat_z	sekveca značilk predloge	$64 \times 1 \times 128$	float16
mask_z	maska predloge	1×64	bool
pos_z	pozicijsko kodiranje predloge	$64 \times 1 \times 128$	float16

Tabela 3.1: Tabela prikazuje uporabljene argumente pri izvozu modelov v format *ONNX*.

Ko je model uspešno izvožen v format *ONNX*, ga lahko z orodjem iz paketa OpenVINO (*Model optimizer*) optimiziramo in predpripravimo za zadnji korak - prevajanje. *Model optimizer* je orodje, ki nam omogoča optimizacijo modelov. V našem primeru kot vhod prejme model v formatu *ONNX*. Podobno kot pri izvozu v format *ONNX* moramo tudi pri optimizaciji podati argumente, ki določajo imena in dimenzionalnosti vhodnih podatkov in imena izhodnih podatkov. Izhod optimizacijskega orodja sta dve datoteki. Prva je v formatu *.xml* in opisuje topologijo modela, druga pa v formatu *.bin* in vsebuje vrednosti uteži in parametrov modela.

V zadnjem koraku moramo model prevesti v binarni format, ki se lahko uporabi na procesorski enoti MyriadX. V ta namen se je uporabilo orodje

compile tool iz programskega paketa OpenVINO. Orodju podamo pot do *.xml* in *.bin* datotek, ki jih je ustvaril *Model optimizer* ter potrebne argumente. Argumenti, ki smo jih uporabili, so sledeči:

- **-d MYRIAD**, argument določa ciljno platformo, na kateri bo model uporabljen. V našem primeru je to procesorska enota Myriad X.
- **-ip U8** ali **-ip FP16**, argument določa podatkovni tip vhodnih podatkov. V našem primeru je to nepredznačeni 8-bitni celoštevilski tip ali pa 16-bitno število v plavajoči vejici.
- **-VPU_NUMBER_OF_SHAVES x**, argument določa število vektorskih procesorskih enot *SHAVE*, ki jih bo uporabljal model. Simbol x je pri prevajanju modela **tracking** imel vrednost 8, pri modelu **initialization** pa 1.

Nekompatibilnost podatkovnih tipov

Ena izmed omejitev OpenVINO okolja je, da morajo biti podatkovni tipi vseh vhodnih podatkov enaki. V našem primeru so slikovni podatki tipa UINT8, vsi ostali pa tipa FLOAT16. Problem smo rešili tako, da smo v cevovod pred vsako izmed dveh nevronskih mrež sledilnika dodali nevronsko mrežo, ki prejme podatke tipa UINT8, jih pretvori v FLOAT16 in jih pošlje naprej. Sledilnik je tako dobil vhodne podatke tipa FLOAT16, ki so bili ustrezno pretvorjeni iz vhodnih podatkov tipa UINT8.

3.6.1 DepthAI cevovod

Vso funkcionalnost, ki jo želimo izvajati na vgrajeni napravi, je potrebno umestiti v cevovod. Cevovod je sestavljen iz različnih vrst vozlišč, ki so med seboj povezana in tako tvorijo cevovod. V tem delu smo uporabil nasledje tipe vozlišč:

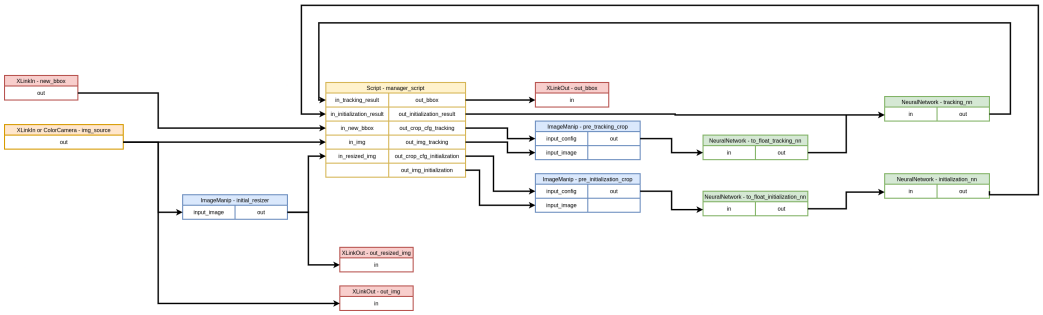
- **ColorCamera**, vozlišče, ki omogoča pridobivanje slik iz kamere.

- **XLinkIn**, vozlišče, ki omogoča pretok podatkov iz gostiteljskega sistema na vgrajeno napravo.
- **XLinkOut**, vozlišče, ki omogoča pretok podatkov iz vgrajene naprave na gostiteljski sistem.
- **NeuralNetwork**, vozlišče, v katerem se izvaja model, ki smo ga prevedli v binarni format.
- **Srcipt**, vozlišče, v katerem se izvaja Python skripta. Vozlišče je namenjeno posredovanju, usmerjanju in lažjemu procesiranju podatkov.
- **ImageManip**, vozlišče, ki omogoča manipulacijo slik. V našem primeru je bilo uporabljeno za skaliranje slike in izrezovanje iskalnega območja.

V sklopu naloge so bili razviti in implementirani štirje tipi cevovoda, kar posledično pomeni štirje različni načini delovanja sledilnika. Implementirane načine lahko razdelimo glede na dve značilnosti:

1. **vir slikovnega toka**: iz vgrajene kamere, v nadaljevanju poimenovane *cam*, ali iz gostiteljskega sistema, v nadaljevanju poimenovan *synthetic*,
2. **način procesiranja**: celotno procesiranje na vgrajeni napravi, v nadaljevanju poimenovani *edge*, ali pa delno na vgrajeni napravi in delno na gostiteljskem sistemu, v nadaljevanju poimenovan *host*.

Zaradi naštetih značilnosti se štiri različice cevovoda (*edge cam mode*, *edge synthetic mode*, *host cam mode* in *host synthetic mode*) med sabo nekoliko razlikujejo. Vseeno pa imajo nekaj skupnih elementov. Na Sliki 3.3 je prikazan diagram cevovoda za načina, pri katerih se celotno procesiranje izvaja na vgrajeni napravi. Na Sliki 3.4 pa je prikazan diagram cevovoda, ki se uporablja pri načinih delovanja, ko se na vgrajeni napravi izvaja samo napovedovanje z nevronskimi mrežami. V nadaljevanju bomo predstavili vsakega od načinov delovanja posebej.



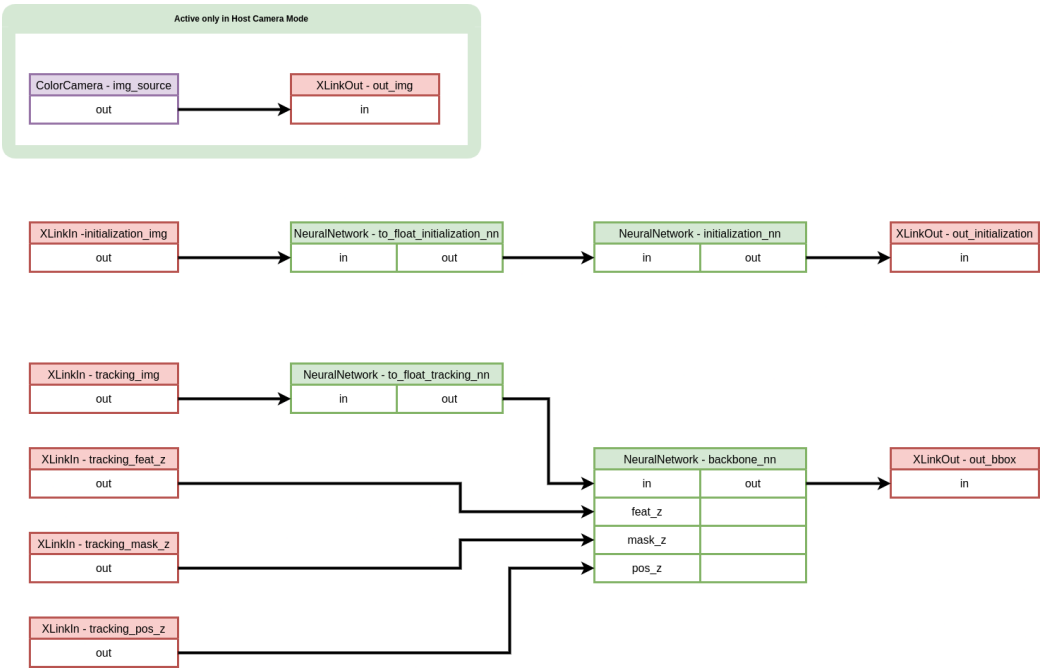
Slika 3.3: Diagram cevovoda *edge cam mode* in *edge synthetic mode*. Pri obeh načinih se celotno procesiranje izvaja na vgrajeni napravi.

Edge synthetic mode

V tem načinu se celotno procesiranje odvija na vgrajeni napravi, vir slikovnega toka pa je gostiteljski sistem. Slikovni tok se preko vozlišča tipa *XLinkIn* pretaka na vgrajeno napravo. Sledilnik od gostiteljskega sistema v vsakem koraku potrebuje novo sličico, v prvem koraku pa tudi začetni omejitveni okvir. Gostiteljski sistem pa po vsakem koraku prejme napovedani omejevalni okvir, ki ga je napovedal model. Ta način delovanja je bil primarno razvit v namene testiranja in evalvacije, saj omogoča ponovljivo testiranje z umetnim slikovnim tokom.

Kot je prikazano na Sliki 3.3, se vsaka sličica po prehodu na vgrajeno napravo najprej skalira na dimenzije 640×840, potem pa preide v vozlišče tipa *Script*, v katerem se izvaja vsa vezna logika. Korak skaliranja smo dodali, da bi lahko zagotovili neodvisnost od uporabljenga vira slikovnega toka in njegove resolucije.

Vezna logika opravlja nalogo usmerjanja podatkov v pravilno vejo cevovoda (*initialization* ali *tracking*), računanja in nastavljanja nastavitev vozlišča za izrez iskalnega območja ter preslikovanja omejitvenega okvirja na začetno velikost vhodne sličice. Usmerjanje se odvija na podlagi vhodnega podatka o novem omejitvenem okvirju. Če je s strani gostiteljskega sistema prišel nov omejitveni okvir, to pomeni, da je sledilnik potrebno ponovno ini-



Slika 3.4: Diagram cevovoda *host camera mode* in *host synthetic mode*. Pri obeh načinih se na vgrajeni napravi izvaja samo napovedovanje z nevronskimi mrežami.

cializirati (*initialization*), če pa tega podatka ni na vhodu, se odvije korak sledenja (*tracking*). V koraku inicializacije se izvede del mreže poimenovan *initialization*, ki izračuna vektorje vhodne vdelave, pozicijskega kodiranja in maske, te podatke vezna logika znotraj vozlišča *Script* ponovno sprejme in ob vsakem naslednjem koraku posreduje v vejo *tracking*. V vseh korakih sledenja vezna logika posreduje slikovni element in vse tri vektorje pridobljene iz inicializacijskega koraka v vejo cevovoda poimenovano *tracking*. Ko vezna logika iz veje *tracking* prejme napovedan omejitveni okvir, ga ponovno preslika na začetno velikost vhodne sličice in ga posreduje na izhodno vozlišče tipa *XLinkOut*. V obeh primerih pa je potrebno slikovne podatke, ki so v podatkovnem tipu *UINT8*, pretvoriti v *FLOAT16*. Za to sta zadolženi dve vozlišči, vsako v svoji pripadajoči veji, v katerih se izvaja preprosta nevronska mreža, ki sprejme podatke tipa *UINT8* in odda podatke tipa *FLOAT16*.

Za tako rešitev smo se odločili zaradi hitrosti izvajanja, saj je to trenutno najboljša rešitev, ki jo omogoča DepthAI. Več o tem problemu in rešitvi smo opisali v Poglavju 3.6.

Edge cam mode

Ta način se od 3.6.1 razlikuje samo v tem, da slikovni tok prejme iz vgrajene kamere. Od gostiteljskega sistema pričakuje samo omejitveni okvir, v vseh nadaljnjih korakih pa sledilnik samostojno sledi objektu in gostiteljskemu sistemu vrača napovedan omejitveni okvir in sličico, ki jo je sledilnik pridobil iz vgrajene kamere. Diagram zgradbe cevovoda je prikazan na Sliki 3.3.

Da bi omogočili enostavno uporabo, smo za oba tipa sledilnika, ki uporabljata podatke iz vgrajene kamere, razvili preprost uporabniški vmesnik. Uporabniški vmesnik omogoča označevanje objekta in prikaz napovedanega omejitvenega okvirja. Celoten sistem je zasnovan na tak način, da lahko uporabnik v katerem koli trenutku označi novo tarčo za sledenje.

Host synthetic mode

V tem načinu se na vgrajeni napravi izvaja samo napovedovanje z nervonskimi mrežami. Zgradba cevovoda je v primerjavi s prejšnjima dvema načinoma (*edge*) precej bolj enostavna. Cevovod je razdeljen na dve popolnoma nepovezani veji *initialization* in *tracking*. Vsa vezna logika in izrezovanje iskalnega območja, ki se je pri prejšnjih dveh načinih izvajala v cevovodu, se sedaj izvaja na gostiteljskem sistemu. Ker se v tem načinu uporablja sintetični slikovni tok, gostiteljski sistem na vgrajeno napravo najprej pošlje sličico in začetni omejitveni okvir, ki sta potrebna za inicializacijo sledilnika. Kot rezultat inicializacije gostiteljski sistem prejme 3 matrike, ki jih proizvede model *initialization*. V vseh nadaljnjih korakih, ko se izvaja sledenje, pa gostiteljski sistem poleg sličice pošlje tudi 3 matrike, ki jih je sprejel v inicializacijskem koraku. Kot rezultat koraka sledenja pa gostiteljski sistem sprejme napovedan omejitveni okvir. Diagram zgradbe cevovoda je prikazan na Sliki 3.4.

Host cam mode

V tem načinu sistem deluje popolnoma enako kot v 3.6.1, le da je na začetek vrinjen še dodaten korak. Ker se v tem načinu delovanja uporablja slikovni tok iz vgrajene kamere, mora gostiteljski sistem najprej iz vgrajene naprave prebrati sličico.

Poglavje 4

Evalvacija

Implementirano različico sledilnika smo eksperimentalno ovrednotili, pri čemer so nas najbolj zanimali hitrost izvajanja, primerljivost učinkovitosti sledilnika STARK Lightning na različnih platformah, primerljivost sledilnika STARK Lightning in STARK_ST ter umestitev dobljenih rezultatov v širši kontekst.

Za eksperimentalno evalvacijo smo uporabili javni podatkovni zbirki izzivov VOT2021 [10] in VOT2022 [11]. Prva podatkovna zbirka je sestavljena iz 60, druga pa iz 62 različnih sekvenc. Organizatorji izziva VOT vsako leto podatkovno zbirko spremenijo glede na rezultate prejšnjega izziva. V grobem vsako leto zamenjajo 10% sekvenc. Testiranje smo izvedli s pomočjo orodja VOT tollkit[1], ki precej olajša izvajanje evalvacije.

Iz izziva VOT2021, so izbrali test VOT-ST2021, saj je bila na tem testu prvotno evalvirana različica uporabljenega sledilnika - STARK_ST. Referenčne vrednosti *(ang. ground truth)* so podane v obliki segmentacijske maske, rezultat sledenja pa lahko predstavlja segmentacijska maska ali pa omejitveni okvir. Iz izziva VOT2022 smo izbrali test VOT-STb2022, kjer so referenčne vrednosti in rezultati sledenja v obliki omejitvenih okvirjev. Metrika, ki na VOT izzivih določa zmagovalca, je pričakovano povprečno prekrivanje *(ang. expected average overlap, EAO)* referenčnega in napovedanega območja.

[1] `https://github.com/votchallenge/toolkit`

4.1 Hitrost delovanja

Ugotovili smo, da v vseh štirih načinih delovanja sledilnik dosega povprečno hitrost 10 FPS. Iz te ugotovitve lahko izpeljemo zaključek, da največ časa za procesiranje zavzame ravno napovedovanje z nevronsko mrežo *tracking*. To smo tudi potrdili z izvajanjem meritev časa izvajanja posameznih delov cevovoda. Ugotovili smo, da korak, v katerem se izvaja nevronska mreža *tracking*, traja v povprečju kar 68 ms. Vsi ostali koraki pa v povprečju doprinesejo še 30 ms. Seštevek vseh korakov pa znaša približno 100 ms, kar je tudi v skladu z izmerjenimi 10 FPS.

Za primerjavo sledilnik STARK Lightning, implementiran v okolju PyTorch, na grafični procesorski enoti NVIDIA GeForce GTX 1080, doseže povprečno hitrost 214 FPS. Če pa sledilnik poženemo znotraj okolja ONNX Runtime in za procesiranje uporabimo procesor Intel i7-6700K, sledilnik doseže povprečno hitrost 63 FPS.

Pri tej analizi velja izpostaviti tudi energetske porabe posameznih sistemov. Maksimalna poraba električne energije grafične procesorske enote NVIDIA GeForce GTX 1080 je 180 W, maksimalna poraba procesorske enote Intel i7-6700K je 91 W, medtem ko pa je maksimalna poraba naprave Luxonis OAK-1 samo 4.56 W. Pri tem je seveda potrebno upoštevati, da je za prvi dve enoti podana maksimalna dovoljena poraba, ki pa ne nujno predstavlja realne porabe med delovanjem, hkrati pa to predstavlja samo porabo ene komponente in ne porabo celotnega računalniškega sistema. Očitno pa je razvidno, da vgrajena naprava ponuja odličen kompromis med učinkovitostjo sledenja in porabo energije.

4.2 Platforma

Primerjali smo natačnost sledilnika, glede na platformo, na kateri se izvaja. Test smo izvedli na treh različnih platformah. Prva platforma je bila vgrajena naprava Luxonis OAK-1 (uporabljena je bila različica 3.6.1), druga ONNX Runtime, pri čemer se celotno procesiranje izvaja na procesorju Intel i7-

6700K ter PyTorch, pri čemer se napovedovanje z nevronsko mrežo izvaja na grafični procesorki enoti NVIDIA GeForce GTX 1080. Pri testu nismo opazili omembe vredne spremembe pri točnosti napovedovanja, s tem lahko zaključimo, da platforma in posledično strojna oprema, na kateri se sledilnik izvaja, ne vpliva na njegovo točnost. Iz te ugotovitve lahko izpeljemo tudi zaključek, da optimizacija in pretvorba podatkovnega tipa uteži in ostalih parametrov modela, iz FLOAT32 v FLOAT16, pri pretvorbi v OpenVINO format nimata vpliva na točnost sledilnika. Rezultati testa so podani v prvem delu Tabele 4.1.

4.3 Primerjava verzij metode

Sledilnike smo najprej primerjali na testu VOT-ST2021, kjer je bil tudi prvotno testiran sledilnik STARK_ST (v [10] je sicer poimenovan STARK_RT, po pregledu kode[2] pa smo opazili, da se je uporabila različica sledilnika STARK_ST). Različica sledilnika, ki je bila uporabljena na VOT2021, je na samem izhodu sledilnika uporabljala dodaten modul Alpha-Refine [18]. Dodan modul omogoča napovedovanje segmentacijske maske na podlagi napovedanega omejitvenega okvirja in vhodne slike. Taka uporaba modula Alpha-Refine je bila zelo popularna na izzivu VOT2021 [10]. Naši implementaciji sledilnika STARK Lightning na vgrajeni napravi dodatnega modula ni bilo moč dodati. Oba sledilnika pri inicializaciji referenčno segmentacijsko masko pretvorita v omejitveni okvir, tako da vse vrednosti segmentacijske maske ležijo znotraj omejevalnega okvirja, in ga uporabita za inicializacijo. Ker pa sledilnika podajata napovedane regije v različnih formatih direktna primerjava med njima ni poštena, saj bo pri napovedi v obliki segmentacijske maske potencialno višja natančnost. Iz rezultatov testa, ki so podani v Tabeli 4.1, lahko opazimo, da v primerjavi z sledilnikom STARK_ST naša implementacija sledilnika STARK Lightning proizvede približno 60% slabšo

[2] `http://data.votchallenge.net/vot2021/trackers/STARK_RT-code-2021-05-30T13_54_26.067770.zip`

pričakovano povprečno prekrivanje omejevalnega okvirja in za približno 45% in 30% slabšo natačnost ter robustnost. Tako veliko razliko lahko pojasnimo z različnimi načini podajanja napovedanega območja. Vpliv načina podajanja napovedanega območja smo tudi potrdili z dodatnim testom. Sledilniku STARK_ST smo odstranili dodan modul Alpha-Refine in ponovili test VOT-ST2021. Iz rezultata v Tabeli 4.1, lahko opazimo, da se samo s spremembo načina formata napovedovanja, EAO poslabša za za približno 50%, natačnost za približno 40%, pri robustnosti pa lahko opazimo precej manjši padec, za zgolj 11%.

Oba sledilnika smo testirali tudi na testu VOT-STb2022, ki referenčne in napovedane vrendosti obravnava zgolj kot omejitvene okvirje. Iz rezultatov testa, ki so podani v Tabeli 4.2, lahko razberemo precej bolj realno sliko razlike med sledilnikoma. Naša implementacija sledilnika STARK Lightning na vgrajeni napravi proizvede zgolj 27% slabši EAO kot sledilnik, ki za delovanej potrebuje bistveno več resursov. Če našo implementacijo primerjamo z rezultati ostalih sledilnikov iz [11], lahko opazimo, da bi naša implementacija dobila boljše rezultate kot 11 sodelujočih sledilnikov.

Pri analizi rezultatov za posamezne sekvence, bi radi izpostavili sekvenco *birds2*, ki je prisotna v VOT2021 in VOT2022 podatkovni zbirki. Opazili smo, da ta sekvenca povzroča največ težav obem sledilnikom. Nekaj sličic iz sekvence z vrisanimi napovedanimi okvirji in referečnimi segmentacijskimi maskami je prikazanih na Sliki 4.1.

Slika 4.1: Slika prikazuje šest sličic iz sekvence *birds2* iz podatkovne zbirke testa VOT2021. Številke v zgornjem levem kotu vsake sličice predstavljajo zaporedno številko sličice v sekvenci, rdeč okvir predstavlja napovedan omejevalni okvir, modri poligon pa predstavlja referenčno segmentacijsko masko.

Sledilnik	**EAO**	**Accuracy**	**Robustnest**
Sledilniki, ki smo jih sami testirali na podatkovni zbirki VOT-ST2021 [10]			
STARK Lightning (DepthAI, FLOAT16)	0.213	0.429	0.590
STARK Lightning (ONNX Runtime, FLOAT32)	0.212	0.429	0.590
STARK Lightning (PyTorch, FLOAT32)	0.211	0.429	0.583
STARK_ST (brez modula Alpha-Refine [18])	0.271	0.468	0.732
Različica sledilnika STARK, ki je bila testirana v [10]			
STARK_ST	0.534	0.781	0.830
3 najboljši sledilniki na testu VOT-ST2021 [10]			
RPTMask	0.568	0.764	0.859
CFRPT	0.551	0.745	0.853
TransT_M	0.550	0.742	0.869
3 najslabši sledilniki na testu VOT-ST2021 [10]			
KCF	0.168	0.421	0.489
LGT	0.133	0.335	0.448
L1APG	0.083	0.359	0.222

Tabela 4.1: Tabela prikazuje rezultate testa VOT-ST2021 [10].

Sledilnik	**EAO**	**Accuracy**	**Robustnest**
Sledilniki, ki smo jih sami testirali na podatkovni zbirki VOT-STb2022 [11]			
STARK Lightning (DepthAI, FLOAT16)	0.351	0.690	0.627
STARK_ST (brez modula Alpha-Refine [18])	0.482	0.774	0.780
3 najboljši sledilniki na testu VOT-STb2022 [11]			
MixFormerL	0.602	0.831	0.859
DAMT	0.602	0.776	0.887
OSTrackSTB	0.591	0.790	0.869
3 najslabši sledilniki na testu VOT-STb2022 [11]			
KCF	0.239	0.542	0.532
ANT	0.209	0.492	0.484
LGT	0.195	0.461	0.486

Tabela 4.2: Tabela prikazuje rezultate testa VOT-ST2021 [10].

Poglavje 5

Zaključek

V diplomskem delu smo implementirali in ovrednotili delovanje modernega vizualnega sledilnika STARK. Sledilnik deluje na osnovi umetnih nevronskih mrež, bolj natančno umetnih nevronskih mrež z arhitekturo transformer. Med pregledom objavljenih del na temo vizualnega sledenja na vgrajenih napravah nismo našli nobenega, ki bi uporabil sledilnik z najmodernejšo arhitekturo - transformer. Cilj dela je bilo preučiti možnosti take implementacije, sama implementacija ter ovrednotenje delovanja sledilnika. Sledilnik smo implementirali na vgrajeni napravi Luxonis OAK-1, ki podpira ekosistem DepthAI. Zaradi omejene procesorske moči smo se odločili za uporabo poenostavljene različice modela STARK - STARK Lightning. Sledilnik je bilo najprej potrebno predelati, ga pretvoriti v ONNX format ter ga s pomočjo nabora orodji OpenVINO optimizirati in prevesti v binarno obliko. V okolju DepthAI je bilo potrebno zasnovati cevovod ter vanj umestiti model. Za testiranje sledilnika smo uporabili testni nabor podatkov VOT-ST2021. Rezultati testiranja so pokazali, da je implementacija na vgrajeni napravi enaka tisti v okolju ONNX Runtime.

Implemetirali smo 4 različne načine delovanja, ki so podrobneje opisani v Podpoglavju 3.6.1. Način delovanja, ki bi bil najbolj uporaben v realnem svetu 3.6.1, uporabniku v realnem času prikazuje sliko, ki jo zajema kamera. Uporabnik lahko na sliki označi željeno tarčo, sledilnik pa bo tej tarči začel

slediti in okoli nje izrisovati omejitveni okvir. Uporabnik lahko v katerem koli trenutku označi novo tarčo za sledenje.

S pomočjo testa VOT-STb2022 [11], smo ugotovili, da naša implementacija sledilnika, ki deluje povsem avtonomno na vgrajeni napravi z maksimalno porabo energije 4.56 W, dosega zgolj 27% slabše rezultate, kot sledilnik STARK_ST, ki za delovanje potrebuje precej več energije in procesorskih resursov.

Z analizo smo ugotovili, da je časovno najbolj potraten del sistema ravno napovedovanje z nevronsko mrežo *tracking*. Čas izvajanja nevronske mreže je v povprečju 68 ms, vse ostalo procesiranje pa doprinese še 30 ms. Glede na ugotovljeno ne pričakujemo, da bi se uporabljeno arhitekturo na trenutni strojni opremi lahko dodatno pohitrilo. Nadaljevati bi bilo treba v smeri izdelave po meri narejene nevronske mreže, ki bi v osnovi bila manj procesorsko zahtevna.

Celotna literatura

[1] Sai Balaji. *Binary Image classifier CNN using TensorFlow*. URL: https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697 (pridobljeno 2023).

[2] David S. Bolme in sod. "Visual object tracking using adaptive correlation filters". V: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, str. 2544–2550. DOI: 10.1109/CVPR.2010.5539960.

[3] Nicolas Carion in sod. "End-to-end object detection with transformers". V: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer. 2020, str. 213–229.

[4] Xiaohan Ding in sod. "Repvgg: Making vgg-style convnets great again". V: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, str. 13733–13742.

[5] Mauro Fernández-Sanjurjo, Manuel Mucientes in Víctor Manuel Brea. "Real-Time Multiple Object Visual Tracking for Embedded GPU Systems". V: *IEEE Internet of Things Journal* 8.11 (2021), str. 9177–9188. DOI: 10.1109/JIOT.2021.3056239.

[6] Erhan Gundogdu in A Aydın Alatan. "Good features to correlate for visual tracking". V: *IEEE Transactions on Image Processing* 27.5 (2018), str. 2526–2540.

[7] Kaiming He in sod. "Deep residual learning for image recognition". V: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, str. 770–778.

[8] João F Henriques in sod. "High-speed tracking with kernelized correlation filters". V: *IEEE transactions on pattern analysis and machine intelligence* 37.3 (2014), str. 583–596.

[9] Matej Kristan in sod. "A Novel Performance Evaluation Methodology for Single-Target Trackers". V: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.11 (nov. 2016), str. 2137–2155. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2516982.

[10] Matej Kristan in sod. *The Ninth Visual Object Tracking VOT2021 Challenge Results*. 2021.

[11] Matej Kristan in sod. "The Tenth Visual Object Tracking VOT2022 Challenge Results". V: *Computer Vision – ECCV 2022 Workshops*. Ur. Leonid Karlinsky, Tomer Michaeli in Ko Nishino. Cham: Springer Nature Switzerland, 2023, str. 431–460. ISBN: 978-3-031-25085-9.

[12] Ville Lehtola in sod. "Evaluation of Visual Tracking Algorithms for Embedded Devices". V: *Image Analysis*. Ur. Puneet Sharma in Filippo Maria Bianchi. Cham: Springer International Publishing, 2017, str. 88–97. ISBN: 978-3-319-59126-1.

[13] Luxonis. *Luxonis OAK-1*. [Splet; dostopano Marec 9, 2023]. 2023. URL: https://docs.luxonis.com/projects/hardware/en/latest/_images/oak-11.png.

[14] Ziang Ma in sod. "Rpt: Learning point set representation for siamese visual tracking". V: *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*. Springer. 2020, str. 653–665.

[15] Nisarg Patel. *What Is Deep Learning?* [Splet; dostopano Marec 9, 2023]. 2023. URL: https://miro.medium.com/v2/resize:fit:828/0*nQzGnGIsW6LJxJHB.

[16] Joseph Redmon in Ali Farhadi. "Yolov3: An incremental improvement". V: *arXiv preprint arXiv:1804.02767* (2018).

[17] Ashish Vaswani in sod. "Attention is All You Need". V: 2017. URL: https://arxiv.org/pdf/1706.03762.pdf.

[18] Bin Yan in sod. "Alpha-Refine: Boosting Tracking Performance by Precise Bounding Box Estimation". V: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, str. 5285–5294. DOI: 10.1109/CVPR46437.2021.00525.

[19] Bin Yan in sod. "Learning spatio-temporal transformer for visual tracking". V: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, str. 10448–10457.