Fachhochschule Südwestfalen  
Standort Meschede  
Bachelor of Science - Wirtschaftsinformatik

**Whitepaper Business Intelligence II  
Sommersemester 2022**

Selenium

Vorstellung von Selenium zum abfragen und sammeln von Daten

**Vorgelegt von**

Fabian Felix Selbach fabian.selbach@edu.siemens-energy.com 30113859

**Ort, Abgabetermin**

Leverkusen, 04.06.2022

Inhaltsverzeichnis

[Inhaltsverzeichnis I](#_Toc81737334)

[Abbildungsverzeichnis II](#_Toc81737335)

[1 Einleitung 1](#_Toc81737336)

[1.1 Problemstellung 1](#_Toc81737337)

[1.2 Aufbau der Arbeit 1](#_Toc81737338)

[2 Literaturverzeichnis 16](#_Toc81737339)

[3 Eigenständigkeitserklärungen 17](#_Toc81737340)

[Eigenständigkeitserklärung Fabian Felix Selbach, 30113859 17](#_Toc81737341)

Abbildungsverzeichnis

[Abbildung 1: PyTorch Bibliotheken importieren 3](#_Toc95729728)

[Abbildung 2: PyTorch Aufbau neuronales Netzwerk 4](#_Toc95729729)

[Abbildung 3: PyTorch Netzwerk forward Methode 5](#_Toc95729730)

[Abbildung 4: PyTorch Trainingsmethode 5](#_Toc95729731)

[Abbildung 5: PyTorch Testmethode 6](#_Toc95729732)

[Abbildung 6: PyTorch Datensatz laden 7](#_Toc95729733)

[Abbildung 7: PyTorch Verlust- und Optimierungsfunktion 8](#_Toc95729734)

[Abbildung 8: PyTorch Ausführungsschleife 8](#_Toc95729735)

[Abbildung 9: PyTorch Ergebnis 1. Epoche 9](#_Toc95729736)

[Abbildung 10: PyTorch Ergebnis 10. Epoche 9](#_Toc95729737)

[Abbildung 11: TensorFlow Bibliotheken importieren 10](#_Toc95729738)

[Abbildung 12: TensorFlow neuronales Netzwerk definieren 10](#_Toc95729739)

[Abbildung 13: TensorFlow Verlust- und Optimierungsfunktion 11](#_Toc95729740)

[Abbildung 14: TensorFlow Trainings- und Testfunktionen 12](#_Toc95729741)

[Abbildung 15: TensorFlow Datensatz laden 13](#_Toc95729742)

[Abbildung 16: TensorFlow Ausführungsschleife 14](#_Toc95729743)

[Abbildung 17: TensorFlow Ergebnis 14](#_Toc95729744)

# Einleitung

## Problemstellung

Mit dem immer prominenteren Einsatz von Deep Learning geht auch eine immer größer werdende Auswahl an Framework einher, die es einem erlauben neurale Netze zu trainieren und einzusetzen.

Zwei sehr beliebte Deep Learning Frameworks, PyTorch und TensorFlow, werden in dieser Arbeit miteinander verglichen um einen Überblick beider Frameworks zu bieten und die Einsatzgebiete abzugrenzen.

## Aufbau der Arbeit

Die Arbeit wird zu Beginn PyTorch und TensorFlow vorstellen, um so einen generellen Überblick über die beiden Frameworks zu bieten. Darauffolgend wird es Anwendungsbeispiele beider Frameworks geben, an denen ein praktischer Vergleich durchgeführt wird.

# Frameworkvorstellung

## 2.1. PyTorch[[1]](#footnote-2)

PyTorch ist ein Ende-zu-Ende Framework für maschinelles Lernen für Python basierend auf Torch. PyTorch bietet eine schnelle und flexible Umgebung, sowohl zum Experimentieren und Entwickeln als auch für den Einsatz in produktiven Systemen.

Bereitgestellt wird diese Flexibilität durch benutzerfreundliche Front-Ends, die Möglichkeit, das Lernen der neuralen Netzwerke verteilt durchzuführen und der Vielzahl an bestehenden Tools und Bibliotheken, die PyTorch ergänzen.

PyTorch unterstützt mehrere Geräte zur Ausführung und zum Anlernen der neuronalen Netze, wie z.B. CPUs und GPUs mit CUDA Unterstützung von Nvidia.

## 2.2. TensorFlow[[2]](#footnote-3)

TensorFlow ist, ähnlich wie PyTorch, ein Ende-zu-Ende Framework für maschinelles Lernen von Google. Es besteht aus einer Kernbibliothek auf welcher dann mehrere Module aufbauen, um so einerseits vielseitige Möglichkeiten zum maschinellen Lernen anzubieten und andererseits auch mehrere Programmiersprachen und Einsatzplattformen zu Unterstützen.

Während PyTorch ein vollständiges, größtenteils in sich geschlossenes, System bietet beruht TensorFlow auf das Einbinden von weiteren Bibliotheken die auf die Kernbibliothek aufbauen, somit ist ein wesentlicher Unterschied in der Herangehensweise bei, da man kaum eine Anwendung ausschließlich mit TensorFlow erstellt, sondern häufig Hilfsbibliotheken verwendet.

# Anwendungsbeispiel

Das für diese Arbeit gewählte Anwendungsbeispiel ist ein neuronales Netzwerk zur Erkennung von handschriftlichen Ziffern in Bildern.  
Trainiert und getestet wird dieses anhand der MNIST Datenbank, welche aus 60.000 Trainingsbildern und 10.000 Testbildern besteht. Sie ist eine modifizierte und zusammengefügte Variante zweier NIST Datensätze, „Special Database 1“ und „Special Database 3“, welche einerseits eine Sammlung von handgeschriebenen Ziffern von Amerikanischen Schülern und andererseits von Angestellten des „US Bureau of the Census“ ist. Des Weiteren wurden diese Datensätze modifiziert damit alle Ziffern in einer 28\*28 Pixel graustufen Bilddatei abgespeichert werden können.

## 3.1. PyTorch

Zu Beginn werden die benötigten Bibliotheken und Namespaces importiert um alle nötigen Methoden von PyTorch nutzen zu können.

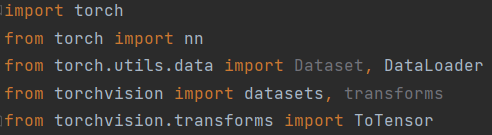


Abbildung 1: PyTorch Bibliotheken importieren

Die importierten Bibliotheken sind PyTorch und Torchvision gefolgt von den Paketen für neuronale Netze „nn“, den mitgelieferten Datasets und einem DataLoader sowie Transformationsfunktionen aus Torchvision.

Als nächstes wird das gewünschte Netzwerk erstellt und die benötigte Konfiguration der Eingangs- und Zwischen- und Ausgangsebenen definiert.

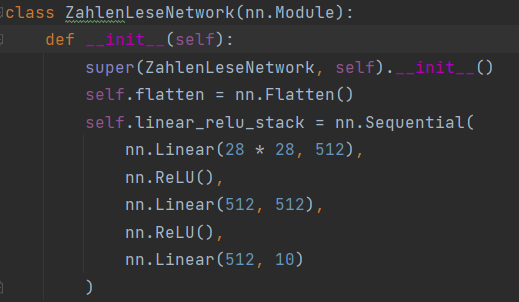


Abbildung 2: PyTorch Aufbau neuronales Netzwerk

Das hier gewählte Netzwerk verfügt über drei Ebenen, beginnend mit einer Ebene mit 28\*28 Eingängen, die die Bilder des MNIST Datensatzes entgegennehmen und 512 Ausgängen die an die darauffolgende Ebene ihrer Daten weitergeben. Diese darauffolgende Ebene hat dementsprechend 512 Eingänge und 512 Ausgänge. Die letzte Ebene weißt 512 Eingänge auf, und hat 10 Ausgänge, wovon jeder Ausgang die relative Sicherheit für jede Ziffer ausgibt, die das Netzwerk anhand der Eingabedaten bestimmt hat.

Zu Beginn erzeugen alle Eingaben eine gleiche Sicherheit am Ausgang des Netzwerkes dadurch, dass das Netzwerk noch nicht trainiert wurde und keine Gewichtung verteilen kann um eine Antwort zu treffen. Damit man dieses Netzwerk nun trainieren kann wird einerseits eine Methode benötigt um auf das interne Netzwerk unserer Klasse zuzugreifen und andererseits eine Trainingsmethode.

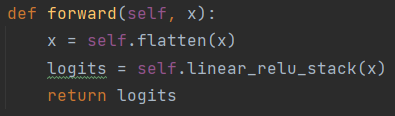


Abbildung 3: PyTorch Netzwerk forward Methode

Diese Methode weißt unsere Klasse an, wie auf das Netzwerk zugegriffen werden soll und bietet die Möglichkeit Daten ggf. aufzubereiten bevor sie durch das Netzwerk verarbeitet werden. In unserem Beispiel werden die Eingabedaten ausschließlich in einen Tensor geglättet und daraufhin durchlaufen sie das vorher definierte Netzwerk.

Die Trainingsmethode sieht dann wie folgt aus:

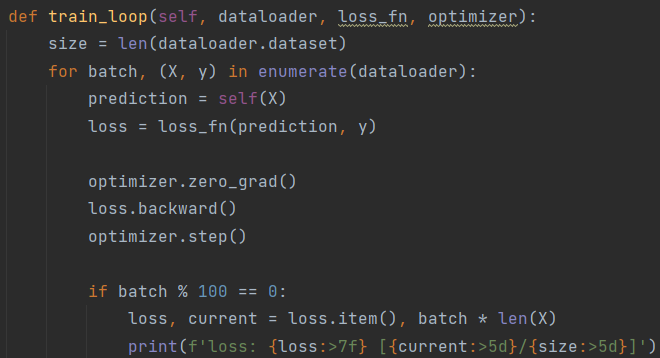


Abbildung 4: PyTorch Trainingsmethode

Sie nimmt einen Dataloader, eine Verlustfunktion und eine Optimierungsfunktion entgegen.  
Daraufhin durchläuft sie die Trainingsdaten aus dem Dataloader, welche das Eingabebild und den erwarteten Ausgabewert enthalten. Diese Daten durchlaufen dann einzeln das Netzwerk, welches eine Vorhersage über die präsentierten Ziffern gibt. Diese Vorhersage durchläuft dann die Verlustfunktion, welche bestimmt wie gut sich das Netzwerk dabei verhält die Daten zu erraten. Zu Beginn werden die Verluste sehr hoch ausfallen, da das Netzwerk praktisch nur zufällig eine Entscheidung trifft. Um dies zu verbessern werden die Verluste in das Netzwerk Rückpropagiert, das heißt die Gewichtungen des Netzwerkes werden leicht angepasst damit das Netzwerk näher an den Erwartungswert gelangt bei seiner Entscheidung. Jeder Trainingslauf hilft somit, die Genauigkeit der Entscheidungen des Netzwerkes zu verbessern.

Um nun jedoch die Effektivität des Netzwerkes zu überprüfen, wird eine Testmethode durchlaufen, die Daten überprüft an denen das Netzwerk nicht trainieren kann.

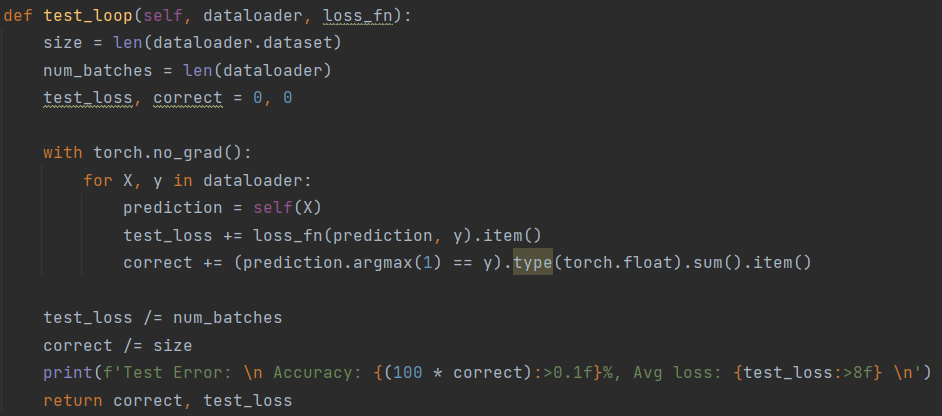


Abbildung 5: PyTorch Testmethode

Diese Methode ist ähnlich wie die Trainingsmethode aufgebaut, jedoch mit dem wesentlichen Unterschied, dass sie nur Testdaten und die Verlustfunktion entgegennimmt. Somit trägt diese Methode nicht der Verbesserung des Netzwerkes bei, sondern ausschließlich der Bemessung der Leistung des Netzwerkes an Daten mit denen es nicht trainiert wurde.  
Die Implementation besteht funktional wesentlich nur aus dem Abruf der Testdatensätze, gefolgt von dem Durchlauf dieser durch das trainierte Netzwerk, gefolgt von einer Auswertung der Genauigkeit anhand der gegebenen Verlustfunktion.

Damit ist unser torch.nn.Module vollständig und es beinhaltet seine eigenen Trainings- und Testmethoden. Diese Klasse hat nun den Vorteil, dass viele gleichartige Netze erstellt werden können ohne, dass diese Ressourcen teilen müssen und somit völlig unabhängig voneinander agieren können.

Nun wird natürlich noch weiterer Quellcode benötigt um diese Klasse tatsächlich auch zu nutzen.  
Zuerst müssen die Datensätze geladen werden. Dies geschieht indem man zuerst die Datensätze aus Torchvision abruft und darauffolgend mithilfe von Dataloadern in Pakete aufteilt und diese ggf. durchmischt.



Abbildung 6: PyTorch Datensatz laden

In diesem Beispiel werden die Trainings- und Testdatensätze der MNIST Datenbank zuerst heruntergeladen, wenn nötig, und dann in separate Dataloader zu Arbeitspaketen mit jeweils 64 Bildern gebündelt, welche dann durchgemischt werden.

Als nächstes wird einerseits die Klasse für das neuronale Netzwerk initialisiert und auf unser Zielgerät übertragen und andererseits die Verlust- und Optimierungsfunktionen bestimmt.

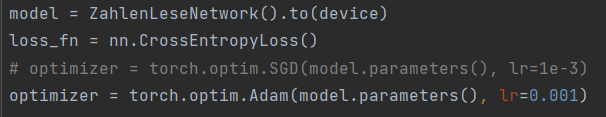


Abbildung 7: PyTorch Verlust- und Optimierungsfunktion

Ich habe hier als Verlustfunktion die mitgelieferte CrossEntropyLoss Funktion verwendet und zuerst als Optimierungsfunktion eine SGD Funktion genutzt, bin jedoch dann auf Adam umgestiegen da diese Funktion wesentlich schneller Fortschritte erzielt.

Zuletzt werden dann die Trainings- und Testmethode in einer Schleife wiederholt aufgerufen, bis entweder eine gewünschte Genauigkeit oder Durchlaufanzahl erreicht wurde.

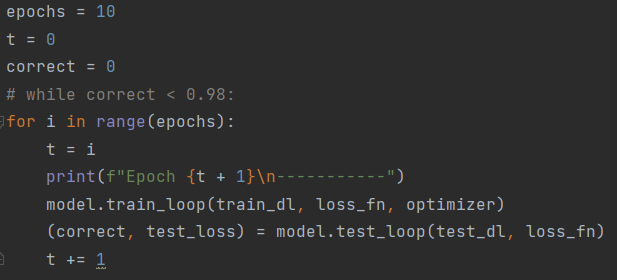


Abbildung 8: PyTorch Ausführungsschleife

Für dieses Beispiel wird die Schleife Zehn mal durchlaufen wodurch letztendlich folgendes Ergebnis zustande kommt:

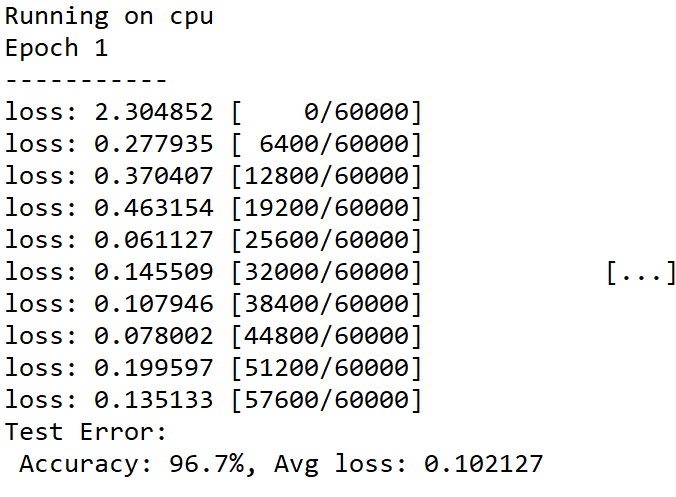


Abbildung 9: PyTorch Ergebnis 1. Epoche

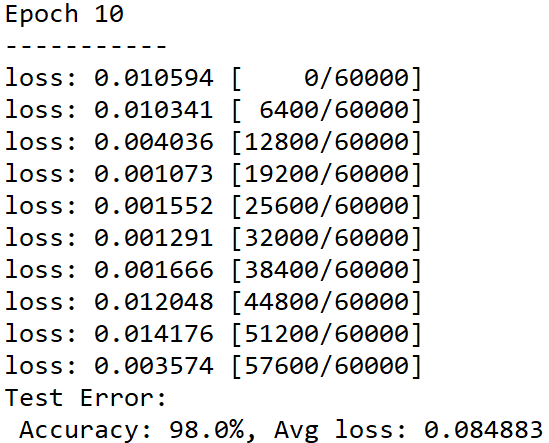


Abbildung 10: PyTorch Ergebnis 10. Epoche

## 3.2. TensorFlow

Der TensorFlow Code ist dem Fortgeschrittenem Beispiel nachempfunden, welches man hier finden kann: https://github.com/tensorflow/docs/blob/b5803ba0b65ad008da7dfb3932cf3cdbdfa917b3/site/en/tutorials/quickstart/advanced.ipynb

Auch in TensorFlow starten wir zuerst mit dem importieren der benötigten Bibliotheken und Namespaces.

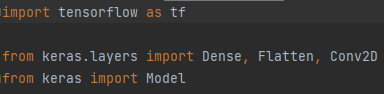


Abbildung 11: TensorFlow Bibliotheken importieren

Diese sind einmal TensorFlow selbst, ein Netzwerkmodul aus Keras und verschiedene Module für die Ebenen des neuronalen Netzwerkes.

Danach erstellen wir wieder das gewünschte neuronale Netz, welches dann wie folgt aussieht:

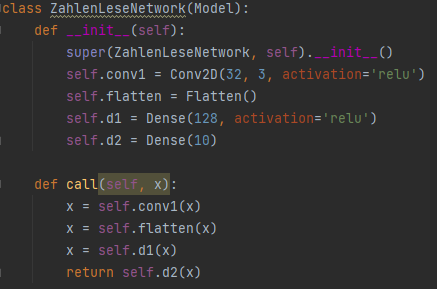


Abbildung 12: TensorFlow neuronales Netzwerk definieren

Hier definieren wir Zuerst eine Conv2D Ebene welche das Eingabebild verarbeitet und dann verarbeiten wir das Bild weiter bis wir nur noch 10 Ausgänge erreichen.

Als nächstes definieren wir die Verlust- und Optimierungsfunktion, sowie die dazugehörigen Metriken.

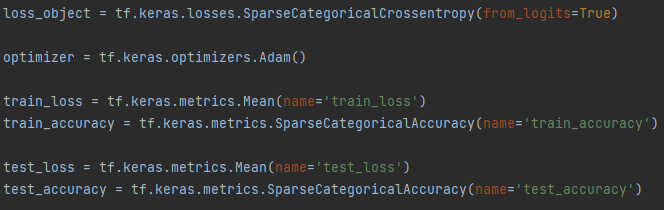


Abbildung 13: TensorFlow Verlust- und Optimierungsfunktion

Auch hier wird eine Crossentropy Verlustfunktion und die Adam Optimierungsfunktion verwendet.

Als nächstes folgen die Trainingsfunktion und die Testfunktion.

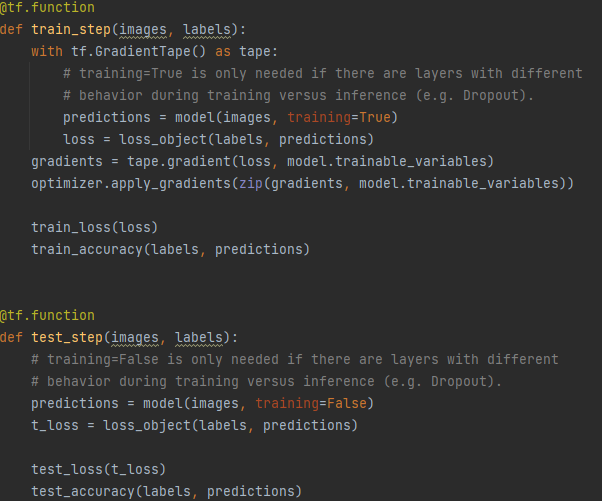


Abbildung 14: TensorFlow Trainings- und Testfunktionen

Auch hier sind diese sehr ähnlich, mit dem einzigen Unterschied, dass beim Testen das Netzwerk nicht trainiert wird.  
Im Gegensatz zu dem PyTorch Beispiel sind diese Funktionen nicht Bestandteil des eigentlichen Netzwerks.

Zuletzt kommt die eigentliche Nutzung des Netzwerks, beginnend mit dem Laden der MNIST Daten.

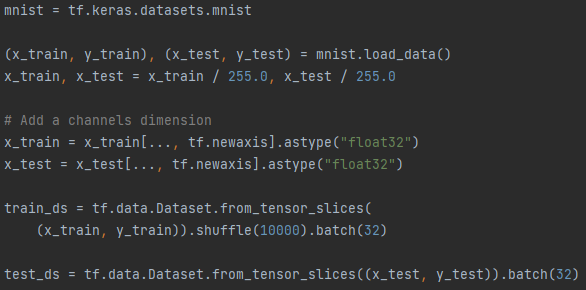


Abbildung 15: TensorFlow Datensatz laden

In TensorFlow werden die Datensätze nicht durch Dataloader aufgebaut, sondern durch spezielle Funktionen die eine Sammlung an Tensors direkt in einen Datensatz umwandeln.

Als letztes wird wiederholt der Trainings- und Testzyklus ausgeführt.

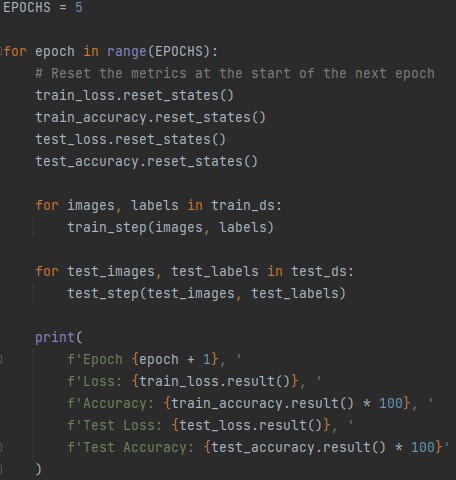


Abbildung 16: TensorFlow Ausführungsschleife

Somit ist das Netzwerk dann vollständig trainiert und bereit eingesetzt zu werden.

Das Ergebnis beim Ausführen ist folgendes:

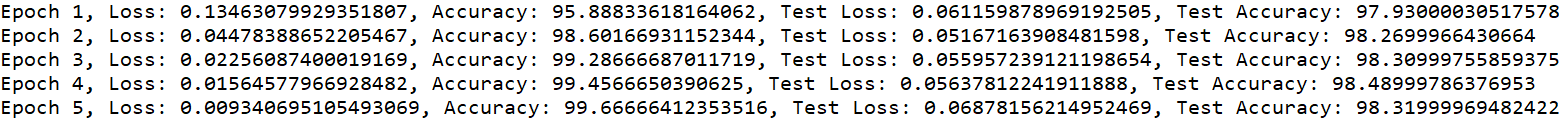


Abbildung 17: TensorFlow Ergebnis

4. Fazit

Sowohl PyTorch als auch TensorFlow sind in ihrer Handhabung sehr einfach und bieten mittlerweile alle nötigen Bibliotheken und Funktionen von Haus aus um relativ einfach neurale Netzwerke zu trainieren. Während PyTorch alle Funktionen in der eigentlichen Hauptbibliothek bietet, benötigt TensorFlow das Einbinden von Bibliotheken wie Keras um eine leichte Handhabung und Erstellung von neuralen Netzwerken zu ermöglichen. Dies ist jedoch keineswegs ein Problem, da Keras beispielsweise in der Standardinstallation enthalten ist und immer weiter Bestandteil von TensorFlow wird.   
Generell konnte ich wenige Unterschiede im praktischen Einsatz beider Frameworks finden außer der Tatsache, dass TensorFlow mehrere Programmiersprachen bietet in denen entwickelt werden kann.

1. (PyTorch, 2022) [↑](#footnote-ref-2)
2. (TensorFlow, 2022) [↑](#footnote-ref-3)