Uporaba naprednih modelov za detekcijo objektov na robnih napravah

Jure Špeh

Mentor: as. dr. Goran Andonovski

Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška cesta 25, 1000 Ljubljana, Slovenija

js0020@student.uni-lj.si, goran.andonovski@fe.uni-lj.si

Using advanced models for object detection on edge devices

Object detection at the edge is becoming an important building block of modern control systems. Advanced algorithms based on convolutional neural networks are not suitable for direct use on devices with limited computing power. In this paper, we present some of the more important algorithms for object detection, in particular the YOLO algorithm and its variant YOLOv7. The process of deploying the model on an Nvidia Jetson Nano microcomputer and the optimisation of the model using the TensorRT and DeepStream tools is presented. Based on the experiment, we evaluate the comparison of detection speed and accuracy between different implementations of the YOLOv7 *Tiny* model. The paper also includes a repository of software code that provides a basis for further development of applications with advanced object detection algorithms on Nvidia edge devices.

Kratek pregled prispevka

Detekcija objektov na robnih napravah postaja pomemben gradnik sodobnih sistemov vodenja. Napredni algoritmi, ki temeljijo na konvolucijskih nevronskih mrežah niso primerni za neposredno uporabo na napravah z omejeno računsko zmogljivostjo. V tem članku predstavljamo nekatere pomembnejše algoritme za detekcijo objektov, zlasti algoritem YOLO in njegovo različico YOLOv7. Prikazan je postopek namestitve modela na mikroračunalnik Nvidia Jetson Nano. Predstavljen je postopek optimizacije modela z orodjema TensorRT in DeepStream. Na podlagi preizkusa ovrednotimo primerjavo hitrosti in natančnosti detekcije med različnimi implementacijami modela YOLOv7 *Tiny*. V okviru članka je izdelan tudi repozitorij s programsko kodo, ki podaja osnovo za nadaljnji razvoj aplikacij z naprednimi algoritmi za detekcijo objektov na rabnih napravah Nvidia.

1 Uvod

V sodobnih sistemih vodenja se vse bolj uporabljajo napredni modeli za detekcijo objektov. Najdemo jih v različnih aplikacijah varnostnih sistemov, avtonomnih vozil, industrijskih obratov in robotov. Z razvojem vse močnejših kompaktnih mikroračunalnikov se obdelava seli na rob omrežja. To omogoča večjo avtonomnost, hitrejše odločanje, nižje stroške prenosa in hrambe podatkov ter več zasebnosti.

Ker so viri robnih naprav omejeni, so potrebne določene omejitve pri uporabi naprednih modelov. V tem članku bomo predstavili najnovejše tehnike in izzive pri uporabi naprednih modelov za detekcijo objektov na podlagi računalniškega vida.

Najprej bomo predstavili osnove konvolucijskih nevronskih mrež, na katerih temeljijo napredne metode za detekcijo objektov. Na kratko bomo opisali delovanje dvostopenjskih modernih detektorjev objektov R-CNN [1], Fast R-CNN [2] in enostopenjske skupine algoritmov YOLO [3].

V nadaljevanju bomo govorili o uporabi modelov na robnih napravah. Zaradi omejenih zmogljivosti je pomembna optimizacija modelov za namensko strojno opremo. Na praktičnem primeru aplikacije za detekcijo objektov z uporabo robne naprave Nvida Jetson Nano bomo prikazali namestitev in uporabo naprednega modela YOLOv7 [4], vključno z metodami za optimizacijo zmogljivosti, učinkovitosti in zmanjšanje velikosti modela.

2 Detekcija objektov

Osnovne nealoge v računalniškem vidu so klasifikacija, lokalizacija, detekcija in segmentacija. Klasifikacija vsaki sliki dodeli enega oz. več razredov, ki se na njej nahajajo. Pove nam kateri objekti se na sliki nahajajo. Lokalizacija poda položaj objekta na sliki. Detekcija objektov je sestavljena iz klasifikacije in lokalizacije. Poda nam mejni kvader (okvir okrog objekta) in ime razreda za vsak razpoznan objekt na sliki. Segmentacija v osnovi dodeli razred vsakemu pikslu na sliki in je računsko zahtevnejša naloga.

3 Algoritmi za detekcijo objektov

Algoritme za detekcijo v grobem delimo na dve vrsti:

1. Tradicionalni algoritmi (temeljijo na izbiri značilk): Viola-Jones, HOG, DPM,...
2. Algoritmi na osnovi nevronskih mrež (samodejna izbira značilk glede na oznake slik): CNN, Faster R-CNN, YOLO, SSD,...

V članku bomo na kratko predstavili strukturo in delovanje naprednih algoritmov na osnovi nevronskih mrež. Podrobneje bomo predstavili delovanje in uporabo modela YOLOv7 *Tiny* na robni napravi.

3.1 Konvolucijske nevronske mreže

V tem članku se bomo osredotočili na napredne algoritme za detekcijo objektov na osnovi konvolucijskih nevronskih mrež (CNN). Uporabljajo se za reševanje kompleksnih vizualnih nalog, razpoznavanje zvoka in procesiranje naravnega jezika. CNN je sestavljena iz konvolucijskih, združevalnih (ang. *pooling*) in polno-povezanih (ang. *fully connected*) slojev. Konvolucijski sloji so namenjeni iskanju značilk s pomočjo različnih jeder (ang. *kernel*) oz. filtrov (ang. *filters*). Izhod sloja v katerem vsi nevroni uporabljajo isti filter (npr. vertikalni filter) je zemljevid značilk. Med procesom učenja se konvolucijski sloji samodejno naučijo najbolj ustreznih filtrov za dano nalogo. Vsak nadaljnji konvolucijski sloj združi filtre v bolj kompleksne vzorce. Nad rezultatom konvolucije se izvede korigirana linearna aktivacijska funkcija (ReLU), prikazana z enačbo (1). Združevalni sloj je namenjen zmanjševanju števila parametrov. Polno-povezani sloj je namenjen klasifikaciji.

CNN same po sebi ne detektirajo rotacije in skaliranja objektov. Pred učenjem CNN je potrebno pripraviti učno množico, ki že vsebuje rotirane oz. skalirane vzorce.

$$f(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases} \tag{1}$$

3.2 *Moderni detektorji objektov*

Moderne algoritme za detekcijo delimo na enostopenjske (ang. *single-stage*) in dvostopenjske (ang. *two-stage*). V splošnem so enostopenjski algoritmi hitrejši, dvostopenjski pa so natančnejši pri lokalizaciji in razpoznavanju [5].

Najprej so se razvili dvostopenjski algoritmi, ki so sestavljeni iz faze predloga regij in klasifikacijske faze. Ti modeli na podlagi referenčnih polj oz. sider (ang. *anchors*) najprej predlagajo kandidate za objekte, ki jim pravimo tudi regije interesa (ang. *region of interest, RoI*). V drugem koraku sledi razvrstitev predlogov in izboljšava lokalizacije znotraj predlaganih regij. Rezultat klasifikacije so verjetnosti za pripadnost posameznemu razredu, in parametri mejnega kvadra: x in y pozicija spodnjega levega kota kvadra ter širina in višina kvadra (x,y,w,h). Koordinate mejnega kvadra se pridobi z uporabo L2 mere, pri klasifikaciji pa se uporablja funkcija *softmax*.

Pri enostopenjskih algoritmih se klasifikacija in lokalizacija izvedeta neposredno na celi sliki v enem koraku.

3.3 *Pregled naprednih metod za detekcijo*

Regijska konvolucijska mreža (R-CNN) [1] je prvi dvostopenjski detektor. Arhitektura je sestavljena iz treh delov. V prvem so avtorji za generiranje približno 2000 predlogov regij uporabili algoritem selektivnega iskanja. Vsaka regija se transformira v sliko fiksne velikosti (227x227) z uporabo affine transformacije. V drugem je uporabljena CNN s petimi konvolucijskimi sloji in dvema popolnoma-povezanima slojema. Izhod je vektor značilk fiksne dolžine za vsako predlagano regijo. CNN zahteva fiksno velikost vhodne slike. V tretjem delu se izvede klasifikacija vsake regije z uporabo metode SVM in lokalizacija (regresija) mejnih kvadrov. Drugi in tretji del skupaj predstavljata drugi korak detekcije.

Zaradi večnivojskega učenja, prostorske in časovne kompleksnosti ter počasnega detektiranja R-CNN, so avtorji model modificirali in tako izdelali Fast R-CNN [2]. Slika gre samo enkrat skozi CNN, ki generira zemljevid značilk. Nanj so nato projecirani predlogi regij. Ta korak se imenuje združevanje interesnih območij, ki pretvori neko območje iz zemljevida značilk v fiskno velikost (h x w) z uporabo *max poolinga* na sorazmernem številu pikslov. Izhodi združevanja RoI (vektorji enakih velikosti) so nato prenesejo v polno-povezane sloje.

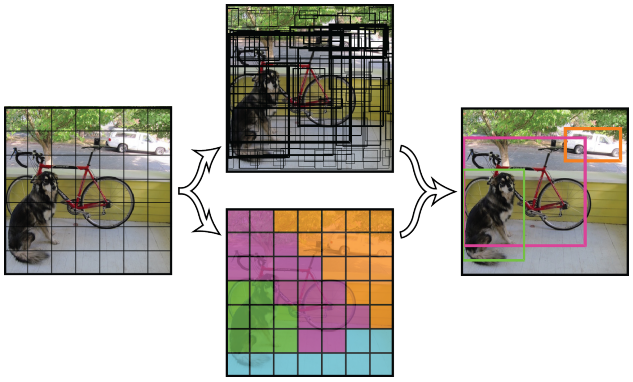
YOLO (ang. *You Only Look Once*) je najbolj znan enostopenjski algoritem za detekcijo objektov, ki je sestavljen iz ene same CNN. V prvi različici [3] so avtorji detekcijo objektov definirali kot regresijski problem s prostorsko ločenimi mejnimi kvadri in pripadajočimi verjetnostmi. Pred tem se je detekcijo objektov definiralo kot klasifikacijsko nalogo. Prvotni različici modela so sledile številne različice, ki so detekcijo še izboljšale.

Glavne prednosti YOLO modela so hitrost (preprostejša arhitektura od Faster R-CNN), kontekst (upošteva celotno sliko - razpoznavanje objektov v kontekstu) in posplošenost (odporen na nove, nepričakovane slike).

Glavna slabost modela je v natančnosti pri lokalizaciji manjših objektov. V tem članku smo model YOLO oz. njegovega naslednika različico YOLOv7 izbrali zaradi hitrosti in manjše porabe računskih virov.

Model za razpoznavanje uporablja značilke iz celotne slike. Pri tem napoveduje vse mejne kvadre preko vseh razredov objektov hkrati. To lahko doseže z razdelitvijo vhodne slike na mrežo velikosti *SxS*. Za vsako celico napove *B* mejnih kvadrov s petimi pripadajočimi parametri: center kvadra relativno na meje celice (b_x, b_y), velikost kvadra relativno na velikost slike (b_w, b_h) in oceno zaupanja (ang. *confidence score*), da kvader vsebuje objekt. Ocena zaupanje predstavlja IoU med napovedanim in pravilnim (ang. *ground truth*) kvadrom. Vsaka celica napove C pogojnih verjetnosti, da objekt pripada določenemu razredu. Delovanje modela je prikazano na sliki 1.

Če se v celici nahaja objekt bi morali zaupanje biti enako preseku regij - IOU (ang. *intersection over union*) med napovedanim okvirjem in resničnim objektom. Če se v celici ne nahaja noben objekt, je vrednost gotovosti enaka nič. Vsaka ce-



Slika 1: Prikaz delovanja algoritma YOLO. [4]

lica napove tudi C pogojnih verjetnosti razredov Pr(razred|objekt).

3.4 YOLOv7

YOLOv7 je ena izmed novejših različ modelov iz skupine YOLO. V članku [4] so opisane številne kompleksne novosti, ki izboljšujejo hitrost in natančnost detekcije. V našem projektu smo ga izbrali, ker je zaradi novosti in omenjenih prednosti v času pisanja veljal za najhitrejši in najbolj natančen realno-časovni detektor. Med glavnimi spremembami glede na prejšnje algoritme YOLOv7 sta dve arhitekturni spremembi in dve dodatni spremembi. Ogrodje, ki ga uporablja je E-ELAN, pri katerem je zemljevid značilk bolj robusten. Druga sprememba je skaliranje modela, ki ima vlogo prilagajanja atributov za generiranje različno skaliranih modelov (npr. YOLOv7 *Tiny*, YOLOv7, YOLOv7X). Dodatne spremembe so metode, ki izboljšajo zmogljivost modela brez povečanja cene učenja. Taka sprememba je reparametrizacija modela, ki se deli na nivo modela in nivo modula:

- Reparametrizacija modela: Uporaba različnih učnih podatkov ob enakih nastavitvah za učenje večih modelov. Nato sledi povprečenje uteži za izdelavo končnega modela. Drug način je uporaba povprečja uteži ob različnih epohih.
- Reparametrizacija modula: Proces učenja se porazdeli v več modulov, izhodi so združeni v en končni model.

YOLOv7 vsebuje več glav (ang. *head*). Glavna glava (ang. *lead head*) je odgovorna za končni izhod. Pomožna glava (ang. *auxiliary head*) pa pomaga pri učenju srednjih slojev.

4 Uporaba modelov na robnih napravah

V tem članku se bomo osredotočili na robne naprave, ki se uporabljajo v namene avtomatizacije, robotike in vodenja sistemov, izpustili bomo mobilne naprave. Pri izbiri ustrezne robne naprave je potrebno upoštevati naslednje vidike: *računsko moč*, *velikost pomnilnika*, *porabo energije*, *velikost* in *povezljivost*.

Najbolj pogoste robne naprave, ki se uporabljajo za namen detekcije so:

- **NVIDIA Jetson Xavier NX** (2020): (cena) 459 EUR, (računska moč) 32 TOPS
- **Coral Dev Board** (2019): 150 EUR, 32 GFLOPS
- **Intel Movidius Neural Compute Stick 2** (2018): 80 EUR, 4 TOPS
- **Raspberry Pi 4 4GB** (2019): 70 EUR, 13,5-32 GFLOPS
- **Nvidia Jetson Nano** (2019): 150 EUR, 472 GFLOPS / 1,1 TOPS

Med pregledom trga smo ugotovili, da med napravami ni velike izbire, sploh med vstopnimi modeli (cena pod 200 EUR). Za naš projekt smo izbrali mikroračunalnik Nvidia Jetson Nano, ki predstavlja najboljše razmerje med računsko močjo in ceno.

4.1 Pregled orodij za optimizacijo modelov na napravah Nvidia

Optimizacija naprednih modelov je pomemben del aplikacij za detekcijo na robu. Navadno želimo z modeli na robnih napravah doseči delovanje v realnem času, da lahko z dobljenimi rezultati vplivamo na sisteme vodenja ali zmanjšamo pretok surovih podatkov na strežnike.

V tem podpoglavju bomo predstavili nekatera orodja, ki so pomembna na področju optimizacije modelov na robnih napravah.

Osnovni model je navadno napisan v enem iz-

med vodilnih ogrodij za strojno učenje: TensorFlow, Keras ali PyTorch. Za uporabo na robnih napravah obstajajo manjše različice (npr. *TensorFlow lite*), ki skrči TensorFlow model in ga pripravi za uporabo na mobilnih oz. robnih napravah.

ONNX (ang. *Open Neural Network Exchange*) je odprtokodni standard za predstavitev modelov globokega učenja. Namenjen je enostavni pretvorbi med različnimi ogrodji za strojno učenje in prenosljivosti med različnimi strojnimi opremami, neodvisno od proizvajalca. Ob pretvorbi je mogoča tudi optimizacija modela za hitrejše delovanje in manjšo porabo računskih zmogljivosti.

Druga možnost optimizacije je pretvorba v pogon TensorRT (ang. *TensorRT engine*). Gre za knjižnico, ki optimizira modele strojnega učenja za uporabo na GPU-jih Nvidia. Navadno se uporablja za razpoznavanje (ang. *inference*) v realnem času. TensorRT poenostavi sloje v enostavnejšo strukturo (FLOPS), ki se porazdeli med več jeder CUDA. Pri pretvorbi v model TensorRT, vzamemo naučen model in ga optimiziramo za točno določeno strojno opremo, v našem primeru mikroračunalnik Jetson Nano.

Nvidia DeepStream SDK je orodje za izdelavo visoko zmogljivih cevovodov za procesiranje video gradiva. Uporablja se na napravah s procesorji Nvidia in med drugim vključuje orodja in knjižnice za detekcijo, klasifikacijo in sledenje objektom. DeepStream optimizira celoten cevovod (ang. *pipeline*), porabo spomina in zmogljivost. Znotraj cevovoda DeepStream lahko uporabimo model TensorRT, ki v obliki vtičnika opravlja nalogo detekcije objektov.

5 Uporaba in optimizacija modela YOLOv7 *Tiny* na robni napravi

Uporabo naprednih modelov bomo predstavili na primeru aplikacije detekcije objektov v prometu. Kot strojno opremo smo uporabili mikroračunalnik Nvidia Jetson Nano. Nanj smo namestili prilagojen operacijski sistem Ubuntu 18.04LTS. Sistem smo testirali na video posnetkih prometa, mogoč pa bi bil tudi zajem slike v realnem času z uporabo kamere preko vodila CSI.

5.1 Model za detekcijo

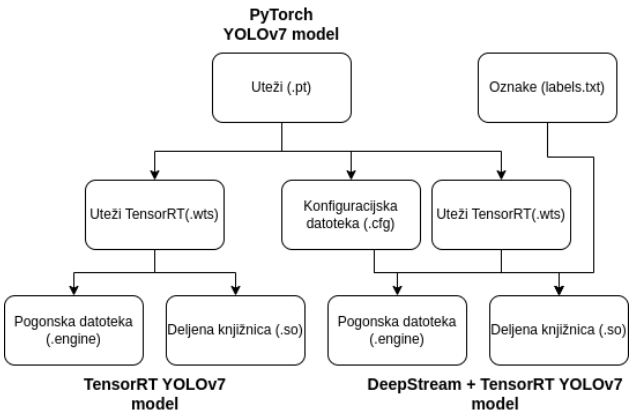
Kot osnovni model za detekcijo smo izbrali YOLOv7, različico *Tiny*. Gre za pomanjšano verzijo modela, ki ima manj konvolucijskih slojev in je optimizirana za naprave z omejeno računsko zmogljivostjo. Različica Tiny je sicer manj natančna pri detekciji, vendar je hitrost glede na preizkus nekajkrat večja. Prav tako zasede manj prostora v pomnilniku.

Namestitve modela smo se lotili v treh korakih. Najprej smo na Jetson Nano klonirali osnovni repozitorij modela YOLOv7 in prenesli ustrezne pred-naučene uteži. Ker virov o pravilni namestitvi in zagonu modela na napravi Jetson Nano ni bilo mogoče najti, smo izdelali podrobna navodila za namestitev vseh potrebnih knjižnic [6]. Model temelji na knjižnicah PyTorch in TorchVision, ki ju je potrebno namestiti v določenem zaporedju, vključno s pripadajočimi odvisnostmi (ang. *dependencies*).

Preizkusili smo osnovni model in njegovo manjšo različico *Tiny*. Ugotovili smo, da osnovni model ni primeren za uporabo na napravah kot je Jetson Nano, saj je prevelik in računsko prezahteven (različica Jetson Nano 4GB je imela pri tem povsem poln pomnilnik). Različica *Tiny* je delovala z okrog 9 FPS. Modeli PyTorch so zaradi številnih vmesnikov uporabni za raziskovanje in testiranje, vendar imajo omejeno zmogljivost za delovanje na grafičnih pogonih (GPU). Za pohitritev delovanja je potrebno model pretvoriti v obliko, ki je optimizirana za namensko strojno opremo.

V drugem koraku smo model optimizirali z uporabo pogona TensorRT. Pri pretvorbi smo uporabili TensorRTx repozitorij [7]. Osnovne uteži v formatu PyTorch (datoteka .pt) so v pogonsko datoteko TensorRT pretvorili po postopku, ki je prikazan na sliki 2. Za preizkus rezultatov smo prilagodili programsko skripto v jeziku Python, ki ji podamo pot do video posnetka, pogonsko datoteko in deljeno knjižnico (datoteka .so). Med delovanjem s pomočjo knjižnice OpenCV prika-

zujemo sliko in hitrost razpoznavanja v FPS (slik na sekundo). Zajem zaslona se nahaja na sliki 3.



Slika 2: Postopek pretvorbe v obliko TensorRT oz. DeepStream.

Slika 3: Rezultat aplikacije z YOLOv7 Tiny in TensorRT.

V tretjem koraku smo skupaj z modelom, ki je temeljil na TensorRT uporabili ogrodje Nvidia DeepStream. Ker se knjižnica TensoRT osredotoča zgolj na nizkonivojsko optimizacijo modela, sama po sebi ni povsem optimalna za uporabo v končnih aplikacijah. Poleg zamud, ki jih prinaša razpoznavanje moramo računati tudi na pred- in po-procesiranje videa. Zato smo v tem koraku združili prednosti modela na osnovi TensorRT in ogrodja DeepStream.

Namestitev, konfiguracija in uporaba ogrodja DeepStream na Jetson Nano je v dokumentaciji in drugih virih slabo podprta. Svojo rešitev smo izdelali predvsem s prilagoditvijo obstoječih projektov, ki so javno dostopni na spletni strani GitHub. Za namestitev ogrodja DeepStream na Jetson Nano smo sledili navodilom [8]. Uporabili smo različico 6.0.1, ki zadnja še podprta s strani Nvidie za namestitev na Jetson Nano. Dodatno smo namestili tudi vezave Python (ang. *Python bindings*). Te so uporabljene za dostop do nižjenivojskih ukazov, ki so napisani v programskih jezikih C/C++ in CUDA. Pri izdelavi cevovoda DeepStream in aplikacije v Pythonu nam je bil v pomoč predvsem projekt [9], ki prikazuje izdelavo osnovnega cevovoda z uporabo ogrodja DeepStream skupaj z uporabo vezav Python. Omenjeni primer je namenjen uporabi z zmogljivimi grafičnimi karticami Nvidia, zato smo ga prilagodili, da je mogoč pogon na napravi Jetson Nano. Izvorna koda celotnega projekta skupaj z navodili za izvedbo je dostopna na [10]. Vključeni so vsi trije koraki za namestitev in preizkus delovanja.

Pri uporabi orodja DeepStream SDK je potrebno model YOLO pretvoriti v obliko vtičnika. V ta namen smo uporabili orodje [11]. Pretvorbo izvedemo po postopku na sliki 2. Deljeno knjižnico prevedemo z uporabo ustrezne različice CUDA.

Za uporabo vtičnika in ostalih datotek v kodi Python je potrebno napisati konfiguracijsko datoteko s končnico .txt. V datoteki navedemo pot do konfiguracijske datoteke za naš model, datoteke s parametri, pogonske datoteke, oznak in deljene knjižnice. Če ob zagonu programa *run.py* že obstaja pogonska datoteka, se začne pretok videa z detekcijo on sledenjem objektom preko protokola RTSP ali shranjevati video na disk. Če datoteka še ni ustvarjena, jo vtičnik najprej zgenerira in nato nadaljuje z izvajanjem detekcije ter sledenja. Video posnetek z detekcijo v realnem času lahko spremljamo s pomočjo orodij za dekodiranje protokola RTSP.

V aplikaciji DeepStream smo uporabili tudi sledenje na podlagi sledilnega algoritma DeepSORT, ki že obstaja v obliki vtičnika DeepStream.

5.2 *Rezultati testiranja*

Pri testiranju smo ugotovili, da na hitrost procesiranja (ang. *inference time*) močno vpliva velikost vhodne plasti v model (velikost vhodne slike). Pri vhodni velikosti 640x640 je bila povprečna hitrost obdelave skoraj polovico počasnejša, kot pri velikosti 416x416. Zmanjšanje vhodne velikosti izboljša čas procesiranja, vendar na račun slabše natančnosti. Manjša vhodna slika pomeni, da so objekti na sliki manjši in ne predstavljajo dovolj detajlov, da bi jih model lahko razpoznal. Rezultat testiranj hitrosti med različnimi nivoji optimizacije je predstavljen v tabeli 1, kjer Yv7t FPS predstavlja hitrost detekcije v slikah na sekundo, sum(N) pa predstavlja skupno število detektiranih objektov na video posnetku. Število detektiranih objektov smo dobili tako, da smo sešteli vse detektirane objekte na vseh slikah posnetka. Vidi se, da pri hitrosti razpoznavanja med optimizacijama s TensorRT in DeepStream ni velike razlike. To je pričakovano saj oba pristopa temeljita na izdelavi pogonske datoteke in namensko prevedene knjižnice, ki sta optimizirani za uporabo na naši strojni opremi. Kljub navidezno boljšim rezultatom optimizacije s TensorRT, ki je tudi enostavnejša za izvedbo, pa v omenjeni tabeli niso zajeti časi pred- in po-procesiranja slike. Ti časi so veliko nižji pri uporabi ogrodja DeepStream, zato je slika v tem primeru bolj tekoča. V vseh primerih je model z manjšo vhodno sliko detektiral manj objektov. Model z uporabo DeepStream-a je pri testiranju detektiral najmanj objektov. To pripisujemo različnim nastavitvam modela in hkratni uporabi sledilnega algoritma, zato neposredna primerjava ni povsem mogoča.

Tabela 1: Primerjava med različnimi implementacijami modela YOLOv7 Tiny.

	Yv7t FPS	**sum(N)**
Osnovni (416x416)	16	13685
Osnovni (640x640)	9	18679
TensorRT (416x416)	33	15517
TensorRT (640x640)	16	20152
DeepStream (416x416)	30	11634
DeepStream (640x640)	16	16188

Kot izboljšavo predlagamo izdelavo vsebnika Docker z vsemi komponentami (DeepStream, PyTorch, TorchVision, PyCuda). To bi pohitrilo namestitev razvojnega oz. produkcijskega okolja in poenostavilo razvoj aplikacij. V nadaljevanju projekta bo potrebno implementirati tudi štetje objektov. To bo storjeno z uporabo navidezne črte, ki jo bo uporabnik določil v uporabniškem vmesniku. Dodatna izboljšava je tudi možnost definiranja smeri štetja.

6 Zaključek

V članku smo predstavili uporabo naprednih modelov za detekcijo objektov na robnih napravah. V našem primeru smo uporabili enega izmed novejših algoritmov YOLOv7 in ugotovili, da je za uporabo na robnih napravah primerna različica *Tiny*. Ugotovili smo, je potrebno modele na robnih napravah optimizirati z orodji, ki so na voljo za dano strojno opremo. V našem primeru smo prikazali kako postaviti osnovo aplikacije za detekcijo objektov v prometu v realnem času. Jetso Nano se je izkazal kot mejno primerna strojna oprema. Zadovoljive rezultate je dosegel v primeru uporabe poenostavljenih modelov. Rezultate smo izboljšali z uporabo optimizacijskih orodij Nvidia TensorRT in DeepStream. Pri delu smo se soočali s pomanjkanjem dokumentacije za delo z mikroračunalnikom Jetson Nano. Zato smo izdelali javno dostopen vodič za namestitev algoritma YOLOv7 na Jetson Nano, ki se z nekaj prilagoditvami lahko uporabi tudi za ostale napredne modele. Dodatno smo objavili tudi predlogo aplikacije, ki jo je mogoče nadgraditi v različne namene detekcije in štetja objektov v realnem času.

7 Literatura

Literatura

[1] R. Girshick, J. Donahue in sod. Rich feature hierarchies for accurate object detection and semantic segmentation. V *2014 IEEE Conference on Computer Vision and Pattern Recognition*, str. 580–587. 2014.

[2] R. Girshick. Fast r-cnn. V *2015 IEEE International Conference on Computer Vision (ICCV)*, str. 1440–1448. 2015.

[3] J. Redmon, S. Divvala in sod. You only look once:

Unified, real-time object detection, 2015.

[4] C.-Y. Wang, A. Bochkovskiy in H.-Y. M. Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022.

[5] M. Carranza-García, J. Torres-Mateo in sod. On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data. *Remote Sensing*, zv. 13, št. 1, 2021.

[6] J. Špeh. Deploy yolov7 to jetson nano for object detection. Dosegljivo: `https://www.hackster.io/spehj/deploy-yolov7-to-jetson-nano-for-object-detection-6728c3`. [Dostopano: 17. 3. 2023].

[7] W. Xinyu. Implementation of popular deep learning networks with tensorrt network definition api. Dosegljivo: `https://github.com/wang-xinyu/tensorrtx`. [Dostopano: 17. 3. 2023].

[8] F. Lanzani. Install nvidia deepstream 6.0.1 with python bindings on jetson. Dosegljivo: `https://medium.com/@lanzani/nvidia-deepstream-with-python-bindings-on-jetson-f9ffdcb16b06`. [Dostopano: 17. 3. 2023].

[9] J. Talloen. Nvidia deepstream 6.1 python boilerplate. Dosegljivo: `https://github.com/ml6team/deepstream-python`. [Dostopano: 17. 3. 2023].

[10] J. Špeh. Nvidia deepstream 6.0 with python bindings on jetson nano. Dosegljivo: `https://github.com/spehj/jetson-nano-yolov7-deepstream`. [Dostopano: 17. 3. 2023].

[11] M. Luciano. Deepstream-yolo. Dosegljivo: `https://github.com/marcoslucianops/DeepStream-Yolo`. [Dostopano: 17. 3. 2023].