RSTensorFlow: Analyse und Optimierung (vorläufiger Titel)

Michael Engel

I. Abstract

- Zusammenfassender Abstract

II. Einleitung

In den vergangenen Jahren haben sich mobile Geräte, wie beispielsweise Smartphones, zu Assistenten des Alltags entwickelt. Auf diesen kleinen Helfern werden viele komplexe Anwendungen und Funktionen genutzt, welche verschiedenste Deep Learning Modelle verwenden. So werden beispielsweise Deep Recurrent Networks Neural Networks zur Spracherkennung Graves et al. (2013) genutzt. Zur Erkennung von Objekten werden Deep Convolutional Networks eingesetzt Krizhevsky et al. (2012). Deep Learning Modelle führen viele sehr rechenintensive Berechnungen durch. Deshalb wird bevorzugt eine Art Client-Server-Anwendung umgesetzt, welche das Deep Learning Modell auf einem separaten Server ausführt. Die mobilen Clients schicken über eine entsprechende App die benötigten Eingangsdaten für das Netzwerk an den Server und dieser schickt das Ergebnis des Deep Learning Netzwerks zurück an den Client. Dies ist sowohl zeit- als auch kostenintensiv. Um Deep Learning Netzwerke direkt auf dem Mobilgerät zu verwenden, bieten Frameworks wie TensorFlow (https://www.tensorflow.org/) eine eigene Variante der Software an. Jedoch nutzen diese nicht alle Rechenressourcen des Endgeräts, sondern lediglich die CPU. Für größere Applikationen ist dies oft nicht ausreichend, um in angemessener Zeit ein adäquates Ergebnis zu erhalten. Projekte wie RSTensorFlow Alzantot et al. (2017) versuchen daher eine Unterstützung der GPU in das bestehende OpenSource-Framework TensorFlow zu integrieren und den Erfolg anhand geeigneter Metriken zu bewerten.
Die Veröffentlichung zum Projekt *RSTensorFlow* Alzantot et al. (2017) dient als zentrale Arbeit dieses Papers und stellt den Ausgangspunkt dar. Die in Alzantot et al. (2017) beschriebenen Experimente werden in diesem Paper nachgestellt und für andere Geräte, welche nicht direkt von Google hergestellt wurden, durchgeführt. Über die angepasste *TF Classify* App werden die Ausführungszeiten gemessen. Ebenso wird mit Hilfe der *Trepn Profiler* App die Auslastung der CPU und GPU überprüft. Diese Anwendung dient ebenfalls dem Aufzeichnen weiterer Metriken, wie beispielsweise der Auslastung des Speichers. Diese wurde bereits in Alzantot et al. (2017) als ein mögliches Bottleneck hinsichtlich der Performance vermutet.

Ab hier falls Optimierung von Conv2D

Dazu wird eine Analyse der RSTensorFlow-Implementierung (todo: github-link) durchgeführt, um ein Verständnis über die Architektur und die bisherige Implementierung der Conv2d-Operation zu gewinnen. Auf Basis dieser Analyse wird eine Optimierung der Conv2D-Operation durch sogenanntes Unfolding versucht zu erreichen. In Chellapilla et al. (2006) sowie in Rajbhandari et al. (2017) werden Convolutional Networks optimiert. Hierfür wägen beide Arbeiten das Potential des Unfoldings und erläutern die Funktionsweise.

III. Related Work

In Alzantot et al. (2017) wird die derzeit noch eher mangelhafte Unterstützung für tiefe neuronale Netze auf mobilen Endgeräten erläutert. Die meisten Deep Learning Frameworks bieten zwar eine Variante für Mobilgeräte, können jedoch lediglich die CPU als Recheneinheit nutzen. Aktuelle Forschungen beschäftigen sich damit, diese Frameworks für Mobilgeräte zu erweitern, sodass weitere Komponenten für die komplexen Berechnungen tiefer neuronaler Netze genutzt werden können. Hier steht insbesondere die GPU als weitere Rechenressource im Fokus der Forschung für unterschiedlichste Anwendungen neuronaler Netze. Um die GPU nutzen zu können, wird von verschiedenen Forschungsprojekten entweder OpenCL ope oder RenderScript ren verwendet. Im Bereich der Continuous Vision Applikationen wird in Huynh et al. (2017) ein GPU-basiertes Deep Learning Framework vorgestellt. Hier wird zur Nutzung der GPU OpenCL ope in Kombination mit Vulkan vul eingesetzt. Ebenso greift wird in Huynh et al. (2016) auf OpenCL ope zurückgegriffen, um ein Framework für Deep Convolutional Neural Networks für mobile Geräte mit GPU-Unterstützung umzusetzen. Ein weiteres Forschungsprojekt ist CNNDroid Latifi Oskouei et al. (2016). Bei der Implementierung CNNDroid handelt es sich um eine OpenSource-Bibliothek für trainierte CNN auf Android-Geräten. Im Gegensatz zu den anderen Projekten integriert RSTensorFlow die GPU-Unterstützung in das beliebte, bereits bestehende Deep Learning Framework *tensorflow*.
Für die Evaluierung des umgesetzten Frameworks werden von allen vorgestellten Forschungsprojekten Metriken wie die Ausführungszeit betrachtet. Diese Arbeit greift die allgemeine Vorgehensweise der in Alzantot et al. (2017) vorgestellten Experimente auf und wendet dies für andere Geräte an. Damit soll eine Validierung der in Alzantot et al. (2017) vorgestellten Ergebnisse erzielt werden. Ebenso wird die Speicherauslastung als eine weitere Metrik betrachtet. Diese wird als möglicher Grund für die verschlechterte Ausführungszeit bei der Conv2D-Operation von den Autoren von Alzantot et al. (2017) vermutet.

IV. RSTensorFlow

+ was ist rstensorflow? + tensorflow, modifiziert, damit renderscript für rechenintensive operationen verwendet wird + wie wird es installiert? + was wird installiert? 3 Apps, kurz erklären... relevant nur die classify app, überleitung zu den operationen (wobei anpassung am kernel == alle apps von betroffen?)

A. Grobe Architektur

+ welche files (.cc, .h, .java etc) sind relevant (also wo wurden anpassungen für rstensorflow umgesetzt?) + diagramme

B. Modifizierte Operationen

+ Welche Operationen wurden damit implementiert? Jeweils kurz erklären, was diese Operation macht + matmul + conv2d - http://cs231n.github.io/convolutional-networks/

V. Vorbereitungen

+ versucht die prebuilds von rstensorflow (welche auf der homepage/github bereit gestellt sind) zu verwenden - für bestmögliche vergleichbarkeit bzw rekonstruktion der experimente - lediglich die eigen-prebuild (ohne rs) ist nutzbar - die rs-prebuild schmiert ab + nach längerer suche, konnte festgestellt werden, dass eine .so nicht geladen werden konnte + es handelt sich dabei um die .so, welche im rahmen des rstf projekts umgesetzt wurde + daher ist das eigenständige kompilieren von rstensorflow bzw. der android demo nötig was in V-A dargestellt wird + optional: dies wird gleichzeitig als vorbereitung für die optimierung der conv2d betrachtet

A. Der Build-Prozess

+ tools - bazel, android sdk, ndk

B. Rekonstruktion der Experimente

+ code analyse + grafiken für matmul und conv2d - wie kommen die auf die jeweiligen werte der x-Achse? + conv2d: Anzahl Filter - conv_ops.cc - Da keine Filteranzahl mit 1, 3 oder xxx gefunden wurde, und leider keine genaue beschreibung vorhanden ist, wie die Grafik zustande kommt, ist davon auszugehen, dass ein separates programm verwendet wurde (diese Hypothese wird gestützt durch fehlendes loggen der anzahl dimensionen im modifizierten tf quellcode) - Anzahl Filter ist Hyperparameter - K == Anzahl der Filter == Output-Dimension3 (Breite und Höhe durch stride definiert)

VI. Experimente

+ Kurze Einleitung, was in dem Kapitel gemacht wird + Hardware-Tabelle (inkl. die Hardware aus Alzantot et al. (2017))

A. Verwendete Metriken

Mögliche weitere Metriken wie in Kapitel 7 von Huynh et al. (2017) beschrieben ggf. verwenden

- Ist Metrik für dieses Projekt sinnvoll?
- Ist Metrik für dieses Projekt umsetzbar (Aufwand etc.)?

Verwendete Metrik in Alzantot et al. (2017): Ausführungszeit + im Code von tensorflow integriert + gibt auskunft über die Zeit einer einzelnen matmul-/conv2d operation und eines kompletten pfades + über adb logcat -s TF_ANDROID_LOG können diese in eine Datei gelenkt werden

Metrik CPU Load + über trepn profiler + gibt auskunft über die auslastung der ressource

Metrik GPU Load + über trepn profiler + gibt auskunft über die auslastung der ressource

Metrik RAM (Memory Space): + über trepn profiler + nicht straight forward, da von Trepn Profiler RAM des kompletten Systems gemessen wird + daher System im Ruhezustand profilen (inkl. laufender Trepn app) und (durchschnittlichen) RAM ermitteln und vom später gemessenen abziehen

B. Aufbau des Experiments

+ Software + Mit was werden die Daten erzeugt? - trepn profiler (app) -> Einstellungen definieren und speichern! - log ausgaben (direkt von tensorflow demo) -> siehe VI-C + Hardware + ggf. abbildung? Beschreibung des Experiment-Aufbaus

- Wahl der Android-Geräte (Samsung J5 und ggf. weitere), deren Android-Version und Hardware)
- Wie werden die gewählten Metriken gemessen?

C. Anpassungen von TensorFlow

+ Die Trepn Profiler öut-of-the-boxäber die logausgaben müssen aufbereitet werden, da die vorhandenen logausgaben unzureichend sind + adb logging da nur ëin Stream"verfügbar - alles mit adb logcat TF_ANDROID_LOG in eine CSV-Datei Format: <Feld1>|<Feld n>|... Dabei ist Feld1 der Index, um was für eine Operation es sich handelt (da von classify sowohl conv2d als auch matmul ausgeführt wird) Für matmul: + die matrizengröße Für conv2d: + anzahl filter (== outdepth) + stride] + padding - Berechnung der Anzahl der matmul operationen -> hat einfluss auf ausführungszeit? ggf. mit unfolding doch verbesserbar? (für Ausblick dann) + filtergröße] + inputgröße]

D. Durchführung

+ trepn profiler: profile system - profiling erst im standard modus um die normale menge an memory zu erhalten - dann app starten

E. Auswertung

+ was kamen für ergebnisse heraus? + vergleich zu Alzantot et al. (2017) - deutet auf eine generelle verschlechterung mit rs hin - direkte vergleichbarkeit aber leider nicht gegeben, da die genaue erhebung der daten unbekannt

VII. Optional: Optimierung der Conv2D-Operation

- Kurze Beschreibung der Convolution-Operation (wie in Li et al. (2017))
- Ist-Analyse des Quellcodes der Conv2D-Operation
- Unfolding (Chellapilla et al. (2006)) als mögliche Optimierung von Conv2D für RSTensorFlow
- Implementierung der (möglichen) Optimierung

VIII. Fazit und Ausblick

Einige Probleme: + erstmal zum laufen bekommn (von rs-tensorflow bereitgestellter mod. build funzt nicht, build prozess etc.) + rekonstruktion der experimente Hier kann aus dem referenzpaper bzw. dessen website zitiert oder sich darauf bezogen werden: ist schweres zeug, rs nativ in tensorflow zu integrieren

Schlussfolgerung: + nur weil es die gpu ist bzw. eine zusätzliche ressource, muss es nicht unbedingt schneller sein + ggf derzeit einfach nicht ausgereift -> standards! + oder renderscript einfach ungeeignet

- Fazit
  - Erzielt RSTensorFlow ähnliche Ergebnisse bei den hier getesteten Geräten im Vergleich zu den Nexus-Geräten, welche in Alzantot et al. (2017) verwendet wurden?
  - Konnte eine Verbesserung von Conv2D erzielt werden?
- Ausblick (oder Future Work)
  - TensorFlow Lite
  - OpenCL statt RenderScript

IX. Zukünftige Arbeiten

+ future work: unfolding umzusetzen (wie auswertung zeigt: sehr viele matmuls bei conv und damit ist nur noch 1 nötig -> performance-memory-tradeoff? eher unwahrscheinlich, da für conv2d und rs eh die tensoren kopiert werden und bei auswertung kaum veränderungen bei dem genutzten speicher zu sehen waren) und somit

Literatur

Opencl. URL https://www.khronos.org/opencl/. Zuletzt besucht: 06.12.2017.

Renderscript. URL https://developer.android.com/guide/topics/renderscript/compute.html. Zuletzt besucht: 06.12.2017.

Vulkan. URL https://www.khronos.org/vulkan/. Zuletzt besucht: 06.12.2017.

M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava. RSTensorFlow: Gpu enabled tensorflow for deep learning on commodity android devices. 1st International Workshop on Deep Learning for Mobile Systems and Applications, 2017. URL https://nesl.github.io/RSTensorFlow. Software available from nesl.github.io.

K. Chellapilla, S. Puri, and P. Simard. High Performance convolutional neural networks for document processing. Tenth International Workshop on Frontiers in Handwriting Recognition, La Baule (France), 2006. URL http://www.suvisoft.com.

A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013. URL http://arxiv.org/abs/1303.5778.

L. N. Huynh, R. K. Balan, and Y. Lee. DeepSense: A gpu-based deep convolutional neural network framework on commodity mobile devices. 2016 Workshop on Wearable Systems and Applications, 2016.

L. N. Huynh, Y. Lee, and R. K. Balan. DeepMon: Mobile gpu-based deep learning framework for continuous vision applications. 15th Annual International Conference on Mobile Systems, Applications, and Services, 2017.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

S. S. Latifi Oskouei, H. Golestani, M. Hashemi, and S. Ghiasi. Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android. In *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, pages 1201–1205, 2016.

F.-F. Li, J. Johnson, and S. Yeung. , 2017. URL http://cs231n.stanford.edu. Zuletzt besucht: 16.11.2017.

S. Rajbhandari, Y. He, O. Ruwase, M. Carbin, and T. Chilimbi. Optimizing cnns on multicores for scalability, performance and goodput. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, pages 267–280, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4465-4. doi: 10.1145/3037697.3037745. URL http://doi.acm.org/10.1145/3037697.3037745.