Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nik Prinčič

Vizualno sledenje 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:
Besedilo teme diplomskega dela študent prepiše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stavkih bo opisal, kaj pričakuje od kandidatovega diplomskega dela. Kaj so cilji, kakšne metode naj uporabi, morda bo zapisal tudi ključno literaturo.

Title: Visual tracking on embedded devices

Description:
opis diplome v angleščini

Na tem mestu zapišite, komu se zahvaljujete za pomoč pri izdelavi diplomske naloge oziroma pri vašem študiju nasploh. Pazite, da ne boste koga pozabili. Utegnil vam bo zameriti. Temu se da izogniti tako, da celotno zahvalo izpustite.

Svoji dragi Alenčici.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Pregled področja **3**

3 Metodologija **5**

3.1 Umetne nevronske mreže 5

3.2 Konvolucijske nevronske mreže 6

3.3 Arhitektura transformer 7

3.4 Model STARK 11

4 Implementacija **17**

4.1 Luxonis OAK-1 17

4.2 DepthAI 18

4.3 OpenVINO 19

4.4 Uporabljene tehnologije 20

4.5 Prilagoditev modela 20

4.6 Prevajanje modela 22

5 Evalvacija? Diskusija? **31**

6 Zaključek **37**

Članki v revijah **39**

Članki v zbornikih **41**

Celotna literatura **43**

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 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, 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 uporabno 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 to, da so performance sledenja neodvisne od gostiteljskega sistema.

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

Abstract

Title: Visual tracking on embedded devices

Author: Nik Prinčič

This sample document presents an approach to typesetting your BSc thesis using LaTeX. A proper abstract should contain around 100 words which makes this one way too short.

Keywords: embedded computer vision, DepthAI, visual tracker.

Poglavje 1

Uvod

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

Vizualno sledenje je področje, ki se ukvarja z iskanjem in sledenjem objektov v videu. V tem delu smo se osredotočili na SOT, kjer je cilj slediti samo enemu objektu. Sledilniku je naprej potrebno podati omejevalni okvir tarče *(ang. bounding box, BBOX)*, kateri 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 pripadajoč BBOX. V preteklosti so bil najbolj popularni tako imenovani klasični algoritmi (npr. KCF [6], MOSSE [2], ...), sedaj pa prevladujejo sledilniki, ki temeljijo na nevronskih mrežah. V zadnjih nekaj letih je upravičeno vedno bolj popularna arhitektura transformer [14], 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, so se začele razvijati namenske procesorske enote VPU *(ang. visual processing unit)*, ki so optimizirane za izvajanje nevronskih mrež in pospešeno izvajanje operacij na slikovnimi tokovi. V to družino procesorskih enot spada tudi čip, Intel Movidius Myriad X, ki je v osrčju uporabljen naprave v tem diplomskem delu.

V okviru te diplomske naloge smo se osredotočili na sledilnik STARK [15], ter vgrajeno napravo Luxonis OAK-1. Cilj naloge je bil sledilnik prilagoditi uporabi na tej napravi, ga prevesti v potreben format, ter ga umestiti v cevovod. Cevovod je bilo potrebno tudi smiselno zasnovati, da v model pridejo pravilno oblikovani podatki.

Diplomsko delo je razdeljeno v 5 delov. V poglavju 2 bomo prestavili pregled področja. V poglavju 3 bomo opisali metodologijo, to zavzema hiter opis nevronskih mrež na splošno, arhitekture CNN in transformer ter predstavitev izbranega sledilnika. V poglavju 4 bomo opisali podrobnosti implementacije, kar vključuje opis uporabljene vgrajene naprave, prilagoditev modela ter njegovo pretvorbo v pravilen format, opis implementacije cevovoda in opis 4 različnih načinov delovanja, ki smo jih pripravili. V poglavju 5 bomo prestavili 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 6 je zaključek, ki povzema ugotovitve, ter predstavi možne izboljšave za nadaljnji razvoj.

Poglavje 2

Pregled področja

V tem poglavju bomo pregledali dosedanje delo na področju vizualnega sledenja na splošno in na vgrajenih napravah. Kot merilo uspešnosti sledilnika bom uporabili rezultate izziva VOT *(ang. VOT challenge)* [7], ki je eden najbolj uveljavljenih testov na področju vizualnega sledenja. VOT izziv vsako leto priredi evalvacijo novih sledilnikov. Sledilnike so v zadnji izvedbi, VOT2022 [9], testirali na v sedmih različnih kategorijah. Glede na rezultate izziva v zadnjih nekaj letih, lahko opazimo porast v popularnosti in uspešnosti sledilnikov, ki uporabljajo arhitekturo transformer. Iz rezultatov VOT2022, objavljenih v [9], lahko razberemo da 9 od najboljših 10 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*[10] so primerjali performance med 5 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*[4], so predstavili MOT *(ang. multiple object tracking)* na napravi Nvidia Jetson TX2, kjer so uporabili več stopenjsko arhitekturo, detektor (YOLOv3 [13])

in sledilnik (KCF [6]), in dobili dobre rezultate.

Poglavje 3

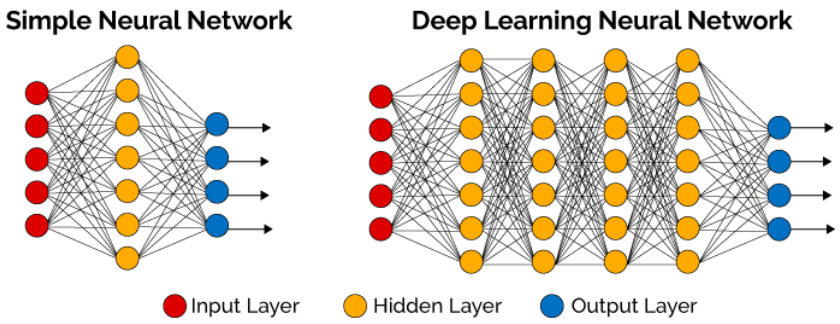
Metodologija

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 in opisali izbrani sledilnik.

3.1 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 3.1 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



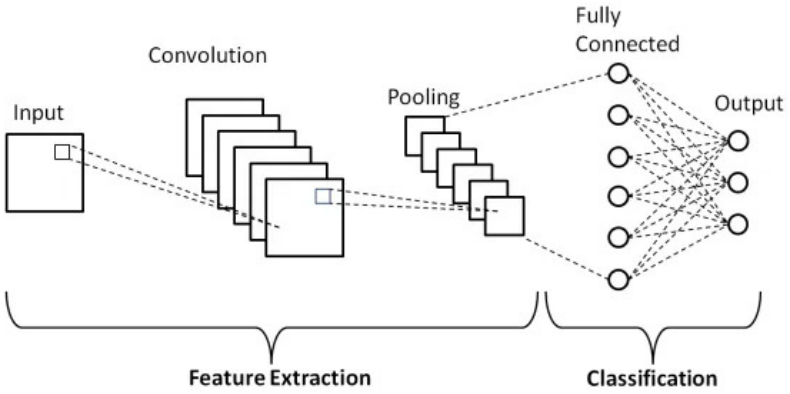
Slika 3.1: Primerjava zgradbe NN in DNN [12].

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 morem 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)*.

3.2 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 3.2.



Slika 3.2: Poenostavljena arhitektura konvolucijkse 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:

$$(f * g)(t) = \int_{\tau=-\inf}^{\inf} f(\tau)g(t-\tau)d\tau \tag{3.1}$$

Kjer je f predstavlja vhodno funkcijo, g pa jedro konvolucije. Ker se pri obdelavi digitalnih podatkov pogosto uporablja diskretna konvolucija, z enačbo 3.1 spremenjena v:

$$(f * g)[t] = \sum_{k=-\inf}^{\inf} f[k]g[n-k] \tag{3.2}$$

3.3 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*

[14]. Primarno je bila arhitektur razvita za naloge procesiranja naravnega jezika *(Natural language processing, NLP)*, sejda pa postaja popularna tudi v drugih domenah strojnega učenja, med drugim tudi v računalniškem vidu. Osnovna arhitektura, kji je bila predstavljena v [14], vključuje kodirni *(ang. encoder)* in dekodirni modul *(ang. decoder)*. Kodirnik je sestavljen iz dveh podslojev, dekodirnik pa iz treh.

3.3.1 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 govorim, 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

$$\begin{aligned} Q &= XW^Q, \\ K &= XW^K, \\ V &= XW^V, \end{aligned} \tag{3.3}$$

Matrika pozornosti je definirana z enačbo:

$$A(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_m}})V \tag{3.4}$$

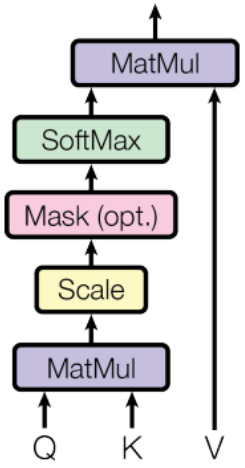
kjer funkcija *softmax* spremeni vektor n realnih števil v vektor verjetnostne porazdelitve.

Iz 3.3 in 3.4 lahko izračunamo končno izhodno vrednost mehanizma

$$S = AV, \tag{3.5}$$

na sliki 3.3 je potek izračuna prikazan v obliki diagrama.

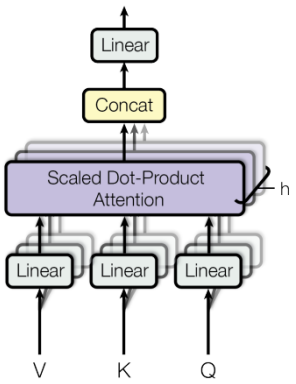
Ko se mehanizem pozornosti uporabi v dekodirnem modulu je potrebno določen del podatkov skriti. Bolj natančno, modelu je treba preprečiti, z podatki, ki jih še ni napovedal.



Slika 3.3: Diagram izračuna pozornosti [14].

3.3.2 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 3.4 je prikazan diagram poteka več-glave pozornosti.



Slika 3.4: Diagram izračuna več-glave pozornosti [14].

3.3.3 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 sledeči funkciji 3.6

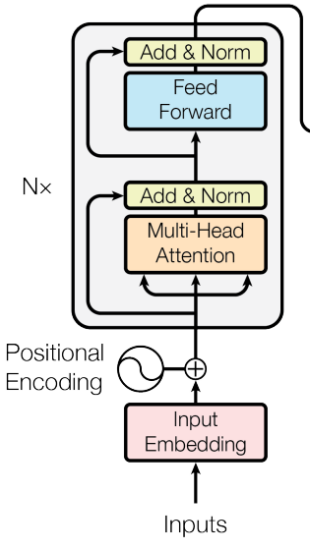
$$\begin{aligned} PE(pos, 2i) &= sin(pos/10000^{2i/d_m}) \\ PE(pos, 2i+1) &= cos(pos/10000^{2i/d_m}) \end{aligned} \tag{3.6}$$

v enačbi 3.6 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.

3.3.4 Kodirni modul

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



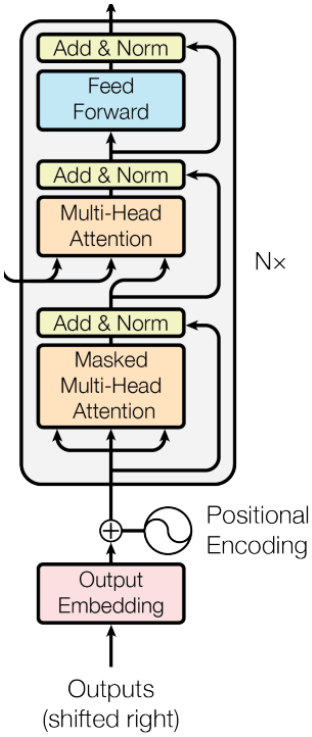
Slika 3.5: Diagram kodrinega modula [14].

3.3.5 Dekodirni modul

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

3.4 Model STARK

Model STARK [15] spada v družino SOT *(ang. single object tracking)* sledilnikov. Primarno je sestavljen iz dveh arhitektur, konvolucijskih nevronskih



Slika 3.6: Diagram dekodirnega modula [14].

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, kateremu 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 matrico *(ang. template)* in dinamično matrico, ki se skozi čas dinamično posodablja. Z uporabo dinamična matrice, 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 prejem 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 matrice. Ob vsaki naslednji sličici, pa so vhodni podatki iskalno območje izrezano iz vhodne slike, začetna matrica in dinamična matrica, sledilnik pa vrne izračunani omejitveni okvir.

3.4.1 Arhitektura modela stark

V tem delu smo zaradi manjše procesorske in prostorske zahtevnosti uporabili, različico modela STARK, ki uporablja samo prostorsko komponento *STARK_S*. V nadaljevanju bomo predstavili delovanje in arhitekturo modela.

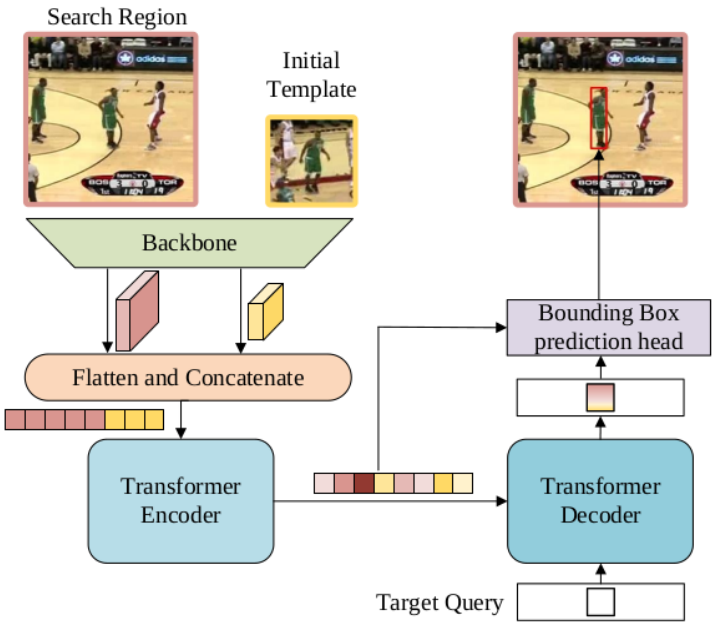
Arhitekturo modela lahko razdelimo na tri glavne dele: *(1)* konvolucijska hrbtenica *(ang. convolutional backbone)*, *(2)* kodirni-dekodirni transformer in predikcijsko glavo za omejitveni okvir *(ang. bounding box prediction head)*. V nadaljevanju bomo predstavili vsakega od teh delov.

Konvolucijska hrbtenica

Konvolucijsko hrbtenico sestavlja model ResNet [5], kateremu so odstranili zadnjo sekcijo in polno povezane sloje. Kot vhod prejme hrbtenica začetno matrico $z \in \mathbb{R}^{3\times H_z\times W_z}$ in iskalno območje $x \in \mathbb{R}^{3\times H_x\times W_x}$. Po prehodu čez hrbtenico dobimo dve matrici značilk $f_z \in \mathbb{R}^{C\times \frac{H_z}{s}\times \frac{W_z}{s}}$ in $f_x \in \mathbb{R}^{C\times \frac{H_x}{s}\times \frac{W_x}{s}}$.

3.4.2 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 sekvenco značilk je vhod za kodirni mo-



Slika 3.7: Diagram arhitekture modela STARK_S [14].

dul. Kodirni modul je sestavljen iz N identičnih 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, katere se uporabijo za lokalizacijo objekta.

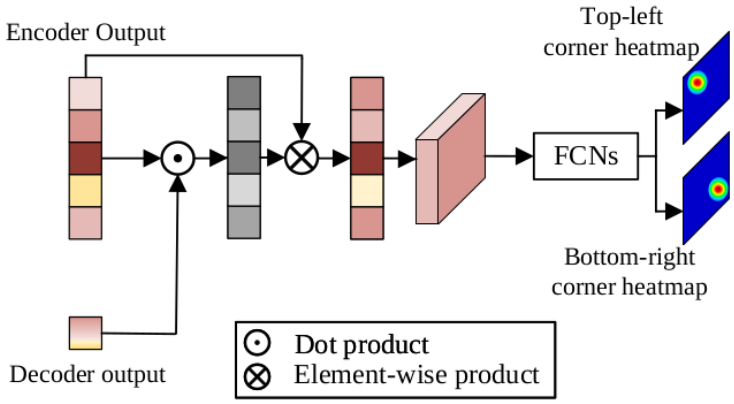
3.4.3 Dekodirnik

Dekodirnik kot vhod prejem 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 matrici in iskalnem območju, kar omogoča učenje robustnih reprezentacij predikcij končnega omejevalnega okvirja.

3.4.4 Glava za predikcijo omejevalnega okvirja

Model uporablja novo zasnovani sistem za predikcijo omejevalnega okvirja, ki deluje na predikcij 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 z 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 3.8 je z diagramom predstavljen opisan potek predikcije omejevalnega okvirja, zadnji korak računanja kordinat iz verjetnostnih matrik pa je opisan v formuli 3.7.

$$
\begin{aligned}
(\hat{x_{tl}}, \hat{y_{tl}}) &= (\sum_{y=0}^{H} \sum_{x=0}^{W} x \cdot P_{tl}(x, y), (\sum_{y=0}^{H}) \sum_{x=0}^{W} y \cdot P_{tl}(x, y)), \\
(\hat{x_{br}}, \hat{y_{br}}) &= (\sum_{y=0}^{H} \sum_{x=0}^{W} x \cdot P_{br}(x, y), (\sum_{y=0}^{H}) \sum_{x=0}^{W} y \cdot P_{br}(x, y)).
\end{aligned}
\tag{3.7}
$$



Slika 3.8: Diagram poteka izračuna omejitvenega okvirja [14].

Poglavje 4

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.

4.1 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 jih razlikuje odprtokodnost vseh njihovih naprav in ogrodja *(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 2 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 4.1.

V osrčju naprave je modul RVC2 *(Robotic Vision Core 2)*. Modul ponuja 4 TOPS procesorske moči, od katerih je 1.4 TOPS rezerviranih za izvajanje nevronskih mrež. Podpira pa tudi hardversko kodiranje slikov-

Slika 4.1: Slika prikazuje uporabljeno napravo Luxonis OAK-1 [11].

nih 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)*, 16 vektorskih procesorskih enot SHAVE, 20 hardverskih pospeševalnih enot poimenovanih *Enhanced Vision Accelerators*, ter 2.5 MB vgrajenega hitrega homogenega spomina.

4.2 DepthAI

DepthAI je hkrati programsko ogrodje *(ang. framework)* in tudi ekosistem odprtokodne programske in hardverske 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 odstopamo do resursov naprave. Princip delovanja stoji na 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 z kamero *ColorCamera*,

- vozlišče za pretok podatkov preko USB povezave v semri iz naprave na gostiteljski sistem *XLinkOut*,
- vozlišče za pretok podatkov preko USB povezave v semri iz gostiteljskega sistema v napravo *XLinkIn*,
- vozlišče za izvajanje pomeri narejenih skript na naparvi - *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,

4.3 OpenVINO

OpenVINO 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 MyriadX. Vsak model je potrebno najprej pravilno prilagoditi, da ustreza zahtevam in omejitvam ciljnega sistema.

Pri prilagajanju moremo biti pozorni na tipe slojev, ki so uporabljeni v modelu, saj vsi 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.

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 parametru model, imena izhodnih in vhodnih podatkov ter dimenzionalnost vhodnih podatkov. Pri podajanju dimenzionalnosti vhdonih podatkov moramo upoštevati, da nekatere ciljne naprave ne podpiranjo dinamičnih velikosti.

Po optimizacijskem koraku sledi še zandji korak, prevajanje. Model je potrebno prevesti v pravilen format. Pri prevajanju moramo podati tip končne

napave, podatkovni tip vhodnih podatkov in število vektorskih procesorjev SHAVE, ki jih bo model uporabil. Rezultat prevajanja je binarna datoteka v formatu *.blob*.

4.4 Uporabljene tehnologije

Uporabljen je bil že naučen model STARK [15], 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. Model so [15] implementirali v ogrodju PyTorch, z uporabo programskega jezika Python, v katerem smo tudi mi nadaljevali z razvojem. Za lažji razvoj sta se uporabili dve konetejnerizirani 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 okolji je prenosljivost in reproduciranje rezultatov.

4.5 Prilagoditev modela

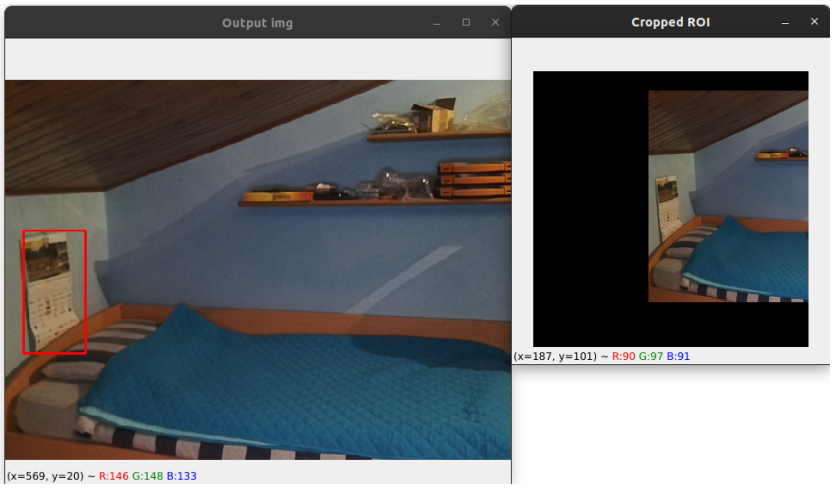
Model pri delovanju potrebuje vhdono iskalno območje, ki je izrezano iz vhdone slike. Iskalno območje je fiksne dimenzije, če gre za inicializacijski korak je to dimenzija $128 * 128$, pri vseh nadljnih korakih pa je dimenzija iskalnega območja $320 * 320$. Izrezovanje iskalnega območja deluje tako, da se iz vhdone slike izreže območje ki ga omejuje omejevalni okvir, katerega povečamo za določen faktor. Izračun faktorja C, ki predstavlja vrednost za katero se vhodni omejevalni okvir poveč, poteka po formuli 4.1, kjer w predstavlja širino, h pa višino vhdonega omejevalnega okvirja. F predstavlja faktor iskanja. Pri inicializacijske koraku im F vrednost 2, pri vseh ostalih korakih pa 5.

$$C = \lceil \sqrt{w * h} * F \rceil \tag{4.1}$$

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 uplivali 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 vmeščeno 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, maks pa se izračuna po naslednjih korakih:

1. izračunamo povprečje po 2. dimneziji 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 piksele 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, katerag povreča vrednost bo 0), izvedemo še operacijo maksimalnega združevanja *(ang. max pooling)* z velikostjo okna 3×3 in vrednostjo 1.

Za enostavnejšo umestitev modela v cevovod, je model razdeljen na 2 dela. Prvi model, od sedaj naprej ga bomo poimenovali samo **backbone**, zajema samo hrbtenico, sloj ozkega grla in pozicijsko kodiranje. Ta model se uporabi ob inicializaciji. Drugi model, od sedaj naprej ga bomo poimenovali **complete**, pa ostaja nespremenjen in se uporablja pri vsakem nadaljnem koraku.



Slika 4.2: Primer situacije kjer je izrezanemu iskalnemu območju dodan rob.

4.6 Prevajanje modela

Za lažjo pripravo delovnega okolja in programksega paketa OpenVINO je bilo uporabljeno kontejnerizirano okolje Docker.

Model je najprej potrebno iz ogrodja PyTorch izvoziti v format *ONNX*. *ONNX* je odprtokodni format za shranjevanje modelov, ki ga je ustvarila Microsoft. Za izvoz lahko uporabi funckionalnost, ki je vgrajena v PyTorch. Pri izvotu moramo podati, primer vhodnih podatkov, ter poimenovati - labelirati vhodne in izhodne argumente. V tabeli 4.1 so podani uporabljeni argumenti.

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 izvotu v format *ONNX*, moramo tudi pri optimizaciji podati argumente, ki določajo imena in dimenzionalnosti vhodnih podatkov, ter imena izhodnih podatkov. Izhod optimizacijeksega 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.

labela	**opis**	**dimenzije**	**podatkovni tip**
Model: backbone			
img	iskalno območje	$1 \times 3 \times 128 \times 128$	float16
Model: complete			
img_x	iskalno območje	$1 \times 3 \times 320 \times 320$	float16
feat_z	sekveca značilk matrice	$64 \times 1 \times 128$	float16
mask_z	maska matrice	1×64	bool
pos_z	pozicijsko kodiranje matrice	$64 \times 1 \times 128$	float16

Tabela 4.1: Tabela prikazuje uporabljene argumnte pri izvozu modelov v format *ONNX*.

V zadnjem koraku moramo model prevesti v binarni format, ki se lahko uporabi na procesorski enoti MyriadX. V ta namen se je uporabilo prodje *compile tool* iz programksega paketa OpenVINO. Orodju podamo pot do *.xml* in *.bin* datotek, ki jih je ustvaril *Model optimizer* ter potrbne 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 MyriadX.
- **-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 **complete** imel vrednost 8, pri modelu **backbone** pa 1.

Nekompatibilnost podatkovnih tipov

Med tem ko smo optimizirali model, smo se zavedali, da imamo težavo. Vsi vhodni podatki modela morajo biti enakega podatkovnega tipa, ampak sli-

kovni podatki so v formatu *UINT8*, vsi ostali pa v formatu *FLOAT16*. Problem smo rešili tako, da smo dodali dodatno nevronsko mrežo, ki služi samo za pretvorbo podatkov iz *UINT8* v *FLOAT16*. Ta rešitev je bila najbolj optimalna saj omogoča najhitrejše izvajanje pretvorbe.

4.6.1 DepthAI cevovod

Vso funkcijonalnost, ki jo želimo izvajati na vgrajeni napravi je potrbno vmestiti 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:

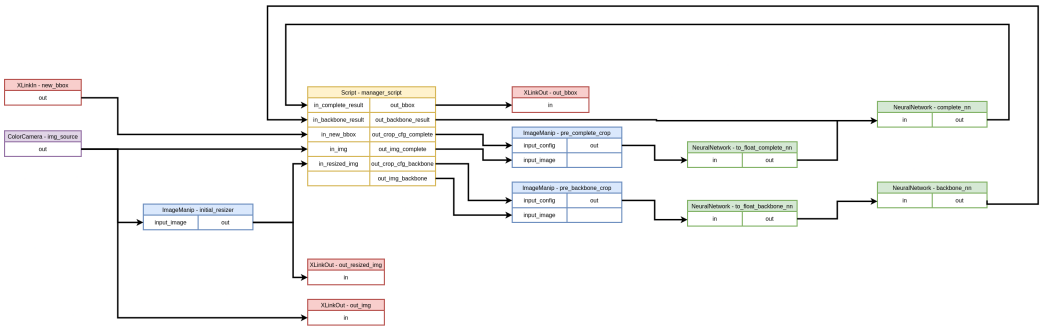
- **ColorCamera**, vozlišče, ki omogoča pridobivanje slik iz kamere.
- **XLinkIn**, vozlišče, ki omogoča pretok podatkov iz gostiteljskega sistema na vgrajeno napravo.
- **XLinkOut**, vozlišče, ki omogoča pretok podatkov iz vgrajene naprave na gostiteljski sistem.
- **NeuralNetwork**, vozlišče, v katerem 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 izrezovanju iskalnega območja.

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

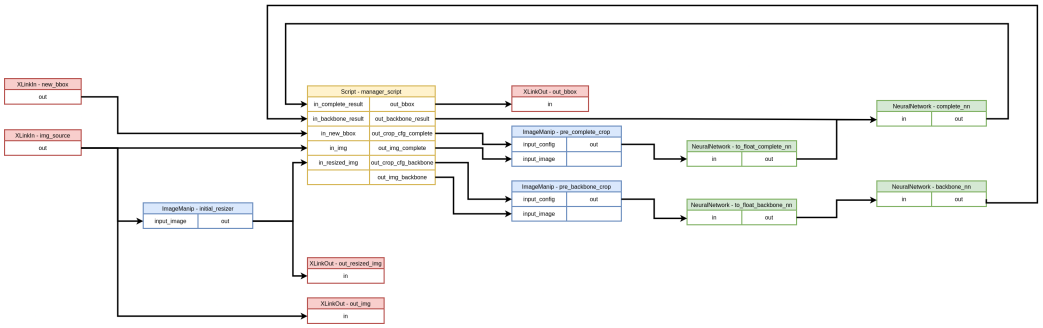
1. **vir slikovnega toka**: iz vgrajene kamer, v nadaljevanju poimenovan *cam*, ali iz gostiteljskega sistema, v nadaljevanju poimenovan *synthetic*,

2. **način procesiranja**: celotno procesiranje na vgrajeni napravi, v nadaljevanju poimenovan *edge*, ali pa delno na vgrajeni napravi in delno na gostiteljskem sistemu, v ndaljevanju poimenovan *host*

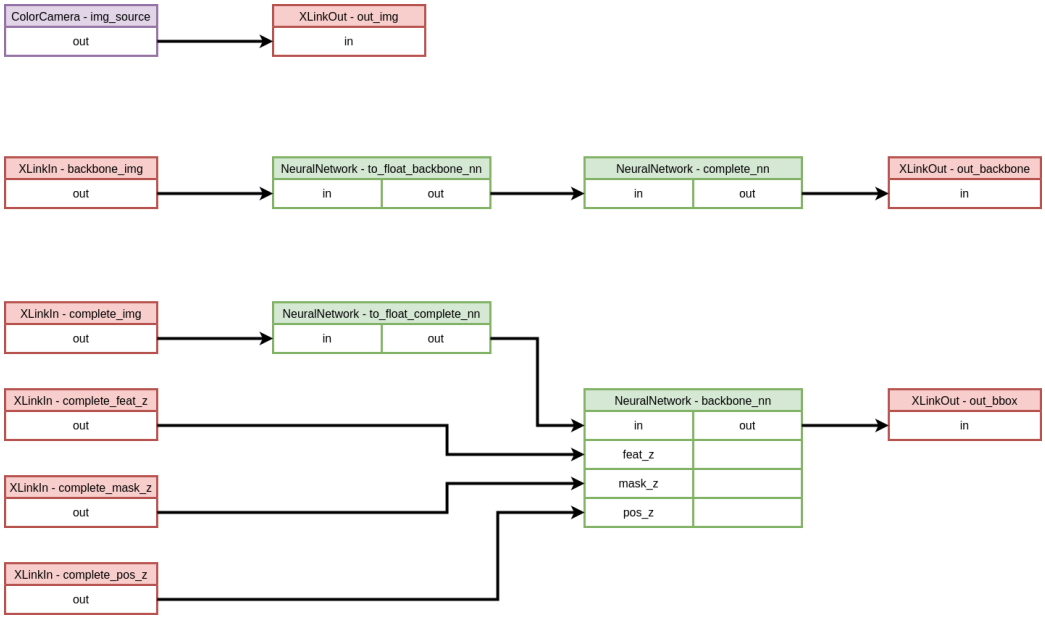
Zaradi naštetih značilnosti, se štiri različice cevovoda (*edge cam mode*, *edge synthetic mode*, *host cam mode* in *host synthetic mode*) med sabo nekoliko razlikujejo. Vseeno pa imajo nekaj skupnih elementov. Na slikah 4.3, 4.4, 4.5 in 4.6 so prikazani diagrami štirih cevovodov. V nadaljevanju bomo opisali vsakega od štirih načinov delovanja.



Slika 4.3: Diagram cevovoda *edge cam mode*, ki uporablja slikovni tok iz vgrajene kamere ter vse procesiranje opravi na vgrajeni napravi.



Slika 4.4: Diagram cevovoda *edge synthetic mode*, ki uporablja slikovni tok iz gostiteljskega sistema vendar vse procesiranje opravi na vgrajeni napravi.

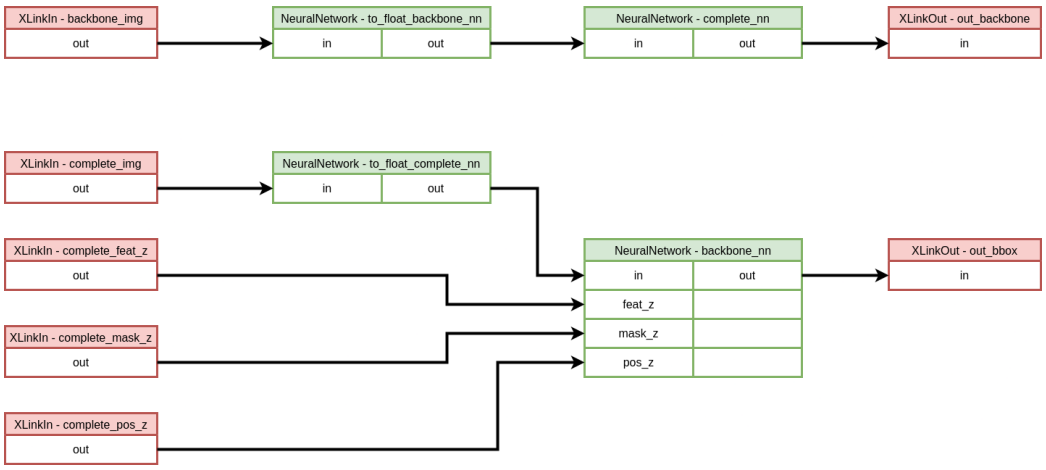


Slika 4.5: Diagram cevovoda *host cam mode*, ki uporablja slikovni tok iz vgrajene kamere vendar na vgrajeni napravi opravi le napoved z nevronsko mrežo.

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 prve 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 4.4, 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



Slika 4.6: Diagram cevovoda *host synthetic mode*, ki uporablja slikovni tok iz gostiteljskega sistema in na vgrajeni napravi opravi le napoved z nevronsko mrežo.

njegove resolucije.

Vezna logika opravlja nalogo usmerjnanja podatkov v pravilno vejo cevovoda (backbone ali complete), računanja in nastavljnanj 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 inicializirati (backbone), če pa tega podatka ni na vhdou se odvije korak sledenja (complete). V koraku inicilizacije se izvede del mreže poimenovan backbone, ki izračuna vektorje vhdone vdelave, pozicijskega kodiranja in maske, te podatke vezna logika znotraj vozlišča *Script* ponovno sprejme in ob vsakem naslednjem koraku posreduje v vejo *complete*. V vseh korakih sledenja vezakna logika posreduje slikovni element in vse tri vektoje pridobljene iz inicializacijeksega koraka v vejo cevovoda poimenovano *complete*. Ko vezna logika, iz veje *complete* 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 podat-

kovnem 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 4.6.

Edge cam mode

Ta način se od 4.6.1 razlikuje samo v tem, da slikovni tok prejme iz vgrajene kamere. Od gostiteljskega sistema pričakuje samo omejitveni okvir, v vseh nadaljnih korakih pa sledilnik samostojno sledi objektu in gostiteljskemu sisitemu vrača napovedan omejitveni okvir in sličico, ki jo je sledilnik pridobil iz vgrajene kamere. Diagram zgradbe cevovoda je prikazan na sliki 4.3.

Da bi omogočilo enostavno uporabo smo za oba tipa sledilnika, ki uporabljata podatke iz vgrajene kamere, razvili preprost uporabniški vmesnik. Uporabniški umesnik omogoča označevanje objekta in prikaz napovedanega omejitvenega okvirja. Celoten sistem je zasnovan na tak način, da lahko uporabnik v bilokaterem 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 z prejšnjima dvema načinoma (*edge*) precej bolj enostavna. Cevovod je razdeljen na dve popolnoma ne povezani veji *backbone* in *complete*. 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 *backbone*. V vseh nadaljnih korakih, ko se izvaja sledenje, pa gostitleljski sistem poleg sličice pošlje tudi 3 matrike, ki jih je sprejel v ini-

cializacijeksem koraku. Kot rezultat koraka sledenja pa gostiteljski sistem sprejme napovedan omejitveni okvir. Diagram zgradbe cevovoda je prikazan na sliki 4.6.

Host cam mode

V tem načinu sistem deluje popolnoma enako, kot v 4.6.1, 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 5

Evalvacija? Diskusija?

Prvi aspekt, ki smo ga evalvirali je hitrost sledilnika. Ugotovili smo, da v vseh štirih načinih delovanja sledilnik dosega porvpečno hitrost 10 FPS. Iz te ugotovitve lahko izpeljemo zaključek da je ozko grlo v sistemu izvajanje nevronske mreže. To smo zudi potrdili z izvajanjem meritev časa izvajanja posameznih delov cevovoda. Ugotovili smo, da korak v katerem se izvaja nevronska mreža *complete* traja v povprečju kar 68 ms. Vsi ostali koraki pa v povprečju doprinesejo še 30 ms. Seštevek vseh koraku pa znaša približno 100ms, kar je tudi v skladu z izmerjenimi 10 FPS.

Evalvirali smo tudi točnost sledilnika. Uporabili smo test VOT-ST2021 [8] saj je na tem testu bil leta 2021 testiran in primerjan model STARK_RT [15]. Na testu VOT-ST2021, je bila testirana različica modela STARK - STARK_RT, mi pa smo v tem delu uporabili poenostavljeno različico STARK Lightning. Na sliki 5.1 so prikazani rezultati testa VOT-ST2021, na sliki 5.2 pa rezultati, ki smo jih dosegli mi. Za Evalvacijo smo uporabili dve vrsti sledilnika, prvi je uporabljal 4.6.1 arhitekturo, drugi pa je bil pognan na osebnem računalniku v okolju *ONNX Runtime*. S tem smo hoteli preveriti razliko med modeli, ki spo bili prevedeni v OpenVINO format in so omejeni na 16-bitne vrednosti v plavajoči vejici in modelom, ki teh omejitev nima.

Če najprej primerjamo rezultate, ki smo jih dobili z uporabo sledilnika STARK Lightning 5.2 v primerjavi z STARK_RT 5.1, lahko opazimo, da

smo dobili polovico slabše rezultate, kar je bilo tudi pričakovano. Spodbudno je dejstvo, da četudi smo uporabili zelo okrnjeno različico modela, ki lahko popolnoma samostojno deluje na vgrjeni napravi, ne bi bili najslabši na testiranju.

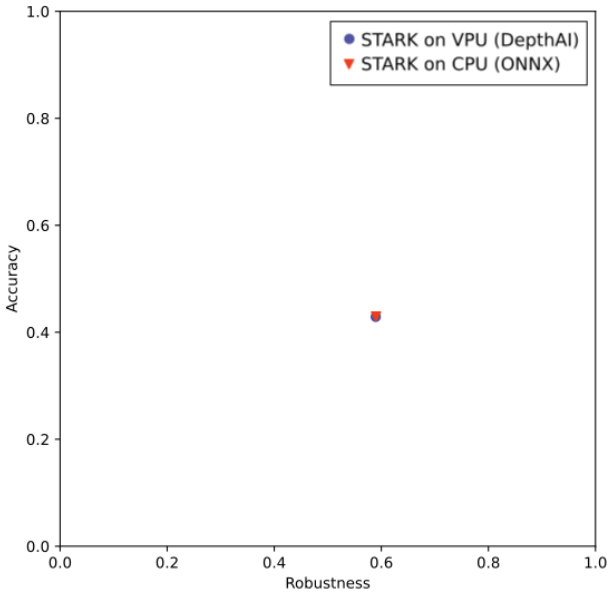
Zanimla nas je tudi primerjava med modelom, ki je optimiziran za delovanje na vgrajeni napravi in navadno implementacijo modela, ki se lahko izvaja znotraj okolja ONNX. Iz slik 5.3 in 5.4 lahko opazimo, da je delovanje sledilnika v obeh primerih identično. Iz tega lahko sklepamo, da se med optimiziranjem in prevajanjem model v OpenVINO format, ni izgubila nobena informacija, ki bi lahko vplivala na delovanje modela. Sklepamo pa lahko tudi to da zmanjšanje resolucije podatkovnega tipa iz 32 bitne plavajoče vejice v 16 bitno plavajočo vejico, ne vpliva na kvaliteto modela.

		baseline			realtime		unsupervised
Tracker	EAO	A	R	EAO	A	R	AUC
RPTMask	0.568①	0.764③	0.859③	0.368	0.659	0.704	0.683①
CFRPT	0.551②	0.745	0.853	0.325	0.612	0.684	0.655③
TransT_M	0.550③	0.742	0.869①	0.550①	0.742	0.869①	0.670②
TregPlus	0.546	0.753	0.852	0.440	0.706	0.777	0.615
DualTFRon	0.539	0.757	0.837	0.395	0.681	0.741	0.629
F_TregPlus	0.537	0.753	0.848	0.490	0.738	0.812	0.626
DualTFRst	0.536	0.755	0.836	0.512③	0.751②	0.816	0.623
STARK_RT	0.534	0.781①	0.830	0.531②	0.780①	0.829③	0.631
DualTFR	0.527	0.748	0.826	0.509	0.746	0.813	0.619
RPT	0.524	0.692	0.866②	0.346	0.598	0.721	0.620
SiamUSCP	0.515	0.696	0.854	0.469	0.679	0.821	0.601
D3Sv2	0.514	0.712	0.843	0.313	0.654	0.614	0.607
LWL_B2S	0.511	0.729	0.826	0.475	0.719	0.806	0.602
TransT	0.507	0.748	0.817	0.507	0.748③	0.817	0.612
TRASFUSTm	0.506	0.738	0.823	0.414	0.687	0.754	0.621
fRPT	0.501	0.691	0.851	0.486	0.689	0.840②	0.596
AlphaRef	0.484	0.752	0.776	0.477	0.745	0.776	0.608
RPT_AR	0.474	0.710	0.790	0.293	0.576	0.632	0.565
SuperDiMP_AR	0.473	0.725	0.772	0.370	0.659	0.702	0.580
LWL	0.467	0.719	0.800	0.421	0.699	0.772	0.583
SiamUSC	0.464	0.694	0.798	0.460	0.694	0.792	0.566
SAMN_DiMP	0.463	0.703	0.776	0.391	0.673	0.716	0.510
keep_track	0.462	0.725	0.773	0.331	0.619	0.660	0.573
SAMN	0.457	0.723	0.774	0.439	0.698	0.770	0.537
KYS_AR	0.455	0.724	0.755	0.390	0.684	0.704	0.527
RTT	0.450	0.767②	0.727	0.387	0.697	0.696	0.610
D3S	0.443	0.700	0.767	0.432	0.694	0.756	0.505
DiMP_AR	0.432	0.717	0.722	0.415	0.710	0.713	0.558
PrDiMP_AR	0.425	0.724	0.722	0.387	0.691	0.693	0.552
ATOM_AR	0.409	0.711	0.707	0.387	0.707	0.677	0.537
SiamEM_R	0.398	0.738	0.681	0.393	0.737	0.675	0.526
TRATMask	0.395	0.632	0.757	0.364	0.621	0.728	0.341
DCDAAR	0.355	0.709	0.625	0.352	0.710	0.619	0.419
STM	0.311	0.739	0.593	0.285	0.698	0.570	0.457
ACM	0.304	0.479	0.766	0.294	0.478	0.746	0.392
KYS	0.282	0.454	0.758	0.265	0.447	0.732	0.370
PrDiMP	0.281	0.470	0.745	0.274	0.469	0.734	0.401
NSpaceRDAR	0.271	0.456	0.721	0.269	0.455	0.723	0.351
DiMP	0.270	0.449	0.736	0.266	0.451	0.722	0.374
ATOM	0.261	0.452	0.711	0.251	0.451	0.686	0.376
deepmix	0.239	0.436	0.666	0.223	0.417	0.641	0.324
SION	0.232	0.434	0.634	0.232	0.434	0.634	0.297
ReptileFPN	0.213	0.423	0.619	0.119	0.384	0.317	0.275
TCLCF	0.200	0.425	0.588	0.200	0.425	0.588	0.247
ASMS	0.196	0.419	0.569	0.196	0.419	0.567	0.257
FSC2F	0.193	0.412	0.567	0.182	0.410	0.539	0.259
VITAL++	0.187	0.366	0.632	0.085	0.334	0.234	0.232
CSRDCF	0.186	0.403	0.563	0.184	0.402	0.557	0.229
SiamFC	0.173	0.408	0.499	0.168	0.417	0.476	0.220
ANT	0.172	0.398	0.485	0.143	0.386	0.405	0.228
KCF	0.168	0.421	0.489	0.168	0.421	0.490	0.165
LGT	0.133	0.335	0.448	0.104	0.335	0.325	0.173
L1APG	0.083	0.359	0.222	0.073	0.370	0.165	0.102

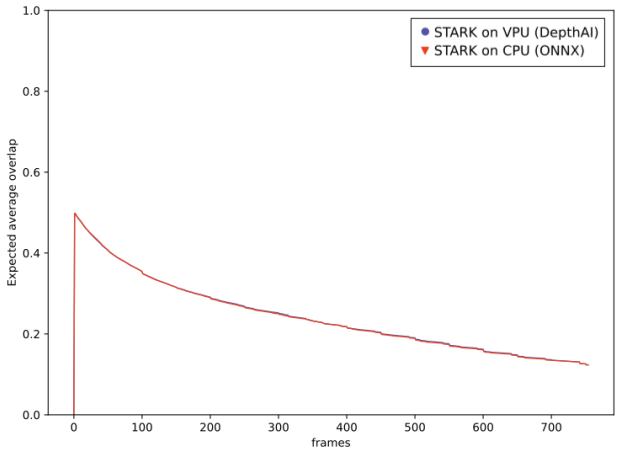
Slika 5.1: Slika prikazuje rezultate testa VOT2021, na eksperimentu *baseline* in *realtime* [8].

	baseline		
	EAO analysis	AR Analysis	
Trackers	EAO	A	R
● Stark on VPU (depthai)	0.213	0.429	0.590
▼ Stark on CPU (ONNX)	0.212	0.429	0.590

Slika 5.2: Slika prikazuje rezultate testov, ki smo jih izvedli nad našo implmentacijo sledilnika.



Slika 5.3: Slika prikazuje retultat testa VOT-ST2021 uporabljenih sledilnikov. Graf prikazuje povprečno natančnost v odvisnosti od povprečne robustnosti.



Slika 5.4: Slika prikazuje retultat testa VOT-ST2021 uporabljenih sledilnikov. Graf prikazuje pričakovano povprečno prekrivanje omejitvenih okvirjev skozi čas.

Poglavje 6

Zaključek

V diplomskem delu smo implementirali in ovrednotili delovanje modernega vizualnega sledilnika STARK. Sledilnik deluje na osnovi umetnih nevronskih mrež, bolj natančno umethinh nevronskih mrež z arhitekjturo transformer. Med pregledom objavljenih del na temo vizualnega sledenja na vrgajenih napravah nismo našli nobenega, ki bi uporabil sledilnika z najmodrenejšo arhitekturo - transformer. Cilj dela je bil preučiti možnosti take implementacije, sama implementacija ter ovrednotenje delovanja sledilnika. Sledilnik smo impmementirali na vgrjanei 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 potrbno zasnovati cevovod, ter vanga vmestiti 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 4.6.1. Način delovanja, ki bi bil najbolj uporaben v realnem svetu 4.6.1, 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 bilokaterem trenutku označi novo tarčo za sledenje.

Največja slabost naše implementacije je hitrost delovanja. Sledilnik deluje z povprečno 10 FPS, kar je premalo, da bi ga lahko uvrstili med sledilnike, ki delujejo v realnem času. Ugotovili smo, da je ozko grlo v sistemu ravno napovedovanje z nevronsko mrežo, saj v povprečju traja kar 68 ms, vse ostalo procesiranje pa doprinese se 30ms. V tej smeri je odrpto še veliko možnosti za optimizacijo in predelave modela, ki bi lahko skrajšalo čas izvajanja.

Članki v revijah

[4] 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] 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.

[7] 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.

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

https://www.overleaf.com/project/609ce2055f917cb2f776732e

Članki v zbornikih

[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.

[5] 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.

[10] 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.

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

[15] 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.

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] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] Matej Kristan in sod. *The Tenth Visual Object Tracking VOT2022 Challenge Results*. 2022.

[10] 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.

[11] *Luxonis*. URL: https://www.luxonis.com/ (pridobljeno 2023).

[12] Nisarg Patel. *What Is Deep Learning?* URL: https://medium.com/@nnpatel4583/what-is-deep-learning-4daa22ceea4e (pridobljeno 2023).

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

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

[15] 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.