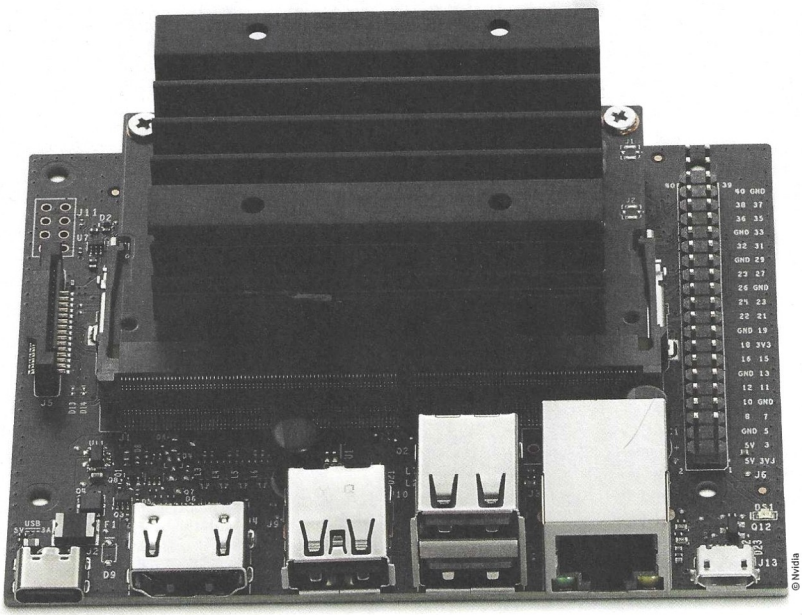


Jetson Nano: KI-Einstieg zum moderaten Preis

Flotter Zwerg

Kaum teurer als ein Raspberry Pi 4, bietet Nvidias Jetson Nano 2GB einen idealen Einstieg in die Welt des Machine Learning. Konstantin Agouros

Der Autor

Konstantin Agouros arbeitet als Head of Open Source & AWS Projects bei der Matrix Technology AG und berät dort mit seinem Team Kunden zu Open-Source-, Sicherheits- und Cloud-Themen. Sein Buch „Software Defined Networking: Praxis mit Controllern und OpenFlow" ist bei de Gruyter erschienen.

Nvidia beschränkt sich schon lange nicht mehr nur auf Grafikkarten. Spätestens seit dem Kauf von ARM schließt das Unternehmen langsam zu Intel und AMD auf, längst ist der Einsatz von GPUs im Bereich des Machine Learning ein wichtiger Aspekt in den Verkaufszahlen. Nvidia bietet aber auch Entwicklerkits an, die im Formfaktor einem Raspberry Pi ähneln, aber auf einer GPU aufbauen.

Den kleinsten Vertreter, den Jetson Nano, gibt es jetzt in einer Variante mit 2 GByte RAM für erschwingliche 75 bis 80 Euro (ohne Netzteil und SD-Karte). Der Prozessor, ein Quadcore-ARM-A57 („Tegra") taktet mit 1,43 GHz und schnitt in einem einfachen Leistungstest (`openssl speed`) etwas schneller ab als ein Raspberry Pi 4. Der Onboard-GBE-Adapter des Jetson Nano hängt am PCI-Bus, nicht wie bei anderen Kleinstrechnern an der USB-Schnittstelle. Außerdem verfügt der Winzling über zwei USB-2-Typ-A-Buchsen, einen USB-2-Micro-B-Port, einen USB-3-Anschluss sowie einen Slot für eine MicroSD-Karte. Daneben besitzt er einen HDMI-Ausgang, einen MIPI-CSI-2-Port für Kameras (an den auch die Pi Cam passt) sowie eine 40-Pin-GPIO. Die Stromversorgung erfolgt über ein USB-C-Kabel. Auf der Platine finden sich Konnektoren für einen Reset-Knopf und einen An/Aus-Schalter.

Bei der wesentlichen Komponente, dem Grafikchip, handelt es sich um eine GPU auf Basis der „Maxwell"-Architektur mit 128 Cores. Der Chip integriert Hardware-En- und -Decoder für diverse Videoformate inklusive H.264 und H.265, was im Machine Learning der Bilderkennung zugute kommt. Optisch fällt der sehr große Kühlkörper auf dem Grafikchip auf, der mit einem Symbol vor Hitzeentwicklung warnt. Auch in Trainingsvideos weist

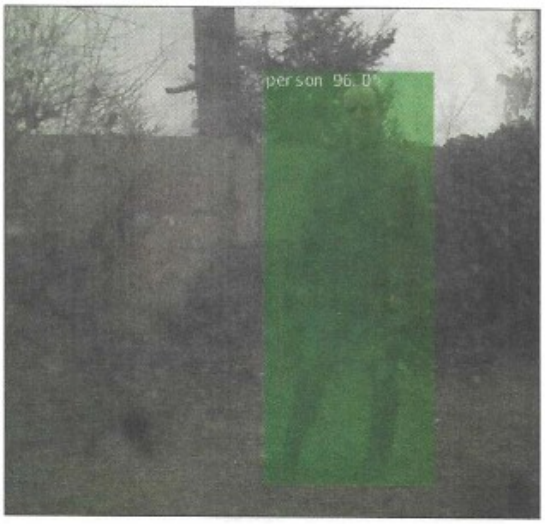
Nvidia mehrfach darauf hin, dass der Chip wirklich heiß werden kann, wenn die GPU wirklich arbeitet, etwa beim Trainieren eines neuronalen Netzes. Die Luft rund um den Kühlkörper heizt sich merklich auf, worauf man beim Aufbau des Jetson Nano unbedingt achten sollte.

Software

Nvidia beschreibt die Ersteinrichtung auf einer eigenen Webseite, auf der sich auch ein Link zum Download des Betriebssystems findet. Das entsprechende Image spielen Sie per Dd oder mit vergleichbaren Werkzeugen auf eine SD-Karte. Es handelt sich um ein Ubuntu 18 für ARM64, der Nano läuft also standardmäßig mit 64 Bit. Dem Image hat Nvidia einige Repositories hinzugefügt, sodass Treiber für die Hardware sowie Werkzeuge und Bibliotheken für das Machine Learning bereitstehen.

Zur Ersteinrichtung gibt es zwei Möglichkeiten: Bei Anschluss von Monitor, Tastatur und Maus melden Sie sich wie an jedem normalen Linux-Rechner an. In einem grafischen Desktop findet eine dialoggeführte Installation statt, bei der Sie einen Benutzer anlegen sowie Hostnamen und Netzwerk konfigurieren. Die SD-Karte lässt sich dabei direkt so umpartitionieren, dass der maximale Platz ausgenutzt wird. Steht kein Monitor zur Verfügung, erledigen Sie die Installation über eine serielle Konsole auf dem Micro-USB-Port, die Sie mit einem Terminalprogramm wie Minicom oder Screen erreichen. Danach erfolgt der Zugriff per SSH.

Jedem geschulten Admin fällt direkt auf, dass auf dem System Docker läuft. Die meisten Nvidia-Schulungsvideos greifen auf Docker-Container zurück, innerhalb derer der Hersteller die Umgebungen passend zum Video vorbereitet hat. Andernfalls müsste der Schüler vor dem Start jede Menge Python-Module installieren. Allerdings macht diese Vorgehensweise den Start der Trainingseinheiten etwas komplizierter, weil man den Docker-Run-Kommandos jeweils Volume-Mounts mitgeben muss. Das könnte Docker-Einsteiger etwas verwirren.



1 **Ein Beispiel für Objekterkennung mit dem Jetson Nano.**

Machine-Learning-Training

Auf der letzten Seite der Beschreibung zur Ersteinrichtung bietet Nvidia bei den weiterführenden Links auch Trainingsvideos an, die einen guten Einstieg in den Umgang mit dem Jetson Nano vermitteln. Eine weitere Videoreihe zeigt auch das vollständige Einrichten inklusive Auspacken und Bespielen der SD-Karte, wobei Jupyter Notebooks (im Container) zum Einsatz kommt, um dem ML-Schüler die Anwendung und das Trainieren von Netzen zu zeigen. Die Videos vermitteln ein Verständnis dafür, an welchen Schrauben man beim Training drehen kann, um die Erkennungsrate zu verbessern. Dabei geht es im Wesentlichen um Bilderkennung, weshalb Sie eine passende USB-Kamera zur Hand haben sollten, um selbst mit den Netzen zu arbeiten.

Ein anderer Video-Track ist etwas kommandozeilenorientierter. Das zugehörige Git-Repository laden Sie entweder herunter und arbeiten dann, wie im Video vorgeschlagen, mit Containern, oder Sie installieren die Software aus dem Repository, sodass Sie sich die Container-Aufrufe ersparen. Auch hier werden nacheinander Bild- und Objekterkennung vorgeführt. Dabei geht das Training allerdings davon aus, dass Sie mit Monitor und Tastatur direkt am Jetson Nano sitzen. Gegebenenfalls lässt sich die Fensterausgabe der verwendeten Programme Imagenet und Detectnet aber per RTP an den Rechner weiterleiten, von dem aus Sie den Jetson Nano headless nutzen. Dort müssen Sie nur einen passenden RTP-Empfänger starten (siehe Kasten RTP-Empfang).

Beide Videoreihen verwenden vordefinierte Netze und Netzmodelle, erläutern aber auch, wie man diese austauschen kann. In den Kursteilen, bei denen trainiert wird, kommt jeweils ein bereits vortrainiertes Netz zum Einsatz, dem Sie neue Objekte zur Erkennung beibringen. Das vermittelt einen guten Eindruck davon, mit welchen Freiheitsgraden Sie spielen können, um die Ergebnisse zu verbessern, ohne gleich am Anfang mit zu viel Wissen zu überfrachten. Die Möglichkeit, verschiedene Netze mit

RTP-Empfang

Um via VLC oder MPlayer einen eingehenden RTP-Strom anzunehmen und anzuzeigen, erstellen Sie zunächst eine SDP-Datei (Session Description Protocol), die beschreibt, was da auf welchem Port ankommt. Anschließend übergeben Sie dem Player die fraglichen Daten zum Abspielen. Eine SDP-Datei für den Empfang der Ausgabe der Jetson Utilities sieht etwa wie in Listing 1 aus. Die Tools verschicken ihre Ausgabe in H.264-Kodierung.

Listing 1: SDP-Datei

```
v=0
c=IN IP4 127.0.0.1
m=video 1234 RTP/AVP 96
a=rtpmap:96 H264/90000
```

Dateien zum Artikel herunterladen unter
www.lm-online.de/dl/45627

Listing 2: Gst-launch-1.0

```
$ gst-launch-1.0 -v souphttpsrc
 location=http://edimax-cam/snapshot.cgi \
  user-id=camera user-pw=camera do-timestamp=true ! multipartdemux ! \
  image/jpeg,width=1280,height=960 ! jpegdec ! videoconvert ! \
  omxh264enc profile=high preset-level=1 bitrate=900000 ! \
  'video/x-h264,stream-format=(string)byte-stream' ! rtph264pay ! \
  udpsink host=127.0.0.1 port=1234
$ detectnet --input-codec=h264 rtp://@:1234 rtp://Anzeigerechner:5000
```

Listing 3: Objekterkennung von der IP-Kamera in Python

```
import jetson.inference
import jetson.utils
import sys
import cv2
import time

frame_width = 1280
frame_height = 960

net = jetson.inference.detectNet("ssd-mobilenet-v2", sys.argv, 0.5)
cap = cv2.VideoCapture("souphttpsrc location=http://edimax-cam/
snapshot.cgi user-id=camera user-pw=camera do-timestamp=true !
multipartdemux ! image/jpeg,width=1280,height=960 ! jpegdec !
videoconvert ! appsink")
font = jetson.utils.cudaFont(size=jetson.utils.adaptFontSize(
frame_width))

while True:
  ret,frame = cap.read()
  if ret == False:
    break
  else:
    frame_rgba = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
    cuda_frame = jetson.utils.cudaFromNumpy(frame_rgba)
    detections = net.Detect(cuda_frame, overlay="box,labels,conf")
    for detection in detections:
      detname =  net.GetClassDesc(detection.ClassID)
      if detection.Confidence > 0.7:
        print(detname+" "+str(detection.Confidence))
        font.OverlayText(cuda_frame, frame_width, frame_height,
"{:f}% {:s}".format(detection.Confidence * 100, detname), 10, 10,
font.White, font.Gray40)
        jetson.utils.cudaDeviceSynchronize()
        filename = "out-"+str(time.time())+".jpg"
        jetson.utils.saveImageRGBA(filename, cuda_frame,
frame_width, frame_height)
```

derselben Aufgabe auszuprobieren, erlaubt Ihnen auch einen Vergleich der Netzarchitekturen.

Anwendungsfall

Der Autor besitzt eine IP-Kamera von Edimax, die eine einfache Bewegungserkennung bietet. Die Bilder, die die Kamera auf einen FTP-Server hochlädt, sehen nach Embedded Linux und dem Programm Motion aus. Die Kamera bietet aber auch einen Live-Stream im MJPEG-Format an. Um zu erkennen, ob Personen durchs Bild laufen, lässt sich das Programm Detectnet verwenden. Anhand der Anweisungen aus den Trainingsvideos können Sie dabei vortrainierte neuronale Netze herunterladen und verwenden.

Im einfachsten Fall folgt der Aufruf dem Muster `detectnet /dev/video0` oder `detectnet /dev/video0 rtp://ziel-server:5000`. Das erste Beispiel liest von der ersten direkt angeschlossenen USB-Kamera und stellt die Ergebnisse lokal auf dem Bildschirm dar. Beim Headless-Betrieb schickt der zweite Aufruf die Daten an einen RTP-Empfänger; Abbildung 1 zeigt den entsprechenden Output. Das als Standardeinstellung vorgegebene, vortrainierte Netz heißt SSD-Mobilenet-v2 und verwendet Tensorflow als Engine.

Um in Echtzeit an die Daten der IP-Kamera zu gelangen, verwendete der Autor Gstreamer, ein sehr flexibles Toolkit zur Manipulation von Audio-Video-Strömen aus allen möglichen Quellen und in vielen Formaten. Neben Kommandozeilenwerkzeugen umfasst das Paket auch Bibliotheken mit Schnittstellen zu allen gängigen Programmiersprachen. Gstreamer erwies sich als momentan einziges Toolkit, das man auf dem Jetson Nano mit voller Hardware-Unterstützung verwenden kann, was dem Autor beim Umwandeln der Videodateien einige graue Haare wachsen ließ.

Für den einfachen Testaufbau kommt das Programm Gst-launch-1.0 zum Einsatz, dem man beim Aufruf eine Reihe von Queues mitgibt – etwa eine Datei als Eingang, eine Umwandlungsmethode und eine andere Datei als Ausgang. Elemente einer solchen Queue trennt jeweils ein Ausrufezeichen. Bei einer Datei mit mehreren Strömen (etwa Video- und Tonspur) kann beziehungsweise muss man

diese getrennt behandeln, um sie gegebenenfalls wieder in einem Container zusammenzufassen. Da die IP-Kamera nur eine Videospur bietet, genügt eine Queue, die Folgendes abwickeln muss:

- das Auslesen des MJPEG-Stroms per HTTP (mit Benutzerauthentisierung),
- das Auspacken des Videostroms aus dem HTTP-Strom,
- das Dekodieren des MJPEG-4-Formats,
- dessen Umwandeln ins H.264-Format, das Detectnet benötigt, und
- das Senden dieses Stroms per RTP an Localhost.

Der Jetson Nano übernimmt in diesem Fall wegen der entsprechenden Hardware-Unterstützung der GPU auch das Konvertieren nach H.264. Der Aufruf von Gst-launch-1.0 sieht so aus wie in den ersten sechs Zeilen von Listing 2. Durch Verändern des Parameters `bitrate` (Zeile 4) lässt sich die Bildqualität beeinflussen; `omxh264enc` verwendet hier den Hardware-beschleunigten Encoder für H.264. Dann kommt Detectnet zum Einsatz (letzte Zeile). Schließlich startet der Anwender auf dem darstellenden Rechner noch den Player mit der SDP-Datei aus Listing 1; nach einem Moment des Einschwingens erscheint das Bild aus Abbildung 1.

Mit etwas zusätzlichem Python-Code (Listing 3) lässt sich das ganze nun als Alarmanlage verwenden. Im Unterschied zur CLI-Pipeline kann der Autor hier etwas effizienter arbeiten, da im Code nur die Bilder aus dem MJPEG-Videostrom zur Verfügung stehen müssen. Die liefert in Python das CV2-Modul, dem man eine GST-Queue mitgibt, die in `appsink` endet (Zeile 11). Das neuronale Netz initialisiert das Modul `jetson.utils` aus dem entsprechenden Repository. Die Hauptschleife liest das jeweils nächste Bild, wandelt es in ein Format um, mit dem Tensorflow etwas anfangen kann, und ruft die Objekterkennung auf. Erkennt das Netz Objekte, arbeitet die Schleife sie ab. Bei einer Erkennungswahrscheinlichkeit von über 70 Prozent gibt das Skript eine Meldung aus und speichert das Bild samt eingeblendeten Boxen um die erkannten Objekte und Zeitstempel ab.

Videos umwandeln

Der Autor verwendet Tvheadend als Videorekorder für Fernsehprogramme. Die Software schreibt die rohen MPEG2-Ströme aus der DVB-C-Quelle auf die Festplatte – nicht besonders platzsparend, dafür genügt aber ein RasPi 2 für die Arbeit. Das Umwandeln einer solchen Aufnahme in das weitaus kompaktere H.265-Format dauert jedoch selbst mit leistungsfähigen CPUs länger.

Dabei leistet die GPU im Jetson Nano allerdings im Verhältnis zum Stromverbrauch des Geräts Erstaunliches. Eine 70-Minuten-Aufnahme in SD (720 x 576) wandelte der Winzling in gut 5 Minuten in eine H.265-Datei um. Zum Platzsparen auf der Festplatte leistet der Jetson Nano also gute und effiziente Dienste. Der Aufruf von Gstreamer ist dabei allerdings etwas trickreich, da es Audio- und Videostrom zu behandeln gilt. Im Ganzen sieht das dann so aus wie in Listing 4.

Die erste Queue (beginnend mit `filesrc`) liest die Rohdaten aus der Datei `aufname.mkv`. Den entsprechenden Matroska-Container zerlegt `matroskademux` in seine Einzelteile. Die Anweisung `name=demux` benennt den Ausgang der Queue (in der Gstreamer-Sprache eine Sink), sodass er sich später referenzieren lässt. Die dritte Zeile erzeugt das Ziel der Aktion – wiederum einen Matroska-Container, der in der Datei `ziel.mkv` landet.

Anschließend geht es darum, etwas aus den beiden Strömen zu machen, die aus dem Quellcontainer kommen. Diese zwei Ströme lassen sich mit *`Name.audio_index`* ansprechen. Bei mehreren Tonspuren in einer Aufnahme kann es also auch `audio_1` geben; in unserem Beispiel existiert jedoch nur die Spur `audio_0`. Die leitet der Aufruf durch Mpegaudioparse, um das MPEG-Format zu verstehen, und anschließend gleich in den Zielcontainer, da hier keine Umwandlung stattfinden soll (Zeile 4).

Erst der Code ab Zeile 5 bringt die GPU zum Arbeiten. Der Videostrom läuft ebenfalls durch einen MPEG-Parser und wird dann via Omxmpeg2videodec mit Hardware-Beschleunigung dekodiert. Nun folgt ein Puffer in Form von queue, bevor es direkt mit der Kodierung mittels Omxh265enc weitergeht.

Dessen Parameter beziehen sich auf die Qualität des Videos: `preset-level 3` sorgt für hohe Qualität, und die Bitrate steuert die entstehende Datenmenge, was ebenfalls Einfluss auf das Bild hat. Die letzten beiden Parameter aktivieren einen zweifachen Durchlauf und eine variable Kontrollrate. Bevor das Ergebnis an den Ziel-Container geht, findet sich wiederum ein Puffer in der Queue, da es sonst zu Verklemmungen kommt.

Fazit

Der Nvidia Jetson Nano 2GB ist etwas teurer als ein Raspberry Pi 4, dafür aber auch leistungsfähiger. Der Einstieg ins Machine Learning kommt damit günstiger als beim Einbau einer entsprechenden Grafikkarte in einen PC. Besonders der Stromverbrauch von 5 bis 10 Watt unterbietet den einer aktuellen Grafikkarte deutlich. Auch für das Umwandeln von Videos eignet sich der Jetson Nano bestens. Die zugehörige Distribution enthält zudem Hardware-Beschleunigungsmodule für Chrome, sodass sich der Rechenzwerg vermutlich auch als Desktop eignet – das hat der Autor aber nicht näher untersucht. (*jlu*)

Weitere Infos und interessante Links

www.lm-online.de/qr/45627

Listing 4: Video rekodieren

```
01 $ gst-launch-1.0 filesrc \
02   location=aufnahme.mkv ! matroskademux name=demux \
03   matroskamux name=mux ! filesink location=ziel.mkv \
04   demux.audio_0 ! \ mpegaudioparse ! queue ! mux. \
05   demux.video_0 ! mpegvideoparse ! omxmpeg2videodec ! \
06   queue ! \ omxh265enc preset-level=3 bitrate=750000 \
07   EnableTwopassCBR=true control-rate=variable ! \
08   queue ! mux.
```