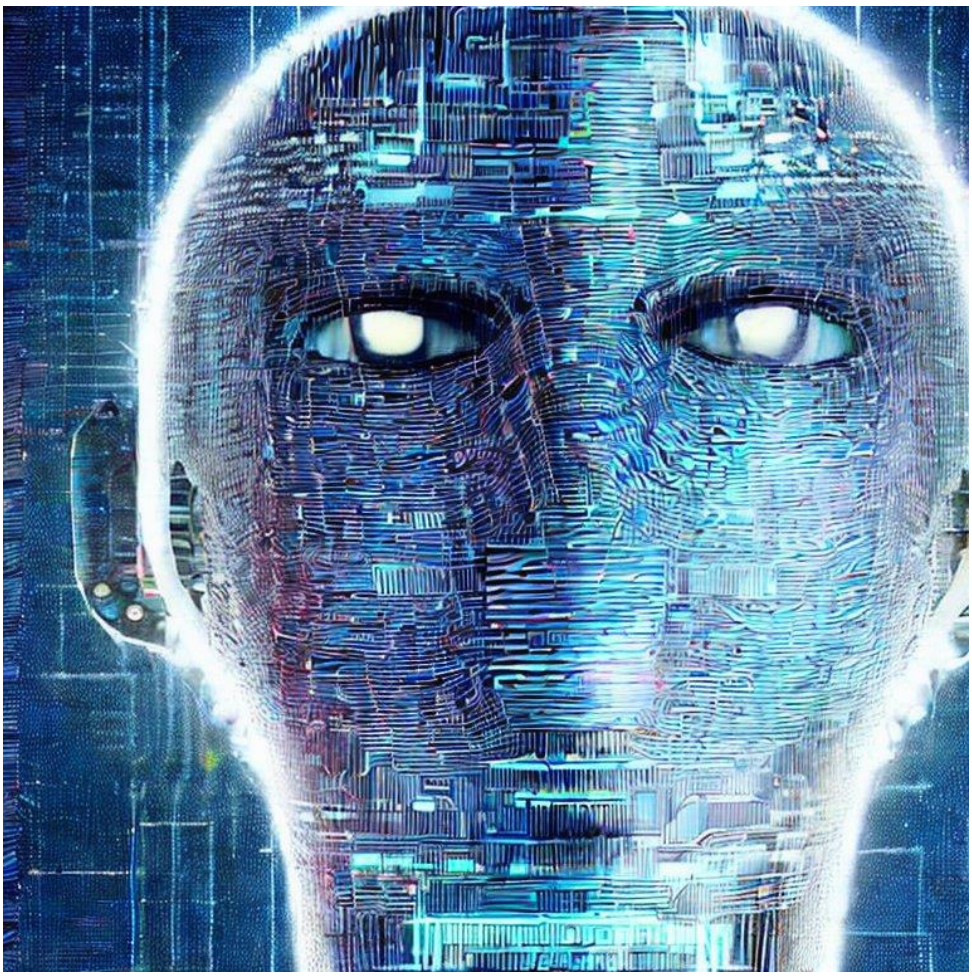

COMPUTER VISION ON EDGE DEVICES

ABSTRACT
Einsatz von computer vision Modellen und machine learning frameworks auf Nvidia Jetson Plattform.

Toni Badertscher
CAS ML4SE Projektarbeit

Inhalt

Thema der Arbeit

Die Projektarbeit befasst sich mit computer vision Modellen und Anwendungen auf embedded Plattformen. Das Thema hat mich interessiert, weil wir uns im zweiten Teil der Ausbildung mit convolution auseinandergesetzt haben und weil ich ursprünglich eine Elektronik Ausbildung abgeschlossen hatte und trotz langjähriger Informatik Laufbahn immer noch grosses Interesse an Hardware und embedded Systemen habe. Mit meinem Sohn zusammen habe ich einige Projekte mit Arduino gebaut und bin durch das auf die Nvidia Jetson Baureihe gestossen, welches ein ähnliches Oekosystem und community wie Arduino hat, aber mit Schwerpunkt machine learning und Robotik. Auch der Aspekt einer end-to-end Anwendung hat eine grosse Rolle gespielt, da wir uns im Unterricht vor allem mit Theorie, Modellarchitektur und Training auseinandergesetzt haben und ich auch die Aspekte deployment, performance/latency und real-time Verarbeitung verstehen wollte.

Allgemeine Angaben

Auftraggeber:	OST - Ostschweizer Fachhochschule Rapperswil
Studiengang:	CAS Machine Learning for Software Engineers
Autor:	Toni Badertscher
Datum:	Q4 2022
Betreuer:	Martin Stypinski

Projektidee

Das Themengebiet computer vision ist recht umfangreich mit vielen Anwendungsbereichen:

- semantic segmentation
- object detection
- pose estimation
- action detection
- object tracking
- gesture recognition
- augmented reality
- autonomous vehicles

Ich habe mich für eine Aufgabe im Bereich pose estimation & classification entschieden. Eine Praktische Anwendung dafür wäre zum Beispiel eine Gestensteuerung für Roboter oder eine Fitnessapplikation (z.B. Erkennung und Analyse von Yoga-Positionen).

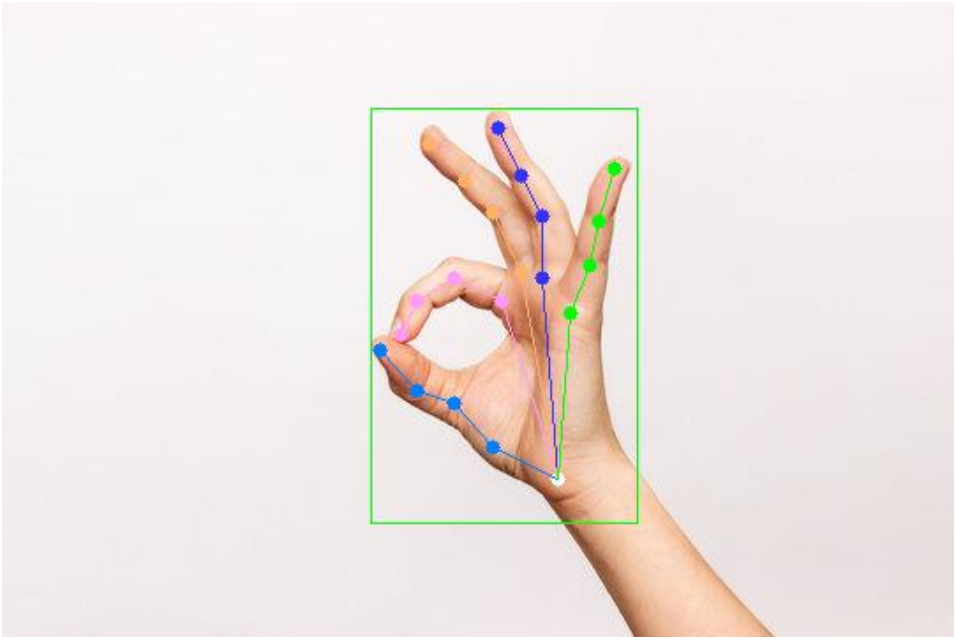

Abbildung 1: mmdetection/mmpose hand pose estimation

Die Idee zur Yoga pose estimation/classification kam durch meine Partnerin, da sie ein Yoga-Studio betreibt.

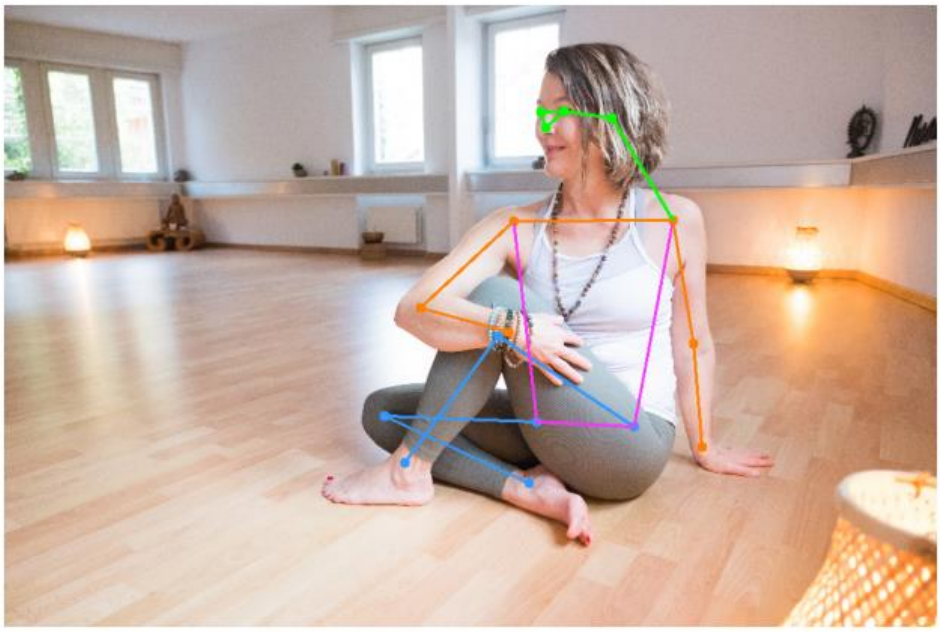

Abbildung 2: yolov7 human pose estimation

Interessant wäre auch etwas im Bereich «autonomous vehicles» zu machen, dies würde aber wahrscheinlich vom Aufwand nicht in den zeitlichen Rahmen passen. Einige Schulen und Universitäten bieten online Kurse in diesem Bereich an, z.B. ETH Zürich:

https://www.edx.org/course/self-driving-cars-with-duckietown

Die Basis/Plattform dazu ist ebenfalls Nvidia Jetson. Ich werde mich nach dem CAS und einer Verschnaufpause in diesem Bereich weiterbilden, unter anderem auch aus persönlichem Interesse an Robotik und ROS.

Zeitlicher Ablauf, Planung

August	Projektidee, Beschaffung der Hardware
September	Analyse von bestehenden Lösungen, Modellen und research papers im Bereich body und hand pose estimation
Oktober	Defintion und Umsetzung eines MVP
November	Erweiterungen, state-of-the-art Lösungen, z.B. 3d pose estimation, transformer Modelle
Dezember	Abschluss der Arbeiten, fine tuning, Prästentation, Dokumentation

Ich habe rasch festgestellt, dass der zeitliche Rahmen recht eng werden würde und viele der anfangs plausiblen Ideen und Anwendungen doch einiges an Problemen mit sich brachten. Der Aufbau und die Konfiguration der Plattform (Jetson) und Anfangsschwierigkeiten haben recht viel Zeit in Anspruch genommen. Ich habe mich deshalb entschieden, bis Ende der Herbstferien eine lauffähige Anwendung zu bauen, zu einem grossen Teil basiert auf bestehenden Modellen und Lösungen, um dann die restliche Zeit für Verbesserungen zu nutzen und andere Ansätze zu evaluieren. Ein Modell und die ganzen Komponenten komplett selbst zu bauen hätte deutlich mehr Zeit in Anspruch genommen.

Hardware

Es gibt verschiedene Kategorien von machine learning accelerator hardware, z.B. Coral USB stick oder Googles Tensor Chip, welcher vor allem bei Mobiltelefonen zum Einsatz kommt. Coral war nicht verfügbar (EOL) und vom Anwendungsgebiet eingeschränkt. Ich wollte eine offene Plattform mit einem grossen Oekosystem von verfügbaren frameworks und Anwendungen. Somit kam die Nvidia Jetson Produktefamilie auf den Radar, unter anderem auch weil Nvidia den machine learning Markt nahezu vollständig dominiert. Andere Grafikkartenhersteller (AMD, Intel) sich praktisch nur im Gaming Segment vertreten.

Nvidia bietet mit der Jetson Familie Entwicklungsboards für die Industrie, Education und Hobbyanwender an. Die Plattform wird weltweit von über 80 Universitäten eingesetzt und es gibt eine Vielzahl von Anwendungen und Beispielen im Netz.

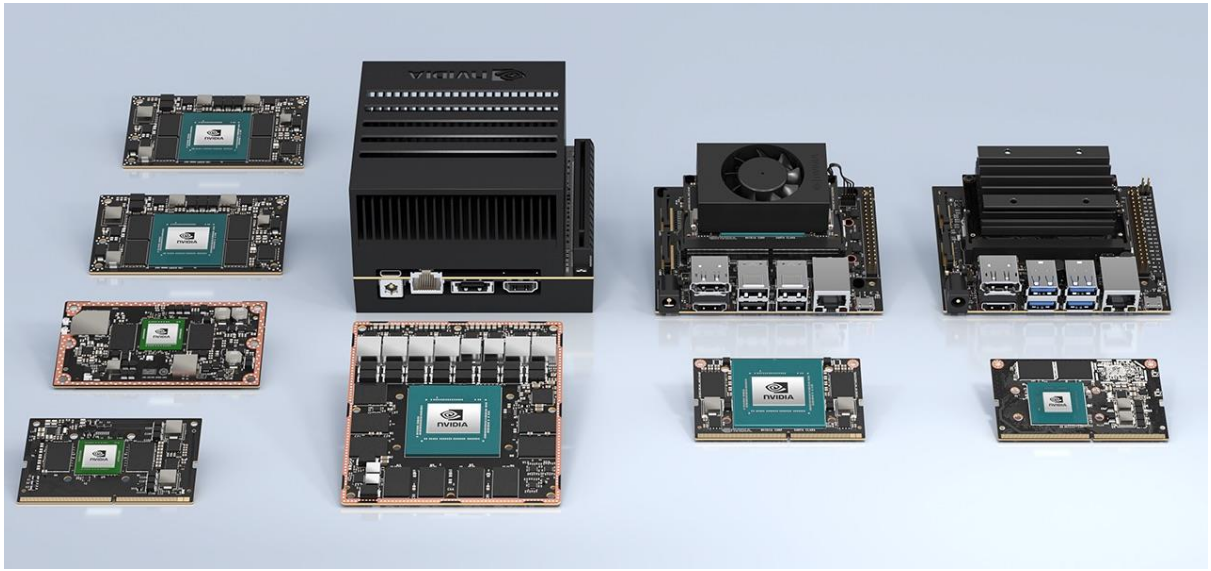

Abbildung 3: jetson product family

Ein Problem bei der Beschaffung war die Verfügbarkeit vom Einstiegs-modell (Jetson Nano). Die bestehende Hardwaregeneration wird nicht mehr produziert, es gibt weltweit keine Lagerbestände mehr und die neue Generation ist erst Anfang 2023 lieferbar. Als Alternative dazu gibt es die NX Baureihe, welche ebenfalls schwierig zu beschaffen und etwas teurer ist. Ich bin aber bei einem OEM in China (seeed studio) fündig geworden und die Ware wurde relativ prompt Anfang August geliefert.

Entwicklungsumgebung

Das Jetson board kann je nach Modell auf verschiedene Arten aufgesetzt werden. Man kann ein komplettes Image auf eine Memory Karte laden und das System damit starten oder via SDK manager und USB-Kabel den kit direkt bootstrappen.

Das vom OEM gelieferte Modell kam mit Jetpack 4.6 vorinstalliert und war innert kurzer Zeit bereit. Die von Nvidia zur Verfügung gestellten Anwendungen waren aber nicht lauffähig Aufgrund eines bekannten bug in der TensorRT Bibliothek. Abhilfe schaffte das kurz darauf (Mitte August) verfügbare Jetpack 5.0.2 release. Da die eingebaute memory card mit 16GB recht knapp bemessen ist und auch von der performance nicht optimal war, habe ich eine SSD verbaut und den kit komplett frisch aufgesetzt. Der Basis-kit beinhaltet unter anderem:

- Ubuntu 20.04 LTS
- Cuda 8.4
- TensorRT
- cuDNN (CUDA Deep Neural Network)
- DeepStream

Das accelerator board wird durch Nvidia hergestellt, der OEM macht das Peripherie-board und Gehäuse. Der kit kann headless oder mit lokalem Display betrieben werden. Ich habe mich der Einfachheit halber für zweiteres entschieden. Es sind HDMI und USB-Anschlüsse vorhanden. Kameras können über MIPI oder USB-Schnittstelle angeschlossen werden.

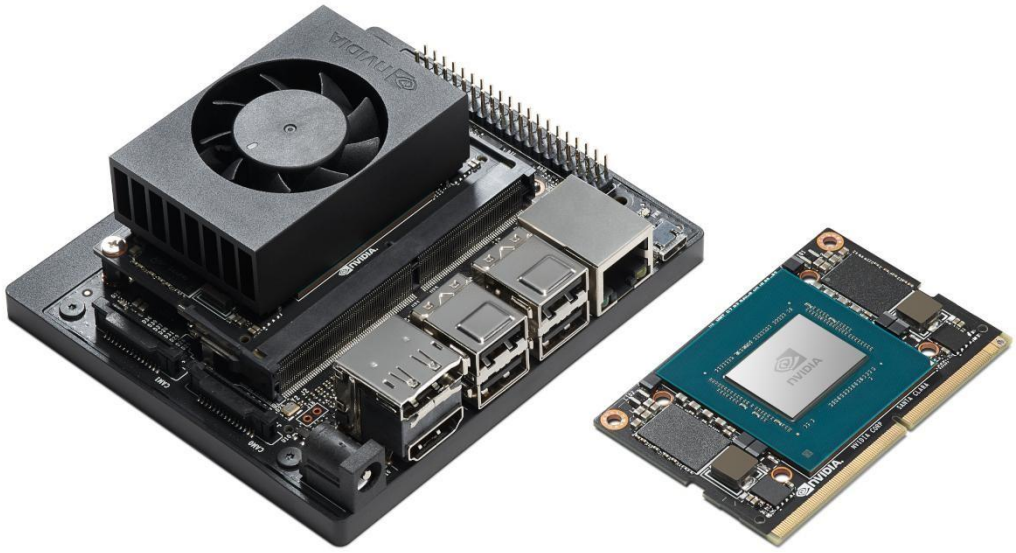

Abbildung 4: Nvidia jetson xavier NX

Ich hatte anfangs die Absicht den kit als Entwicklungsumgebung zu nutzen, da in meinem Fall die Ausstattung recht komfortabel war (16 GB RAM, 250GB SSD, 6-core CPU, 384-core GPU). Die Architektur der CPU ist basiert auf ARM, die Taktrate und Leistungsfähigkeit sind recht eingeschränkt. Die Hardware ist auf niedrigen Stromverbrauch (10 – 20 Watt) und Inferenz ausgelegt. Ein weiterer Faktor war die Kompatibilität der verfügbaren Libraries mit ARM Architektur, welches etliches an Problemen bescherte. Praktisch alle Jetson Anwendungen waren auf Jetpack 4.x ausgelegt, welches auf Ubuntu 18 und Cuda 10 basiert. Das war vor allen problematisch für PyTorch und TensorRT. Viele der Probleme haben sich mit neuen Software-releases lösen lassen. Es hat aber eine Weile gedauert bis diese verfügbar wurden und mich hat es einige Zeit und Nerven gekostet.

Aus diesem Grund habe ich mir einen neuen Laptop mit Nvidia Grafikkarte gekauft. Als Entwicklungsumgebung empfiehlt sich Linux, Windows wird nur schlecht unterstützt (problematisch sind vor allem cuDNN und TensorRT). Alle Tücken die ich mit ARM Architektur hatte waren auf X86 kein Problem. Für ARM sind einige Frameworks nur durch Nvidia vorkompiliert verfügbar (z.B. PyTorch, Torchvision, openCV). Die Versionen waren teilweise veraltet oder nicht kompatibel. PyTorch oder openCV komplett von source zu kompilieren ist aufwendig und dauert lange.

Analyse

Um mich mit pose estimation vertraut zu machen habe ich einiges an Zeit investiert, um bestehende Lösungen anzuschauen. Eine komplette Lösung als Einzelarbeit zu bauen wäre zu anspruchsvoll und vom zeitlichen Budget nicht realisierbar gewesen.

2D Hand Pose Estimation-RGB (Olga Chernytska)

Durch einen Artikel auf Medium (https://medium.com/) bin ich auf die Master Thesis von Olga Chernytska gestossen. Sie beschreibt darin wie pose estimation mit RGB Bildern funktioniert. Die Basis bildet das FreiHAND dataset der Uni Freiburg. Es gibt dazu ein github Projekt mit Notebooks für Training und Inferenz. Der Datensatz besteht aus 32560 RBG Bildern (224x224 pixel) und 3D keypoint annotations (21 Hand keypoints). Die annotations werden auf 2D umgerechnet, weil das Modell nur 2D unterstützt. Ich habe diese Arbeit primär zum Verständnis der Daten und der Lösungsansätze angeschaut. Es gab auch einen Anhaltspunkt für den Rechenleistungsbedarf für das Training. Die Grösse des Datensatzes ist 3.7 GB und es dauert ca. 90 Minuten für 200 Epochen auf GPU. Andere Datensätze sind teilweise massiv grösser und würden sehr viel mehr Zeit in Anspruch nehmen.

Nvidia trt_pose_hand

Nvidia AI IOT (https://github.com/NVIDIA-AI-IOT) bietet eine Vielzahl von Beispielen für embedded Systeme. Unter anderem JetBot, welcher im education Bereich beliebt ist. Ich habe dort diverse Sachen angeschaut und auf Jetson ausprobiert, teilweise jedoch mit etlichen Problemen. Weil viele der Anwendungen auf älteren Versionen von Jetpack und frameworks basieren. trt_pose_hand ist eine 2D hand pose estimation Lösung mit classifier für hand pose detection. Das Modell basiert auf Resnet18 und ist von Nvidia mit internen Daten vortrainiert worden. Die Implementierung basiert auf PyTorch für pose estimation und scikit-learn für den Classifier. Das PyTorch Modell wird mit TensortRT optimiert (via torch2trt).

Google MediaPipe

MediaPipe ist eine cross-platform Lösung für streaming Medien für eine Vielzahl von Anwendungen. Als Plattformen sind Android, IOS, JavaScript, Python und C++ unterstützt. Die verfügbaren Lösungen (z.B. face detection, pose estimation, segmentation) basieren auf vortrainierten TensorFlow oder TensorFlow Lite Modellen.

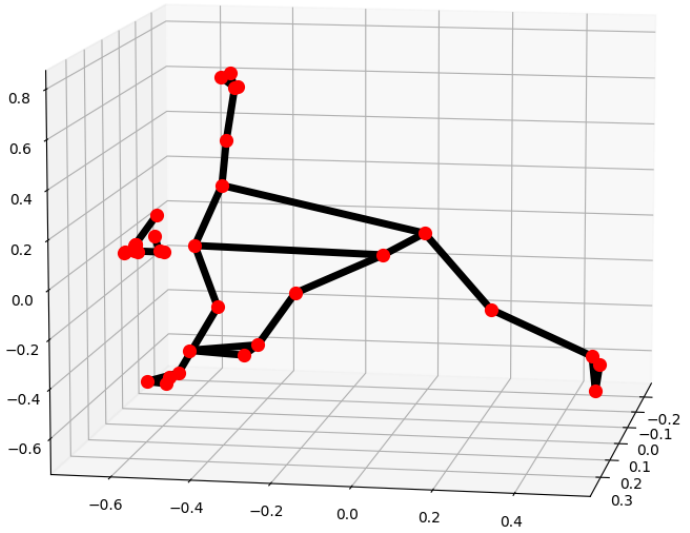

Abbildung 5: MediaPipe 3d body pose landmarks

MediaPipe Hands erzeugt 3D keypoints aus RGB Daten. Die Architektur besteht aus einer pipeline mit detection model welches eine bounding-box erzeugt und das reduzierte Bild an einen Pose Estimator weitergibt. Das framework scheint ausgereift und erreicht ordentliche perfomance. Limitationen sind TensorFlow als Basis und mangelhafte Unterstützung von Cuda (nur Version 10.x unterstützt). Es ist zudem unklar, ob Coda direkt unterstützt wird oder ob man dazu das framework vom Quellcode neu kompilieren muss (wahrscheinlich zweiteres).

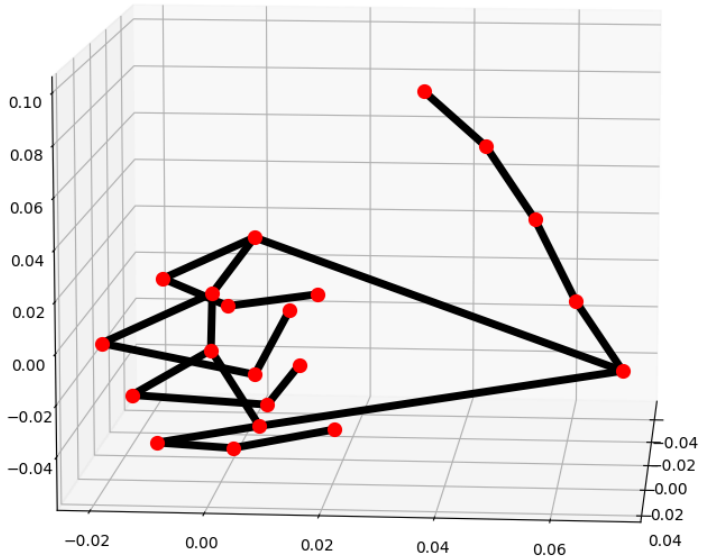

Abbildung 6: MediaPipe 3d hand pose landmarks

Ich habe Anfangs Probleme mit MediaPipe auf Jetson gehabt und Aufgrund von Library Problemen nicht weiter Zeit investiert (bin aber später darauf zurückgekommen und die Probleme gelöst).

OpenMMLab, MMPose

OpenMMLab ist ein open-source framework für research und industrielle Anwendungen. Es unterstützt Objekte detection (mmdetection), pose estimation (mmpose) und eine Vielzahl weitere Anwendung-fälle. Ebenfalls beindruckend ist die grosse Anzahl verfügbarer (vortrainierter) Modelle.

Im Bereich pose detection bietet es eine Anzahl von 2D und 3D Modellen für body-, hand- und animal-pose detection.

Die Basis bietet MMCV mit pipeline und PyTorch integration und Cuda support. Alle Estimator Modelle arbeiten nach einem top-down Ansatz, wo ein Detector und Estimator in einer Pipeline kombiniert werden.

Da sich die research community und contributors primär in China befinden sind die Tutorials auf Chinesisch und die Dokumentation teilweise lückenhaft. Auch ist die Modellkonfiguration recht kompliziert und undurchsichtig. Die Modelle und realisierbaren Lösungen sind aber recht ordentlich und vielseitig. Unklar ist wie zu Laufzeit TensorRT unterstützt wird. Ein Nachteil ist auch dass der pipeline Ansatz mit 2 Modellen sehr rechenintensiv sein kann.

YOLOv7

Yolo bietet verschiedene Generationen von object detection und segmentation Modellen an, in der neuesten Iteration (v7) ebenfalls pose estimation (allerdings nur body pose). Yolo ist in vielen Bereichen im Bezug auf performance und accuracy state-of-the-art. Im Gegensatz zu MediaPipe welches auf single-person optimiert ist funktioniert yolo auch gut mit mehren Personen im Bild (zum Beispiel eine Gruppe von Leuten in einer Yoga-Klasse). Der Rechenleistungsbedarf ist aber wesentlich grösser als bei MediaPipe und eine leistungsfähige GPU ist Voraussetzung. Yolo unterstützt Export nach ONNX Format und von dort Konvertierung nach TensorRT. Man kann dadurch die Modelle in Nvidia TAO importieren, ein Teil von v3 und v4 werden von Nvidia direkt unterstützt.

Nvidia TAO, DeepStream, Triton

Nvidia bietet mit TAO (transfer learning), DeepStream (streaming und pipelines) und Triton (Inferenz server und Modell Repository) eine Anzahl von leistungsfähigen frameworks an. Ebenfalls sind von Nvidia eine grosse Anzahl vortrainierter Modelle via NGC Portal verfügbar.

Ich habe mit TAO ein action detection Modell trainiert welches aus einem Video-stream die Art der Aktivität detektieren kann (z.B. Yoga oder dead-lifting). Die erzeugten Modelle sind mit TensorRT performance optimiert und können mit DeepStream integriert oder in Triton deployed werden. Es gibt 2D und 3D body pose estimation Modelle, aber keine hand pose Unterstützung. Die trt_pose und trt_pose_hand Modelle sind PyTorch basiert und mit TAO nicht kompatibel (Backend basiert auf TensorFlow). Es gibt die Möglichkeit Modelle auf der Basis von ONNX zu importieren.

Ich habe mit allen 3 frameworks einige Startschwierigkeiten gehabt. Speziell DeepStream hat eine learning curve und es gab einige Probleme mit device drivers. Da die meistens Computer neben der Nvidia GPU noch eine interne Grafikkarte haben kann es sein, dass gstreamer (das open-source framework auf dem DeepStream basiert) den falschen Treiber ansteuert. Abhilfe hat eine Konfiguration gebracht welche in Linux die Nvidia Grafikkarte als primary setzt).

Insgesamt hat alles auch sehr viel Zeit gefressen und es braucht etliches an Erfahrung um diese frameworks in der Praxis einzusetzen. Als Komplettlösung aber sicher ein interessanter Ansatz mit guter Performance Optimierung und einer breiten Unterstützung von Plattformen (alle gängigen cloud provider). Ich werde später auf DeepStream zurückkommen und probieren, einen Teil der erwähnten Modelle zu integrieren (z.B. yolo).

MVP

Aufgrund von Zeitdruck und einem bereits beträchtlichen Aufwand beim Aufsetzen der Hardware und Voranalyse entschied ich mich (nach Rücksprache mit Martin) den MVP basiert auf hand pose estimation zu machen. Als Basis dazu diente das Nvidia trt_pose_hand Projekt. Folgende Gründe waren ausschlaggebend:

- Alle pose estimator Modelle (mit Ausnahme von Yolov7) hatten teils erhebliche Probleme mit Yoga Posen (inklusive state-of-the-art Modelle von Nvidia)
- Die body pose Modelle haben alle Unterschiedliche landmark Definitionen als output. Es ist insofern schwierig passende Trainingsdaten für den Classifier zu finden
- Das Testen von hand pose Gesten und das Erzeugen von Testdaten ist wesentlich einfacher als zum Beispiel Yoga Posen beim body pose estimator

Anforderungen

- 2D oder 3D keypoint estimation auf der Basis von RGB input (webcam)
- Klassifizierung von Handzeichen
- single shot bottom-up oder top-down (detection model & pose estimation)
- single person (ein oder beidhändig)
- als Basis sollen PyTorch Modelle dienen, Erweiterbarkeit
- real-time, mindestens 5fps
- target platform: X86 und jetson (ARM)

Anpassungen

Der grösste Zeitbedarf am Nvidia source code war anfangs die Abhängigkeit von TensorRT und die Inkompatibilitäten mit python und C++ Bibliotheken. Die erste Version vom MVP lief auf Jetson nur innerhalb eines docker containers. Dies Aufgrund der fehlenden PyTorch Cuda und TensorRT Unterstützung auf Jetpack 5.0.2. Die Lösung war auch ausgelegt auf headless Betrieb von Jetson und die Laufzeit Umgebung wurde via jupyter server im docker image gestartet. Die Bildausgabe erfolgte in der Auflösung des Modells (224x224 pixel) in einem ipywidget. Das Ganze hat also insgesamt nicht besonders toll ausgeschaut, hat aber ansonsten ordentlich funktioniert.

Nachdem neuere Versionen von PyTorch und Cuda von Nvidia verfügbaren wurden liessen sich die Probleme auf Jetson lösen und ich habe danach folgende Anpassungen vorgenommen:

- Skalierung der Bildausgabe damit die originale Auflösung der webcam ausgegeben wird (z.B. 640x480)
- Ersetzen von JetCam durch openCV für die stream Verarbeitung (openCV hat im docker container von Nvidia nicht funktioniert)
- Diverse kosmetische Anpassungen am Code
- Eigenes Datenset zum Trainieren des Classifiers
- Benchmark zur Evaluation der performance

Es gibt keine Angaben zu dem Daten die Nvidia für das Training des Modells verwendet hat. Von der Architektur ist es ein single-shot detector (SDD) der nach dem bottom-up Prinzip funktioniert. Dieser Ansatz hat aber in der Praxis deutlich öfter Probleme mir der Erkennung der Hand. Das Resnet18 Modell welches als Basis genutzt wurde ist optimiert auf performance, es gibt insofern Abstriche bei der accuracy.

Installation

Folgende Anforderungen bestehen an die Plattform:

- Nvidia Cuda 11.x
- Nvidia TensortRT
- Nvidia Deep Neural Network library (cuDNN)

Die Python Abhängigkeiten können via environment file im Repository in ein virtual env geladen werden. Dies sind die hauptsächlich verwendeten frameworks:

- pytorch, torchvision
- tensorrt, torch2trt
- scikit-learn
- opencv
- trt_pose

Repository Struktur

Die Demo-applikation des MVP befindet sich ich mvp Verzeichnis und kann direkt gestartet werden:

conda activate nvidia

cd mvp

python hand_pose_classification.py

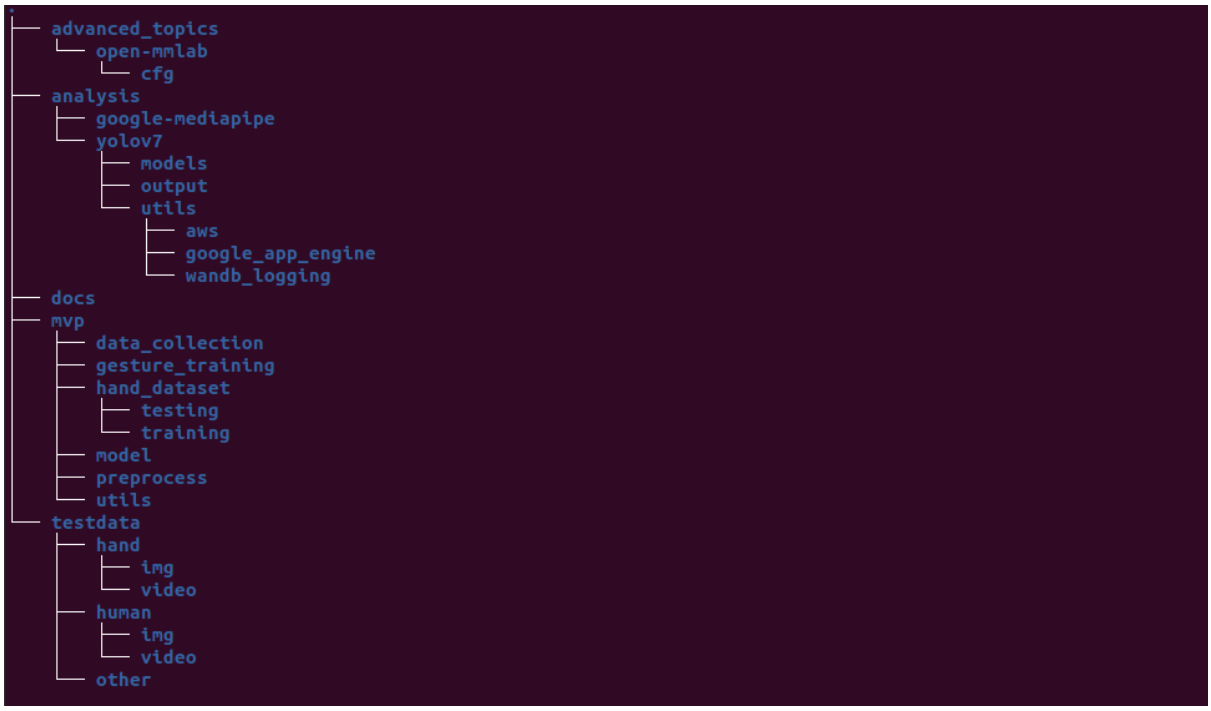

Abbildung 7: git repository Struktur

Classifier

Der Classifier wurde mit scikit-learn SVM (support vector machines) implementiert. Hierzu gab es keine Daten oder Information über die Funktionsweise oder accuracy.

Dataset

Da von Nvidia keine Daten für die Applikation zur Verfügung gestellt wurden, habe ich selbst ein kleines Datenset erzeugt. Es gibt 2 Notebooks, eines zum Erstellen der Daten und eines um den classifier zu trainieren. Pro Klasse gibt es 40 Bilder zum Trainieren und 10 zum Testen. Jeweils 50% linke und rechte Hand.

Klassen

class	*name*
1	thumbs up
2	thumbs down
3	victory
4	horns
5	pinky
6	vulkan salute
7	finger gun

Abbildung 8: Klassen für die Gestenklassifizierung

Confusion matrix

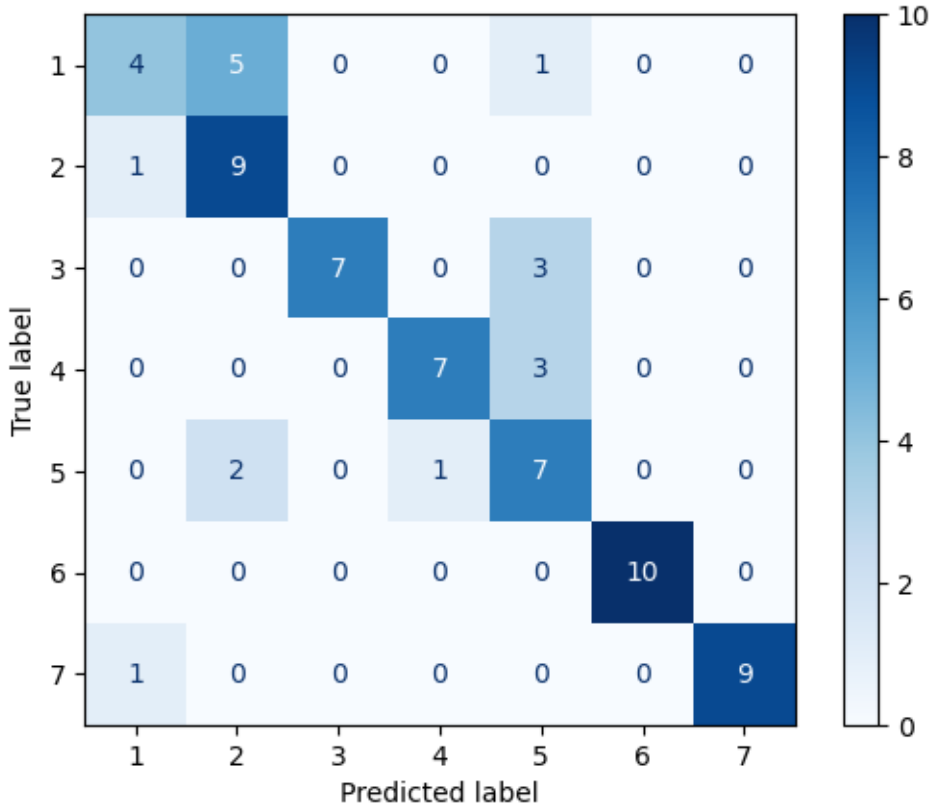

Abbildung 9: sklearn confusion matrix

Classification report

class	*precision*	*recall*	*f1-score*	*support*
1	0.67	0.40	0.50	10
2	0.56	0.90	0.69	10
3	1.00	0.70	0.82	10
4	0.88	0.70	0.78	10
5	0.50	0.70	0.58	10

6	1.00	1.00	1.00	10
7	1.00	0.90	0.95	10
accuracy			0.76	70
macro avg	0.80	0.76	0.76	70
weighted avg	0.80	0.76	0.76	70

Abbildung 10: sklearn classification report

Performance

Normalerweise läuft PyTorch im sogenannten «eager» Modus, welcher für das Training von Modellen optimiert ist. Für das deployment können die Modelle in das torchscript Format exportiert werden. Zu Laufzeit wir dann ein just-in-time compiler (JIT) verwendet. Ich konnte in der Praxis keine grossen Unterschiede feststellen. Ein weiterer Ansatz für Performance-optimierung ist das Konvertieren der Modelle in das Nvidia TensorRT Format. TensorRT ist ein ahead-of-time (AOT) compiler welches den Objektgraphen des Modells vereinfacht und spezielle kernels für die optimale Nutzung der GPU bietet. Es gibt verschiedene Wege TensorRT zu nutzen:

- Direkt aus python via torch2trt framework (Nvidia)
- Neue Versionen von PyTorch bieten eine direkte Integration von TensorRT via torchscript
- Via ONNX export und Konvertierung mit Nvidia trtexec

Um Anhaltspunkte für das Laufzeitverhalten zu bekommen habe ich die Modelle einem benchmark-test unterzogen (Quellcode dazu gibt es von Nvidia).

	batch size 1	**batch size 8**
AMD Ryzen RTX3060 **CPU**	118.93 ms	714.60 ms
AMD Ryzen RTX3060 **GPU**	5.43 ms	30.56 ms
AMD Ryzen RTX3060 **TRT**	1.07 ms	16.46 ms
Jetson Xavier NX **CPU**	7534.95 ms	28720.30 ms
Jetson Xavier NX **GPU**	36.37 ms	343.61 ms
Jetson Xavier NX **TRT**	5.96 ms	24.80 ms

Abbildung 11: performance benchmark

Die Basis für den benchmark bietet das Resnet18 Modell aus dem MVP. AMD Ryzen/RTX 3060 ist die Entwicklungsumgebung und Jetson die angestrebte Laufzeitumgebung für den MVP. Es gibt wie erwartet einen grossen Sprung zwischen CPU & GPU performance, welcher auf Jetson nochmals deutlicher ausfällt, da die CPU erfahrungsgemäss nicht seht leistungsfähig ist. Die GPU performance auf Jetson ist erstaunlich gut, ebenfalls die erreichte Steigerung mit TensorRT um Faktor 6 bei batch size 1 (was dem Praxiseinsatz entsprechen würde). Das Modell wurde mit FP16 (half-precision) Datentyp aufgesetzt.

Ich bin mir nicht sicher, wie es zu diesem grossen Sprung kam (tatsächlich wirbt Nvidia mit Faktor 2-6 Verbesserung), bei Tests mit anderen Modellen habe ich in der Regel knapp eine Verdoppelung der performance erziehlt. In der Praxis bestätigen sich aber die guten Resultate aus dem benchmark, der MVP läuft flüssig auf Jetson, ohne grosse CPU und GPU Auslastung.

Ertweiterungen, SOT

Der MVP hat einige Schwachstellen im Bezug auf Zuverlässigkeit beim Erkennen der Hand und teilweise qualitative Mängel bei bestimmen Gesten oder Fingerpositionen/Winkel. Zudem funktioniert es nur für eine Hand (links oder rechts, nicht beide gleichzeitig) und die Landmarks sind nur 2D (was für den angestrebten use-case durchaus ok war).

OpenMMLab

Mit MMPose und MMDetection bietet sich die Möglichkeit, die Probleme des MVP mit einem 2 stufigen Ansatz zu Lösen (top-down). In einer pipeline werden 2 Modelle kombiniert, ein detector mit bounding-box zur Erkennung der Hände sowie ein estimator für pose esimation. Dieser Ansatz erzielt deutlich bessere Resultate als der MVP, es funktioniert für beide Hände simultan. Es gibt eine Vielzahl dectector und estimator Modelle die konfiguriert werden können. Ich habe dazu ein kleines und performance orientiertes Modell (mobilenetV2) für den detector konfiguriert sowie Resnet50 für den estimator.

Ein Nachteil von diesem Ansatz ist der deutlich höhere Leistungsbedarf. Im normalbetrieb auf GPU läuft das ganze schon mit deutlich tieferer frame Rate, ebenfalls wird massiv mehr CPU-Rechenleistung benötigt. Auf Jetson läuft es zwar, aber mit unbefriedigender performance. OpenMMLab bietet eine weitere Komponente (MMDeploy) an, welche die Modelle auf ONNX respektive TensorRT konvertieren kann. Die Installation all dieser frameworks und die Konfiguration der Modelle ist teils recht kompliziert und zeitaufwendig. Es ist mir nach einiger Zeit gelungen die Modelle nach TensorRT zu exportieren. Unklar ist aber noch, wie man diese Modell zu Laufzeit im TensorRT Format nutzen kann. Dokumentation und online Beispiele sind spärlich respektive nicht vorhanden.

Eine weitere Möglichkeit wäre OpenMMLab nur für das Trainieren, Konfigurieren und Exportieren der Modelle zu nutzen und als Laufzeit Umgebung Nvidia DeepStream zu nutzen.

Facebook (Meta research) InterHand2.6M (3D)

Meta research hat einen Datensatz für 3D hand pose estimation aus RBG Daten erzeugt und dazu ein Projekt auf github zur Verfügung gestellt. Zusätzlich gibt es eine Integration mit einem mesh-rendering Modell auf der Basis von Mano.

Der Ansatz basiert auch auf top-down, das Modell braucht die bounding-box Koordinaten als input. Ich habe für die Laufzeitumgebung ebenfalls MMPose benutzt, es gab dort aber noch Problem mit der korrekten Interpretation der bounding-box.

Conclusion, lessons learned

Computer Vision entwickelt sich immer noch sehr schnell weiter. Es gibt eine Vielzahl research papers und Modelle. Vieles davon kommt aus dem akademischen Umfeld, es braucht oft viel Aufwand die vorgestellten Modelle und Methoden in der Praxis umzusetzen. Was länger als 2 Jahre zurück liegt, lohnt sich meistens nicht viel Zeit damit zu verwenden, weil die verwendeten frameworks schon wieder veraltet sind.

Bei state-of-the-art Ansätzen sind oft die Anforderungen and die Hardware recht hoch und auch der Umfang der Trainingsdaten recht massiv. Es ist als Einsteiger anfangs recht schwierig zu erkennen, welche Ansätze mit vernünftigem Aufwand zum Ziel führen. Viele der auf den online Plattformen angebotenen Ansätze liefern dort im benchmark Vergleich sehr gute accuracy, sind aber in der Praxis nicht immer einfach zu nutzen. Im Fall von pose-estimation gibt es noch keinen klaren Trend

Richtung Transformer Modelle und ob diese dort einen Vorteil bringen. Ebenfalls sind bei Transformer der hohe Trainingsaufwand und die Hardwareanforderungen problematisch.

Auch die Hardware-entwicklung schreitet immer noch schnell voran. Es gibt ca. alle 2 bis 3 Jahre eine neue GPU Generation und die Leistungssteigerungen sind immer noch recht beachtlich (im Vergleich zur Entwicklung der CPU).

Von den evaluierten frameworks und Modellen ist Google MediaPipe das ausgereifteste Produkt. Es lässt sich mit relativ wenig Aufwand aufsetzen, braucht wenig hardware-ressourcen und ist auf einer breiten Palette von Plattformen unterstützt. Alle PyTorch basierten Ansätze bedingten einen deutlich höheren Aufwand.

Pose estimation ist eher eine Nische. Im Bereich object detection und segmentation gibt es deutlich mehr Modelle und Support von Anbietern wie Nvidia. Grund ist vermutlich, dass in diesem Bereich mehr kommerzielles Interesse besteht und mehr investiert wird (z.B. Nvidia Drive, welches von Volvo, Mercedes Benz, LandRover, Nio etc. eingesetzt wird).

Eine positive Erfahrung ist die Tatsache, dass sehr vieles open-source verfügbar ist und man freien Zugang zu Tools und Modellen hat, auch von Anbietern wie Nvidia, Google, Meta.

Positive Erfahrungen

- Freier Zugang zu research papers, Modelle, code
- Viele verschiedene und recht umfangreiche frameworks
- Frei verfügbare Datensets
- Developer community, Support-foren, blogs, Youtube etc.
- Integration von cloud provider (Google vertex, Azure, Amazon Sagemaker)

Negative Erfahrungen

- Die durchschnittliche Halbwertszeit beträgt 2 Jahre, danach sinkt die Nutzbarkeit beträchtlich
- Viele frameworks haben recht umfangreiche Abhängigkeiten von python libraries
- Trotz virtuellen Umgebungen ist der Umgang mit python libraries und Versionen mühsam
- Viele framework haben ein learning curve und die Versionen sind recht schnell veraltet
- Zeit und Kostenaufwand (Anforderungen an CPU & GPU)

References

Olga Chernytska, 2D Hand Pose Estimation from RGB Image
https://github.com/OlgaChernytska/2D-Hand-Pose-Estimation-RGB
NVIDIA AI IOT
https://github.com/NVIDIA-AI-IOT
OpenMMLab
https://github.com/open-mmlab
Google MediaPipe
https://google.github.io/mediapipe
Facebook (Meta research) InterHand2.6M
https://github.com/facebookresearch/InterHand2.6M